SY20-0885-0

IBM Virtual Machine Facility/370:

System Logic and Problem Determination Guide

1976

Page Missing From

Original Document

Page Missing From
Original Document

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 Facility/370 (VM/370).

CMS/DOS is part of the CMS system and is not a separate system. The term CMS/DOS is used in this publication as a concise way of stating that the DOS simulation mode of CMS is currently active; that is, the CMS command

SET DOS ON

has been previously issued.

The phrase "CMS file system" refers to disk files that are in CMS's 800-byte block format; CMS's VSAM data sets are not included.

A system failure is usually accompanied by a dump of processor storage. The dump can occur by means of an automatically invoked dump program, or a standalone dump program. An example of a standalone dump program is:

BPS Storage Print Program, No. 360-UT-056

HOW THIS MANUAL IS ORGANIZED

This manual contains five sections:

"Section 1. Introduction" contains debugging information about error conditions that may occur within VM/370. This debugging information tells you what to do about ABENDs, loops, wait states, and incorrect output. Section 1 also contains a brief description of three of the VM/370 components. The components that are described are: the VM/370 Control Program (CP), the Conversational Monitor System (CMS), and the Remote Spooling Communications Subsystem (RSCS).

"Section 2. Method of Operation and Program Organization" contains the functions and relationships of the program routines in VM/370. Section 2 indicates the program operation and organization in a general way to serve as a guide in understanding the programming of VM/370. It is not meant to be a detailed analysis of VM/370 programming and cannot be used as such.

"Section 3. Directories" contains a description of all the assemble modules in CP, CMS, and RSCS. It also contains extensive cross-references between modules and labels within a VM/370 component.

"Section 4. Diagnostic Aids" contains debugging commands for problem solving, wait state and ABEND codes, error codes, return codes, and information about the DASD Dump Restore Program.

"Section 5. Appendixes" contains reference information that may be useful when debugging VM/370, such as: the VM/370 programming restrictions, the CMS ZAP Service Program, and the VM/370 coding conventions and equate symbols.

HOW TO USE THIS MANUAL

- Use the problem determination part of Section 1 to help you to determine what type of error has occurred. Write down all error messages, ABEND codes and return codes, and obtain a storage dump.
- Consult the _VM/370: System Messages_ for information about the error message, ABEND code, or return code. The _VM/370: System Messages_ also contains extensive cross-reference information that may be helpful to you.
- Isolate the component of VM/370 in which the problem occurred.
- Use the list of restrictions in "Section 5. Appendixes" to be certain that the operation that was being performed was valid.
- Use "Section 3. Directories" and use the _VM/370: Data Areas and Control Block Logic_ to help you to isolate the problem.
- Use "Section 2. Method of Operation and Program Organization" if necessary, to understand the operation that was being performed.

PREREQUISITE PUBLICATIONS

IBM Virtual Machine Facility/370:

Introduction, Order No. GC20-1800

Terminal User's Guide, Order No. GC20-1810

Operator's Guide, Order No. GC20-1806

CP Command Reference for General Users, Order No. GC20-1820

CMS Command and Macro Reference, Order No. GC20-1818

IBM System/360 Principles of Operation, Order No. GA22-6821

IBM System/370 Principles of Operation, Order No. GA22-7000

IBM OS/VS and VM/370 Assembler Programmer's Guide, Order No. GC33-4021

IBM OS/VS, DOS/VS, and VM/370 Assembler Language, Order No. GC33-4010

COREQUISITE PUBLICATIONS

IBM Virtual Machine Facility/370:

Data Areas and Control Blocks Logic, Order No. SY20-0884.

System Messages, Order No. GC20-1808

Operating Systems in a Virtual Machine, Order No. GC20-1821

Remote Spooling Communications Subsystem (RSCS) User's Guide, Order No. GC20-1816

FIGURES

INTRODUCTION TO DEBUGGING

The VM/370 Control Program (CP) manages the resources of a single computer such that multiple computing systems appear to exist. Each "virtual computing system", or virtual machine, is the functional equivalent of an IBM System/370. The person trying to determine the cause of a VM/370 software problem must consider three separate areas:

- The Control Program (CP), which controls the resources of the real machine.
- The virtual machine operating system running under the control of CP, such as CMS, RSCS, OS, DOS, or VM/370.
- The problem program, which executes under the control of a virtual machine operating system.

Note: For information about the Interactive Problem Control System refer to the *VM/370: Interactive Problem Control System (IPCS) User's Guide*, Order No. GC20-1823.

Once the area causing the problem is identified, the appropriate person should use all available information and determine the cause of the problem. The IBM Program Systems Representative (PSR) or a system programmer handles all problems with CP, Conversational Monitor System (CMS), Remote Spooling Communication Subsystem (RSCS), and Interactive Problem Control System (IPCS); information that is helpful in debugging CP, CMS, and RSCS is contained in this publication. The applications programmer handles all problem program errors; techniques for applications program debugging are found in the *VM/370: CMS User's Guide*.

If the problem is caused by a virtual machine operating system other than CMS and RSCS, refer to the publications pertaining to that operating system for specific information. However, use the CP debugging facilities, such as the CP commands, to perform the recommended debugging procedures discussed in the publication that pertains to the other operating system. The IBM PSR or a system programmer handles problems with virtual machine operating systems.

If it becomes necessary to apply a PTF (Program Temporary Fix) to a component of VM/370, refer to "Appendix J: Applying PFTs" for detailed information on applying PTFs.

HOW TO START DEBUGGING

Before you can correct any problem, you must recognize that one exists. Next, you must identify the problem, collect information and determine the cause so that the problem can be fixed. When running VM/370, you must also decide whether the problem is in CP, the virtual machine, or the problem program.

A good approach to debugging is:

1. Recognize that a problem exists.
2. Identify the problem type and the area affected.
3. Analyze the data you have available, collect more data if you need it, then isolate the data that pertains to your problem.
4. Finally, determine the cause of the problem and correct it.

DOES A PROBLEM EXIST?

There are four types of problems:

- ABEND (Abnormal End)
- Unexpected results
- Loop
- Wait state

ABEND (Abnormal End)

The abnormal end is the most easily identified problem. An abnormal termination causes an error message.

Unexpected Results

Unexpected results, such as missing or incorrect output, or incorrect format, is another easily identified problem.

Loop and Wait State

Unproductive processing time is a problem not easily recognized, especially in a time-sharing environment. When you are using VM/370, you are usually sitting at a terminal and do not have the lights of the CPU control panel to help you recognize this type of problem.

You may have a looping condition if your program takes longer to execute than you anticipated. Check your output. If the number of output records or print lines is greater than expected, the output may really be the same information repeated many times. Repetitive output usually indicates a program loop.

Another way to identify a loop is to periodically examine the current PSW. If the PSW instruction address always has the same value, or if the instruction address has a series of repeating values, the program probably is looping.

A wait state may exist if your program is taking longer to execute then expected. To identify a probable wait state, display the current PSW on the terminal. Periodically, issue the CP command

QUERY TIME

and compare the elapsed processing time. When the elapsed processing time does not increase, the wait state probably exists.

Figures 1-10 help you to identify problem types and the areas where they may occur.

ANALYZING THE PROBLEM

Once the type of problem is identified, the cause of it must be determined. There are recommended procedures to follow. These procedures are helpful, but do not identify the cause of the problem in every case. Be resourceful. Use whatever data you have available. If the cause of the problem is not found after the recommended debugging procedures are followed, it may be necessary to undertake the tedious job of checking through listings at your desk.

See the VM/370: CMS User's Guide for information on using VM/370 facilities to debug a problem program.

USING VM/370 FACILITIES TO DEBUG

Once the problem and the area where it occurs is identified, you can gather the information needed to determine the cause of the problem. The type of information you want to use varies with the type of problem. The tools used to gather the information vary depending upon the area in which the problem occurs. For example, if looping is the problem, you should examine the PSW. For a CP loop, you must use the operator's console to display the PSW, but for a virtual machine loop you can display the PSW via the CP DISPLAY command.

The following shows specific debugging procedures for the various error conditions. The procedures tell you what to do and what debugging tool to use. For details on how to invoke and use the debugging tools refer to "CP Commands For Debugging", "CMS Commands For Debugging", and "Debugging With CMS" in Section 4.

CP ABNORMAL TERMINATION

When CP abnormally terminates, a dump is taken. This dump can be directed to a tape or printer or dynamically allocated to a DASD. The output device for a CP ABEND dump is specified by the CP SET command. See "ABEND Dumps" in this section for a description of the SET and VMFDUMP commands.

Use the dump to find what caused CP to terminate. Find why the system abnormally terminated and then see how the condition can be corrected. See "Reading CP ABEND Dumps" in this section for detailed information on reading a CP ABEND dump.

REASON FOR THE ABEND: CP terminates and takes an abnormal termination dump under three conditions:

- Program Check in CP

  Examine the PROPSW and INTPR fields in the Prefix Storage Area (PSA) to determine the failing module.

- Module Issuing an SVC 0

  Examine the SVC old PSW (SVCOPSW) and ABEND code (CPABEND) fields in the prefix storage area to determine the module that issued the SVC 0 and the reason it was issued.

  CPABEND contains an abnormal termination code. The first three characters identify the failing module (for example, ABEND code BLD001 indicates DMKBLD is the failing module).

- Pressing SYSTEM RESTART on CPU Console

  Examine the old PSW at location X'08' to find the location of the instruction that was executing when SYSTEM RESTART was pressed. The operator presses SYSTEM RESTART when CP is in a disabled wait state or loop.

PROCEDURE WHEN CP ABEND OCCURS: The information in low storage tells you the status of the system at the time CP terminated. Status information is stored in the CPSTAT field of the PSA. You should be able to tell the module that was executing by looking at the PSA. See "Save Area Conventions" in this section and refer to the appropriate save area (SAVEAREA, BALRSAVE, or FREESAVE) to see how that module started to execute.

Examine the real and virtual control blocks to find the status of I/O operations. The PSA is described in the VM/370: Data Areas and Control Block Logic.

Examine the CP internal trace table. This table can be extremly helpful in determining the events that preceded the ABEND. The CP internal trace table description in Section 4 tells you how to use the trace table.

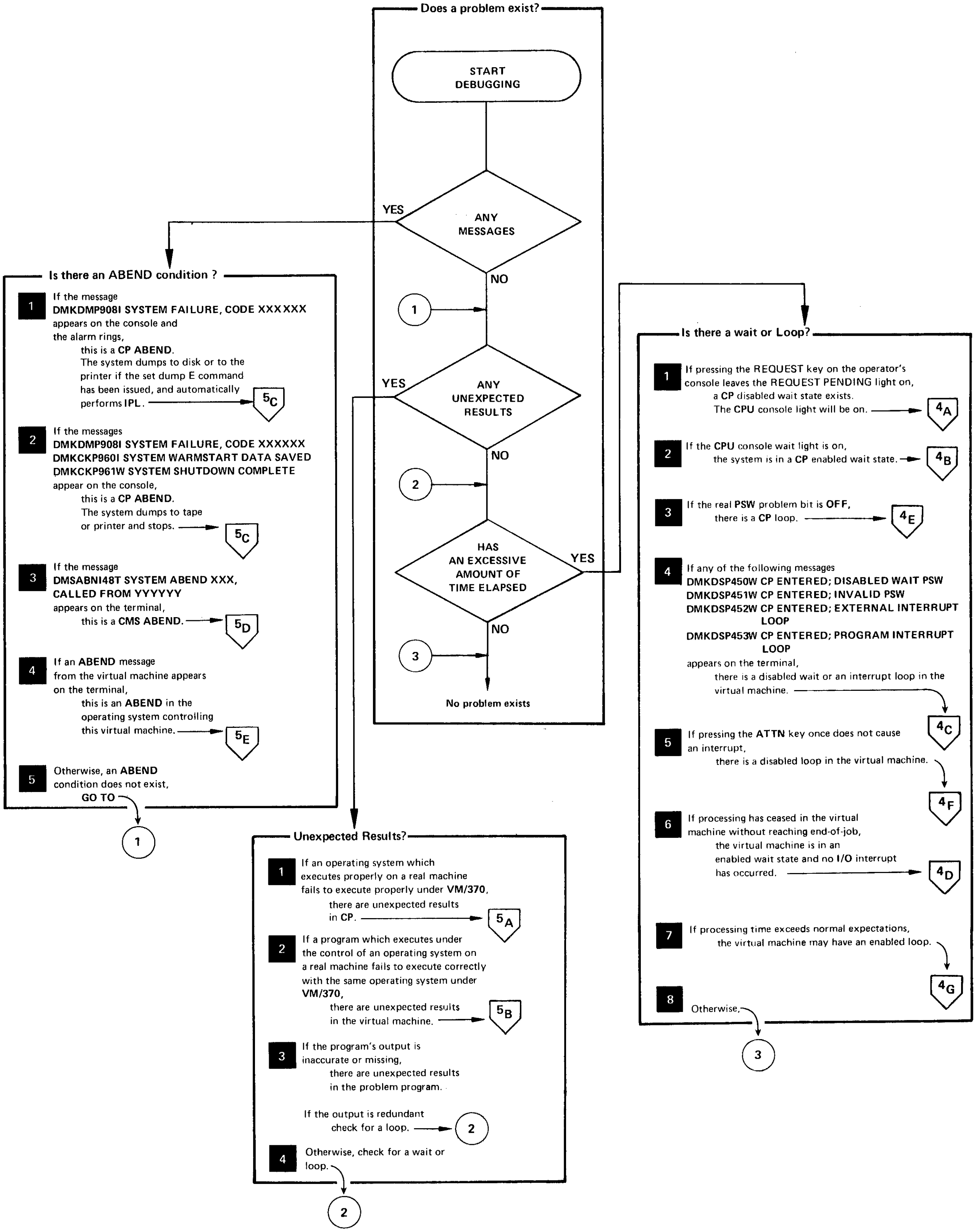


Figure 1. Does a Problem Exist?

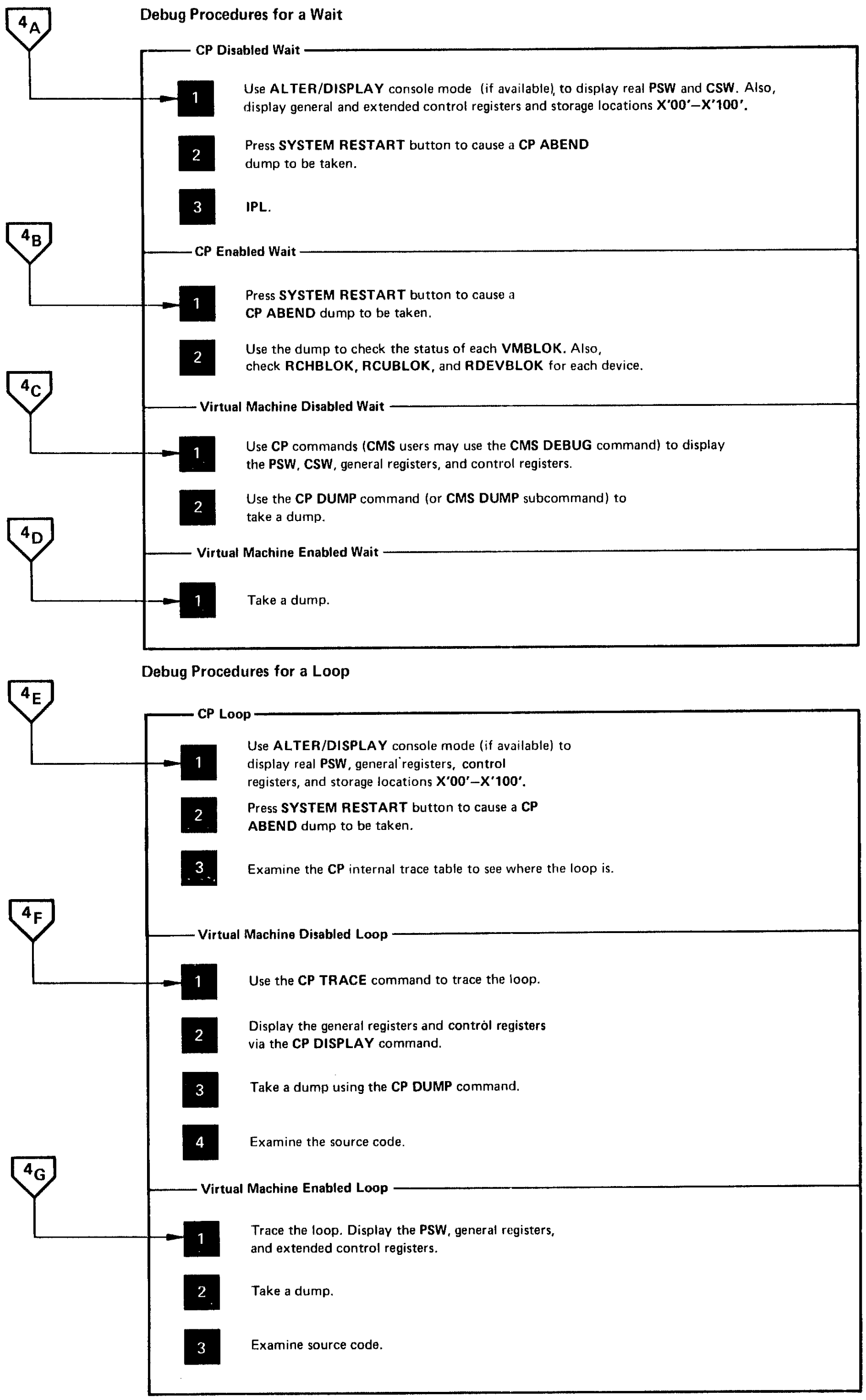


Figure 2. Debug Procedures for Waits and Loops

5A

5B

Debug Procedures for Unexpected Results

Unexpected Results in CP

1. Check that the program is not violating any **CP** restrictions.
2. Check that the program and operating system running on the virtual machine are exactly the same as those that ran on the real machine.
3. Use the **CP TRACE** command to trace **CCWs**, **SIOs**, and interrupts. Look for an error in **CCW** translation or interrupt reflection.
4. If disk **I/O** error, use the **CP DDR** (**DASD** Dump Restore) program to print the contents of any disk.

Unexpected results in a virtual machine

1. Check that the program executing on the virtual machine is exactly the same as the one that ran on the real machine.
2. Make sure that operating system restrictions are not violated.
3. Use **CP TRACE** to trace all **I/O** operations.

5C

5D

5E

Debug Procedures for an ABEND

CP ABEND

1. Find out why **CP** abnormally terminated. Examine the **PROPSW**, **INTPR**, **SVCOPSW**, and **CPABEND** fields in the **PSA** from the dump.
2. Identify the module that caused the **ABEND**. Examine the **SAVEAREA**, **BALRSAVE**, and **FREESAVE** areas of the dump.
3. If **I/O** operation, examine the real and virtual **I/O** control blocks.

CMS ABEND

1. Determine reason for **ABEND** from code in **ABEND** message **DMSABN148T**.
2. Enter debug environment or **CP** console function mode to use the commands, to display the **PSW**, and to examine low storage areas:
   - **LASTLMOD** and **LASTTMOD**
   - **LASTCMND** and **PREVCMND**
   - **LASTEXEC** and **PREVEXEC** and **DEVICE**

   Look at the last instruction executed.
   Take dump if need be.

Virtual Machine ABEND (other than CMS)

1. Examine dump, if there is one.
2. Use **CP** commands to examine registers and control words.
3. Use **CP TRACE** to trace the processing up to the point where the error occurred.

Figure 3. Debug Procedures for Unexpected Results and an ABEND

The values in the general registers can help you to locate the IOBLOK, VMBLOK, and the save area. Refer to "Reading CP ABEND Dumps" in this section for detailed information on the contents of the general registers.

In the PSA, if the program check old PSW (PROPSW) or the SVC old PSW (SVCOPSW) points to an address beyond the end of the resident nucleus, the module that caused the ABEND is a pageable module. Refer to "Reading CP ABEND Dumps" in this section to find out how to identify that pageable module. Use the CP load map that was created when the VM/370 system was generated to find the address of the end of the resident nucleus.

CP TERMINATION WITHOUT A DUMP

Two types of severe machine checks can cause the VM/370 control program to terminate:

- An unrecoverable machine check in the control program
- A machine check that cannot be diagnosed

A machine check error cannot be diagnosed if either the machine check old PSW or the machine check interruption code is invalid. These severe machine checks cause CP to terminate, but no dump is taken since the error is recorded on the error recording cylinders. The system is automatically restarted and a message is issued identifying the machine check error.

If an unrecoverable machine check occurs in CP, the message

    DMKMCH610I MACHINE CHECK SUPERVISOR DAMAGE

appears on the CPU console. CP is terminated and automatically restarted.

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

    DMKMCH611I MACHINE CHECK SYSTEM INTEGRITY
               LOST

appears on the CPU console, CP is terminated and automatically restarted.

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

CMS ABNORMAL TERMINATION

When CMS abnormally terminates, the following error message appears on the terminal:

    DMSABN148T SYSTEM ABEND xxx CALLED
               FROM yyyyyy

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

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

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

REASON FOR THE ABEND: First determine the reason CMS abnormally terminated. There are four types of CMS abnormal terminations:

- Program Exception

  The DMSITP routine gets control whenever a hardware program exception occurs. If a routine other than a SPIE exit routine is in control, DMSITP issues the message

      DMSITP141T xxxxxxxx EXCEPTION OCCURRED
                 AT xxxxxx IN ROUTINE xxxxxxxx

  and invokes DMSABN (the ABEND routine). The ABEND code is 0Cx, where x is the program exception number (0-F). The possible programming exceptions are:

Code	Meaning
0	Imprecise
1	Operation
2	Privileged operation
3	Execute
4	Protection
5	Addressing
6	Specification
7	Decimal data
8	Fixed-point overflow
9	Fixed-point divide
A	Decimal overflow
B	Decimal divide
C	Exponent overflow
D	Exponent underflow
E	Significance
F	Floating-point divide

- ABEND Macro

  Control is given to the DMSSAB routine whenever a user routine executes the ABEND macro. The ABEND code specified in the ABEND macro appears in the abnormal termination message DMSABN148T.

- Halt Execution (HX)

  Whenever the virtual machine opertor signals an attention interruption and enters HX, CMS terminates and issues "CMS".

* System ABEND

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

  The format of the DMSABN macro is:

```
┌─────────────────────────────────────────────────┐
│        │      │      ┌            ┌   ┐┐        │
│[label] │DMSABN│code  │,TYPCALL=   │SVC ││        │
│        │      │(reg) │            │BALR││        │
│        │      │      └            └   ┘┘        │
└─────────────────────────────────────────────────┘
```

where:

label — is any valid Assembler language label.

code — is the abnormal termination code (0-FFF) that appears in the DMSABN148T system termination message.

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

TYPCALL= — specifies how control passes to the TYPCALL=BALR abnormal termination routine, DMSABN.

TYPCALL=SVC — Routines that do not reside in the nucleus should use TYPECALL=SVC to generate CMS SVC 203 linkage.

TYPCALL=BALR — Nucleus-resident routines specify TYPCALL=BALR to generate a direct branch to DMSABN.

If a CMS SVC handler abnormally terminates, it sets an ABEND flag and stores an ABEND code in NUCON (the CMS nucleus constant area). After the SVC handler has finished processing, the ABEND condition is recognized. The DMSABN ABEND routine issues the ABEND message, DMSABN148T, with the ABEND code stored in NUCON.

<u>PROCEDURE WHEN CMS ABEND OCCURS</u>: After a CMS ABEND, CMS provides two courses of action. In addition, you can enter the CP command mode and use CP's debugging facilities by signalling attention.

The two courses of action available in CMS are:

* Issue the DEBUG command and enter the debug environment. After using all the DEBUG subcommands that you need, exit from the debug environment. Then, either issue the RETURN command to return to DMSABN so that ABEND recovery occurs, or issue the GO command to resume processing at the point the ABEND occurred.

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

The ABEND recovery function performs the following:

* The SVC handler, DMSITS, is re-initialized, and all stacked save areas are released.

* "FINIS * * *" is invoked by means of SVC 202, to close all files, and to update the master file directory.

* If the EXECTOR module is in real storage, it is released.

* All link blocks allocated by DMSSLN are freed.

* All FCB pointers are set to zero.

* All user storage is released.

* The amount of system free storage which should be allocated is computed. This value is compared to the amount of free storage that is actually allocated.

* The console input stack is purged.

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

```
DMSABN149T xxxx DOUBLEWORDS OF SYSTEM
           STORAGE HAVE BEEN DESTROYED
```

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

```
DMSABN150W nnn (HEX xxx) DOUBLEWORDS OF
           SYSTEM STORAGE WERE NOT
           RECOVERED
```

appears on the terminal.

<u>A DEBUGGING PROCEDURE</u>: When a CMS ABEND occurs, you probably want to use the DEBUG subcommands or CP commands to examine the PSW and certain areas of low storage. Refer to "CMS Debugging Commands" in Section 4 for detailed description of how to use the CMS DEBUG subcommands. See "CP Commands Used to Debug the Virtual Machine" and "CP Commands Used to Debug CP" in Section 4 for a detailed description of how to use the CP commands. Also refer to Figure 12 for a comparison of the CP and CMS debugging facilities.

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

1. Display the PSW. (Use the CP DISPLAY command or CMS DEBUG PSW subcommand.) Compare the PSW instruction address to the current CMS load map to determine the module that caused the ABEND. The CMS storage-resident nucleus routines are in fixed storage locations.

   Also check the interruption code in the PSW.

2. Examine areas of low storage. The information in low storage can tell you more about the cause of the ABEND.

Field	Contents
LASTLMOD	Contains the name of the last module loaded into storage via the LOADMOD command.
LASTTMOD	Contains the name of the last module loaded into the transient area.
LASTCMND	Contains the name of the last command issued.
PREVCMND	Contains the name of the next-to-the-last command issued.
LASTEXEC	Contains the name of the last EXEC procedure.
PREVEXEC	Contains the name of the next-to-last EXEC procedure.
DEVICE	Identifies the device that caused the last I/O interrupt.

The low storage areas examined depend on the type of ABEND.

3. Once you have identified the module that caused the ABEND, examine the specific instruction. Refer to your listing.

4. If you have not identified the problem at this time, take a dump by issuing the DEBUG DUMP subcommand. Refer to "Reading CMS ABEND Dumps" in this section for information on reading a CMS dump. If you can reproduce the problem, try the CP or CMS tracing facilities.

VIRTUAL MACHINE ABEND (OTHER THAN CMS)

The abnormal termination of an operating system (such as OS or DOS) running under VM/370 appears the same as a similar termination on a real machine. Refer to publications for the specified operating system for debugging information. However, all of the CP debugging facilities may be used to help you gather the information you need. Because certain operating systems (OS/VS1, OS/VS2, and DOS/VS) manage their own virtual storage, CP commands that examine or alter virtual storage locations should be used only in virtual=real storage space with OS/VS1, OS/VS2, and DOS/VS.

If a dump was taken, it was sent to the virtual printer. Issue a CLOSE command to the virtual printer to print the dump on the real printer.

If you choose to run a standalone dump program to dump the storage in your virtual machine, be sure to specify the NOCLEAR option when you issue the CP IPL command. At any rate, a portion of your virtual storage is overlaid by CP's virtual IPL simulation.

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

VM/370 may terminate or reset a virtual machine if a nonrecoverable channel check or machine check occurs in that virtual machine. Hardware errors usually cause this type of virtual machine termination.

One of the following messages appears on the CPU console:

```
DMKMCH616I MACHINE CHECK; USER userid
           TERMINATED

DMKCCH604I CHANNEL ERROR; DEV xxx;
           USER userid; MACHINE RESET
```

Message	Type of ABEND
(Alarm rings) DMKDMP908I SYSTEM FAILURE CODE xxxxxx Optional Messages: DMKDMP905W SYSTEM DUMP FAILURE; PROGRAM CHECK DMKDMP906W SYSTEM FAILURE; MACHINE CHECK, RUN SEREP DMKDMP907W SYSTEM DUMP FAILURE; FATAL I/O ERROR	CP ABEND, system dumps to disk. Restart is automatic. If the dump program encounters a a program check, machine check or fatal I/O error, a message is issued indicating the error. CP enters the wait state with code 3 in the PSW.
DMKCKP900W SYSTEM RECOVERY FAILURE; PROGRAM CHECK DMKCKP901W SYSTEM RECOVERY FAILURE; MACHINE CHECK, RUN SEREP DMKCKP902W SYSTEM RECOVERY FAILURE; FATAL I/O ERROR - NUCL CYL - WARM CYL DMKCKP904W SYSTEM RECOVERY FAILURE; INVALID WARM START DATA DMKCKP910W SYSTEM RECOVERY FAILURE; INVALID WARM START CYLINDER DMKCKP911W SYSTEM RECOVERY FAILURE; WARM START AREA FULL	If the checkpoint program encounters a program check, a machine check, a fatal I/O error or an error relating to a certain warm start cylinder or warm start data conditions, a message is issued indicating the error and CP enters the wait state with code 7 in the PSW.
DMKWRM902W SYSTEM RECOVERY FAILURE; FATAL I/O ERROR DMKWRM903W SYSTEM RECOVERY FAILURE; VOLID xxxxx ALLOCATION ERROR CYLINDER xxx DMKWRM904W SYSTEM RECOVERY FAILURE; INVALID WARM START DATA DMKWRM909W SYSTEM RECOVERY FAILURE; VOLID xxxxxx NOT MOUNTED DMKWRW909W SYSTEM DUMP DEVICE; NOT-READY	If the warm start program encounters a severe error, a message is issued indicating the error and CP enters the wait state code 9 in the PSW.
DMKDMP908I SYSTEM FAILURE, CODE xxxxxx DMKCKP960I SYSTEM WARM START DATA SAVED DMKCKP961W SYSTEM SHUTDOWN COMPLETE	CP ABEND, system dumps to tape or printer. The system stops; the operator must IPL the system to start again.

Figure 4. ABEND Messages (Part 1 of 2)

Message	Type of ABEND
<u>Optional Messages</u>	
DMKDMP905W SYSTEM DUMP FAILURE; PROGRAM CHECK DMKDMP906W SYSTEM DUMP FAILURE; MACHINE CHECK, RUN SEREP DMKDMP907W SYSTEM DUMP FAILURE; FATAL I/O ERROR	If the dump program encounters a program check, a machine check or fatal I/O error, a message is issued indicating the error. CP enters the wait state with code 3 in the PSW. If the dump cannot find a defined dump device and if no printer is defined for the dump, CP enters a disabled wait state with code 4 in the PSW.
	CP termination with automatic restart when the two messages in the "Messages" column are issued:
DMKMCH610I MACHINE CHECK; SUPERVISOR DAMAGE	The machine check handler encountered an unrecoverable error with the VM/370 control program.
DMKMCH611I MACHINE CHECK; SYSTEM INTEGRITY LOST	The machine check handler encountered an error that connot be diagnosed; system integrity, at this point, is not reliable.
	CP termination occurs without automatic restart when the two messages in the "Messages" column are issued:
DMKCCH603W CHANNEL ERROR, RUN SEREP, RESTART SYSTEM	There was a channel check condition from which the channel check handler could not recover. CP enters the wait state with condition code 2 in the PSW.
DMKCPI955W INSUFFICIENT STORAGE FOR VM/370	The generated system requires more real storage than is available. CP enters the disabled wait state with code 00D in the PSW.
DMSABN148T SYSTEM ABEND xxx CALLED FROM xxxxxx	CMS ABEND; the system accepts commands from the terminal. Enter the DEBUG command and then the DUMP subcommand to have CMS dump storage on the printer.
Others Refer to OS and DOS publication for the abnormal termination messages.	When OS or DOS abnormally terminates on a virtual machine, the messages issued and the dumps taken are the same as they would be if OS or DOS abnormally terminated on a real machine.

Figure 4. ABEND Messages (Part 2 of 2)

Problem Type	Where ABEND Occurs	Distinguishing Characteristics
ABEND	CP ABEND	The alarm rings and the message DMKDMP908I SYSTEM FAILURE, CODE xxxxxx appears on the CPU console. In this instance, the system dump device is a disk, so the system dumps to disk and automatically restarts. If an error occurs in the dump, checkpoint, or warmstart program, CP enters the wait state after issuing one or more of the following messages: DMKDMP905W SYSTEM DUMP FAILURE; PROGRAM CHECK DMKDMP906W SYSTEM DUMP FAILURE; MACHINE CHECK, RUN SEREP DMKDMP907W SYSTEM DUMP FAILURE; FATAL I/O ERROR DMKCKP900W SYSTEM RECOVERY FAILURE; PROGRAM CHECK DMKCKP901W SYSTEM RECOVERY FAILURE; MACHINE CHECK, RUN SEREP DMKCKP902W SYSTEM RECOVERY FAILURE; FATAL I/O ERROR DMKCKP904W SYSTEM RECOVERY FAILURE; INVALID WARM START DATA DMKCKP910W SYSTEM RECOVERY FAILURE; INVALID WARM START CYLINDER DMKCKP911W SYSTEM RECOVERY FAILURE; WARM START AREA FULL DMKWRM902W SYSTEM RECOVERY FAILURE; FATAL I/O ERROR DMKWRM903W SYSTEM RECOVERY FAILURE; VOLID xxxxxx ALLOCATION ERROR CYLINDER xxx DMKWRM904W SYSTEM RECOVERY FAILURE; INVALID WARM START DATA DMKWRM909W SYSTEM RECOVERY FAILURE; VOLID xxxxxx NOT MOUNTED
	CP ABEND	The following messages appear on the CPU console: DMKDMP908I SYSTEM FAILURE, CODE xxxxxx DMKDMP960I SYSTEM WARM START DATA SAVED DMKDMP961W SYSTEM SHUTDOWN COMPLETE The system dumps to tape or printer and stops. The operator must IPL the system to restart. If an error occurs in the dump or checkpoint program CP enters the wait state after issuing one or more of the following messages: DMKDMP905W SYSTEM DUMP FAILURE; PROGRAM CHECK DMKDMP906W SYSTEM DUMP FAILURE; MACHINE CHECK, RUN SEREP DMKDMP907W SYSTEM DUMP FAILURE; FATAL I/O ERROR DMKCKP900W SYSTEM RECOVERY FAILURE; PROGRAM CHECK DMKCKP901W SYSTEM RECOVERY FAILURE; MACHINE CHECK, RUN SEREP DMKCKP902W SYSTEM RECOVERY FAILURE; FATAL I/O ERROR DMKCKP910W SYSTEM RECOVERY FAILURE; INVALID WARM START CYLINDER DMKCKP911W SYSTEM RECOVERY FAILURE; WARM START AREA FULL

Figure 5. ABEND Problem Type (Part 1 of 2)

Problem Type	Where ABEND Occurs	Distinguishing Characteristics
ABEND (Cont.)	CP termination with automatic restart	An unrecoverable machine check error has occurred. One of the following messages: DMKMCH610I MACHINE CHECK SUPERVISOR DAMAGE DMKMCH611I MACHINE CHECK INTEGRITY LOST appears on the CPU console. The system is automatically restarted.
	CP termination without automatic restart	An unrecoverable channel check error has occurred. The message: DMKCCH603W CHANNEL ERROR, RUN SEREP, RESTART SYSTEM appears on the CPU console, and CP enters the wait state.
	Virtual machine ABEND (CMS)	The CMS message DMSABM148T SYSTEM ABEND xxx CALLED FROM xxxxxx appears on the terminal. The system stops and waits for a command to be entered on the terminal. To have a dump taken, issue the CMS DEBUG command and then the DUMP subcommand.
	Virtual machine ABEND (other than CMS)	When OS or DOS abnormally terminates on a virtual machine, the messages issued and the dumps taken are the same as they would be if OS or DOS abnormally terminated on a real machine. VM/370 may terminate or reset a virtual machine if a nonrecoverabele channel check or machine check occurs in that virtual machine. One of the following messages appear to the system operator at the CPU console: DMKMCH616I MACHINE CHECK; USER userid TERMINATED DMKCCH604I CHANNEL ERROR; DEV xxx; USER userid; MACHINE RESET Also, the virtual machine user is notified, by one of the following messages, that his machine was terminated or reset: DMKMCH619I MACHINE CHECK; OPERATOR TERMINATED DMKCCH606I CHANNEL ERROR; OPERATOR TERMINATED

Figure 5. ABEND Problem Type (Part 2 of 2)

UNEXPECTED RESULTS

The unexpected results type of errors vary, from operating systems improperly functioning under VM/370 to output printed in the wrong format.

UNEXPECTED RESULTS IN CP

If an operating system executes properly on a real machine but does not execute properly with VM/370, a problem exists. Also, if a program executes properly under the control of a particular operating system on a real machine but does not execute correctly under the same operating system with VM/370, a problem exists.

There are programs (such as time-dependent programs) that CP does not support. Be sure that one of these programs is not causing the unexpected results in CP. Refer to "CP Restrictions" in Section 5 for a list of the restrictions.

Ensure that the program and operating system running on the virtual machine are exactly the same as the one that ran on the real machine. Check for the same:

- Job stream
- Copy of the operating system (and program)
- Libraries

If the problem still is not found, look for an I/O problem. Try to reproduce the problem, tracing all CCWs, SIOs, and interruptions via the CP TRACE command. Compare the real and virtual CCWs from the trace. A discrepancy in the CCWs may indicate that one of the CP restrictions was violated, or that the error occurred in CP.

UNEXPECTED RESULTS IN A VIRTUAL MACHINE

When a program executes correctly under the control of a particular operating system on a real machine but has unexpected results executing under the control of the same operating system with VM/370, a problem exists. You usually find that something was changed in the operating system or problem programs. Check that the job stream, the operating system, and the system libraries are the same.

If unexpected results occur (such as TEXT records interspersed in printed output), you can examine the contents of the system or user disk files. Non-CMS users may execute any of the utility programs, which are included in the operating system they are using to examine and rearrange files. For more details on using the utility programs refer to the specific utilities publication for the operating system running in the virtual machine.

CMS users should use the DASD Dump Restore (DDR) service program to print or move the data stored on direct access devices. The VM/370 DASD Dump Restore (DDR) program can be invoked by the CMS DDR command in a virtual machine controlled by CMS. The DDR program has five functions:

- DUMP — dumps part, or all of the data from a DASD device to magnetic tape.
- RESTORE — transfers data from tapes created by DDR DUMP to a direct access device. The direct access device that the data is being restored to must be the same type of device as the direct access device originally containing that data.
- COPY — copies data from one device to another device of the same type. Data may be reordered, by cylinder, when copied from disk to disk. To copy one tape to another, the original tape must have been created by the DDR DUMP function.
- PRINT — selectively prints the hexadecimal and EBCDIC representation of DASD and tape records on the virtual printer.
- TYPE — selectively displays the hexadecimal and EBCDIC representation of DASD and tape records on the terminal.

CMS users should refer to "Debugging with CMS" in Section 4 for instructions on using the DDR command. "CP Commands for Debugging" in Section 4 contains information about executing the DDR program in a real or virtual machine and a description of the DDR control statements.

Unexpected Results Problem Type	
CP	If an operating system, executes properly on a real machine but not properly with CP, a problem exists. Inaccurate data on disk or system files (such as spool files) could be the cause of the error.
Virtual Machine	If a program executes correctly under the control of a particular operating system on a real machine, but does not execute correctly under the same operating system with VM/370, a problem exists.

Figure 6. Unexpected Results Problem Type

LOOPS

The real cause of a loop usually is an instruction that sets or branches on the condition code incorrectly. The existence of a loop can usually be recognized by the ceasing of productive processing and a continual return of the PSW instruction address to the same address. If I/O operations are involved, and the loop is a very large one, it may be extremely difficult to define, and may even include nested loops. One of the most difficult types of loops to determine is entry to the loop from a wild branch. The problem in loop analysis is finding either the instruction that should open the loop or the instruction that passed control to the set of looping instructions.

CP DISABLED LOOP

The system operator should perform the following sequence when gathering information to find the cause of a CP disabled loop.

1. Use the ALTER or DISPLAY commands to display the real PSW, general registers, control registers, and storage locations X'00' - X'100'.
2. Press the SYSTEM RESTART button to cause an ABEND dump to be taken.
3. Save the information collected for the system programmer or IBM Programming Support Representative.

After the system operator has collected the information, the system programmer or Field Engineering representative examines it. If the cause of the loop is not apparent:

1. Examine the CP internal trace table to determine the modules that may be involved in the loop.
2. If the cause is not yet determined, assume that a wild branch caused the loop entry, and search the source code for this wild branch.

VIRTUAL MACHINE DISABLED LOOP

When a disabled loop is in a virtual machine you cannot communicate with the virtual machine's operating system. This means that signaling attention does not cause an interruption.

To find the cause of a virtual machine disabled loop:

1. Enter the CP console function mode.
2. Use the CP TRACE command to trace the entire loop. Display general and extended control registers via the CP DISPLAY command.
3. Take a dump via the CP DUMP command.
4. Examine the source code.

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

Note: You can IPL a standalone dump program such as the BPS Storage Print to dump the storage of your virtual machine. If you choose to use a standalone dump program, be sure to specify NOCLEAR on the IPL command. Also, be aware that the CP IPL simulation destroys a page of storage in your virtual machine and the standalone dump alters your virtual storage while the CP DUMP command does not.

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

VIRTUAL MACHINE ENABLED LOOP

You should perform the following sequence when locating the cause of an enabled loop:

1. Use the CP TRACE command to trace the entire loop. Display the PSW and the general registers.
2. If your virtual machine has the extended control (EC) mode and the EC option, also display the control registers.
3. Use the CP DUMP command to dump your virtual storage. CMS users can use the DEBUG DUMP subcommand. A standalone dump may be used, but be aware that such a dump destroys the contents of some areas of storage.
4. Consult the source code to search for the faulty instructions, examining previously executed modules, if necessary. Begin by scanning for instructions that set the condition code or branch on it.
5. If the way in which the loop was entered is still undetermined, assume that a wild branch has occurred and begin a search for its origin.

WAIT

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

Loop Problem Type	
CP disabled loop	The CPU console wait light is off. The problem state bit of the real PSW is off. No I/O interruptions are accepted.
CP enabled loop	Condition does not exist.
Virtual machine disabled loop	The program is taking longer to execute than anticipated. Signaling attention from the terminal does not cause an interruption in the virtual machine. You cannot communicate with the virtual machine's operating system by signaling attention.
Virtual machine enabled loop	Excessive processing time often indicates a loop. Use the CP QUERY TIME command to check the elapsed processing time. In CMS, the continued typing of the blip characters indicates that time is elapsing. If time has elapsed, periodically display the virtual PSW and check the instruction address. If the same instruction, or series of instructions continues to appear in the PSW, a loop probably exists.

Figure 7. Loop Problem Type

CP DISABLED WAIT

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

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

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

DMKMCH612W MACHINE CHECK TIMING FACILITIES DAMAGE; RUN SEREP

DMKCCH603W CHANNEL ERROR, RUN SEREP, RESTART SYSTEM

Wait Problem Type	
Type	Distinguishing Characteristics
Disabled CP wait	The CPU wait light is on. Pressing the REQUEST key, or the equivalent action, on the operator's console, leaves the REQUEST PENDING light on. If the message DMKMCH612W MACHINE CHECK TIMING FACILITIES DAMAGE, RUN SEREP appears on the CPU console, a machine check (probable hardware error) caused the CP disabled wait state. If the message DMKCCH603W CHANNEL ERROR, RUN SEREP, RESTART SYSTEM appears on the CPU console, a channel check (probable hardware error) caused the CP disabled wait state. If the message DMKCPI955W INSUFFICIENT STORAGE FOR VM/370 appears on the CPU console, the control program has entered a disabled wait state with code 00D in the PSW. Either the generated system is larger than the real machine size, or a hardware machine malfunction prevents VM/370 from using the necessary amount of storage. If the message DMKPAG415E CONTINUOUS PAGING ERRORS FROM DASD xxx appears on the CPU console, the control program (CP) has entered a disabled wait with code 00F in the PSW. Consecutive hardware errors are occurring on one or more VM/370 paging devices.
Enabled CP wait	The CPU console light is on, but the system accepts interruptions from I/O devices.

Figure 8. CP Wait Problem Type

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

- The generated system is larger than the real machine size

-- or --

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

The message

DMKCPI955W INSUFFICIENT STORAGE FOR VM/370

appears on the CPU console.

If CP cannot continue because consecutive hardware errors are occurring on one or more VM/370 paging devices, the message

DMKPAG415E CONTINUOUS PAGING ERRORS FROM DASD xxx

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

If more than one paging device is available, disable the device on which the hardware errors are occurring and IPL the system again. If the VM/370 system is encountering hardware errors on its only paging device, move the paging volume to another physical device and IPL again.

Note: This error condition may occur if the VM/370 paging volume was not properly formatted.

The following procedure should be used by the system operator to record the needed information.

1. Use the alter/display mode of the CPU console to display the real PSW and CSW. Also, display the general registers and the control registers.

2. Press the SYSTEM RESTART button to get a system ABEND dump.

3. IPL the system.

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

CP ENABLED WAIT

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

Using the dump, examine the VMBLOK for each user and the real device, channel, and control unit blocks. If each user is waiting because of a request for storage and no more storage is available, there is an error in CP. There may be looping in a routine that requests storage. Refer to "Reading CP ABEND Dumps" in this section for specific information on how to analyze a CP dump.

VIRTUAL MACHINE DISABLED WAIT

The VM/370 Control Program does not allow the virtual machine to enter a disabled wait state or certain interrupt loops. Instead, CP notifies the virtual machine operator of the condition with one of the following messages:

DMKDSP450W CP ENTERED; DISABLED WAIT PSW

DMKDSP451W CP ENTERED; INVALID PSW

DMKDSP452W CP ENTERED; EXTERNAL INTERRUPT LOOP

DMKDSP453W CP ENTERED; PROGRAM INTERRUPT LOOP

and enters the console function mode. Use the CP commands to display the following information on the terminal.

- Program status word
- Channel status word
- General registers
- Control registers

Then use the CP DUMP command to take a dump.

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

If CMS is running in the virtual machine, the CMS debugging facilities may also be used to display information, take a dump, or trace the processing. The CMS SVCTRACE and the CP TRACE commands record different information. Figure 11 compares the two.

VIRTUAL MACHINE ENABLED WAIT

If the virtual machine is in an enabled wait state, try to find out why an I/O interruption has not occurred to allow processing to resume.

CP treats the following enabled wait in a virtual machine the same as a disabled wait. If the virtual machine does not have the real timer option and loads a PSW enabled only for external interrupts, CP issues the message

DMKDSP450W CP ENTERED; DISABLED WAIT STATE

Because the virtual timer is not decremented while the virtual machine is in a wait state, it cannot cause the external interrupt. A real timer runs in both the problem state and wait state and an external interruption can cause a virtual machine to resume processing.

Wait Problem Type	
Problem Type	Distinguishing Characteristics
Disabled virtual machine wait	The VM/370 Control Program does not allow a virtual machine to enter a disabled wait state or certain program loops. Instead, CP issues one of the following message: DMKDSP450W CP ENTERED; DISABLED WAIT PSW DMKDSP451W CP ENTERED; INVALID PSW DMKDSP452W CP ENTERED; EXTERNAL INTERRUPT LOOP DMKDSP453W CP ENTERED; PROGRAM INTERRUPT LOOP
Enabled virtual machine wait	A PSW enabled for I/O interruptions is loaded. Nothing happens if an I/O device fails to issue an I/O interruption. If a program is taking longer to execute than expected, periodically issue the CP command, QUERY TIME. If the processing time remains unchanged, probably a virtual machine enabled wait exists. CMS types a blip character for every two seconds of elapsed processing time. If the program does not end and blip characters stop typing, an enabled wait state probably exists.

Figure 9. Virtual Machine Wait Problem Type

RSCS VIRTUAL MACHINE DISABLED WAIT

Three disabled wait conditions can occur during the operation of the RSCS component of VM/370. They can result from either hardware malfunctions or system generation errors. CP notifies the RSCS operator of the wait condition by issuing the message

DMKDSP450W CP ENTERED; DISABLED WAIT PSW

to the RSCS operator's console. Using CP commands, the operator can display the virtual machine's PSW. The rightmost 3 hexadecimal characters indicate the error condition.

__WAIT STATE CODE X'001'__: If no RSCS message was issued, a program check interrupt occurred during the execution of the program check handler. A programming error is the probable cause.

If the RSCS message

DMTREX091T INITIALIZATION FAILURE — RSCS SHUTDOWN

was issued, RSCS operation was terminated because of an error in the loading of DMTAXS or DMTLAX. A dump of virtual storage is automatically taken. Verify that the CMS files 'DMTAXS TEXT' and 'DMTLAX TEXT' are correctly written and that they reside on the RSCS system residence device.

If the RSCS message

DMTREX090T PROGRAM CHECK IN SUPERVISOR — RSCS SHUTDOWN

was issued, the program check handler has terminated RSCS because of a program check interrupt in other than a dispatched line driver. A dump of virtual storage is automatically taken. A programming error is the probable cause.

The wait state code is loaded by DMTREX at RSCS termination or automatically during program check handling.

If neither of the last two messages was issued, use the CP DUMP command to dump the contents of virtual storage. Do an initial program load to restart the system. If the problem persists, notify your system support personnel.

__WAIT STATE CODE X'007'__: A program check interrupt has occurred during initial processing, before the program check handler could be activated. This may be caused by a programming error or by an attempt to load RSCS into an incompatible virtual machine. The latter case can occur if the virtual machine has (1) an incomplete instruction set, (2) less than 512K of virtual storage, or (3) does not have the required VM/370 DIAGNOSE interface support. The wait state code is loaded automatically during the initial loading and execution of the RSCS supervisor, DMTINI, DMTREX, DMTAXS or DMTLAX.

Verify that the RSCS virtual machine configuration has been correctly specified and that the "retrieve subsequent file descriptor" function of DIAGNOSE code X'14' is supported. Dump the contents of virtual storage via the CP DUMP command. If the problem persists, notify your system support personnel.

__WAIT STATE CODE X'011'__: An unrecoverable error occurred when reading the RSCS nucleus from DASD storage. This may be caused by a hardware malfunction of the DASD device. It may also be the result of an incorrect virtual DASD device definition, an attempt to use a system residence device unsupported by RSCS, incorrect RSCS system generation procedures, or the subsequent overlaying of the RSCS nucleus on the system residence device. The wait state code is loaded by DMTINI after an attempt, successful or not, to issue the message:

DMTINI402T IPL DEVICE READ I/O ERROR

Verify that the RSCS system residence device has been properly defined as a virtual DASD device and that the real DASD device is mounted and operable. If the problem persists, dump virtual storage via the CP DUMP command and notify your system support personnel. The RSCS system residence device may have to be restored or the RSCS system may have to be regenerated.

RSCS VIRTUAL MACHINE ENABLED WAIT

Whenever RSCS has no task ready for execution, DMTDSP loads a masked-on wait state PSW with a code of hexadecimal zeros. This occurs during normal RSCS operation and does not indicate an error condition. An external interrupt caused by command entry or an I/O interrupt due to the arrival of files automatically causes processing to resume.

RSCS Wait Problem Type	
Problem Type	Distinguishing Characteristics
Disabled RSCS wait	The RSCS operator is notified of the wait state because CP issues the message DMKDSP450W CP ENTERED; DISABLED WAIT PSW If, in addition, the message DMTINI402T IPL DEVICE READ I/O ERROR appears on the RSCS console, an unrecoverable error has occurred while reading the RSCS nucleus from DASD storage. RSCS enters a disabled wait state with a code of X'011' in the PSW. If a program check occurs before the program check handler is activated, RSCS enters a disabled wait state with a code of X'007' in the PSW. If a program check occurs after the program check handler is activated, RSCS enters a disabled wait state with a code of X'001' in the PSW. One of the following messages also appear on the RSCS console: DMTREX090T PROGRAM CHECK IN SUPERVISOR -- RSCS SHUTDOWN DMTREX091T INITIALIZATION FAILURE -- RSCS SHUTDOWN
Enabled RSCS wait	RSCS has no task ready for execution. A PSW, enabled for external and I/O interruptions, is loaded with a wait code of all zeros.

Figure 10. RSCS Wait Problem Type

SUMMARY OF VM/370 DEBUGGING TOOLS

Figure 11 summarizes the VM/370 commands that are useful in debugging. The CP and CMS commands are classified by the function they perform.

Function	Comments	CP Command	CMS Command
Stop execution at a specified location	Set the address stop before the program reaches a specified address. CMS allows 16 address stops to be active while CP allows only one.	ADSTOP {hexloc | OFF}	DEBUG BReak id {symbol | hexloc}
Resume execution	Resume execution where program was interrupted	Begin	DEBUG GO
	Continue execution at a specific location	Begin [hexloc]	DEBUG GO {symbol | hexloc}
Dump data	Dump the contents of specific storage locations.	DUMP {hexloc1 | Lhexloc1} [[{- | :} [hexloc2 | END]] | [{.} [bytecount | END]]] [*dumpid]	DEBUG DUmp [symbol1 | hexloc1 | 0] [symbol2 | hexloc2 | * | 32] [ident]

Figure 11. Summary of VM/370 Debugging Tools (Part 1 of 4)

Function	Comments	CP Command	CMS Command
Display data	Display contents of storage locations in hexadecimal)	Display hexloc1 [{- or :} [hexloc2 | END] or {.} [bytecount | END]]	DEBUG X { symbol [n | length] or hexloc [n | 4] }
	Display contents of storage locations (in hexadecimal and EBCDIC)	Display Thexloc1 [{- or :} [hexloc2 | END] or {.} [bytecount | END]]	
	Display storage key of specific storage lccations in hexadecimal	Display Khexloc1 [{- or :} [hexloc2 | END] or {.} [bytecount | END]]	
	Display general registers	Display Greg1 [{- or :} [reg2 | END] or {.} [regcount | END]]	DEBUG GPR reg1 [reg2]
	Display floating-point registers	Display Yreg1 [{- or :} [reg2 | END] or {.} [regcount | END]]	
	Display control registers	Display Xreg1 [{- or :} [reg2 | END] or {.} [regcount | END]]	
	Display contents of current virtual PSW in hexadecimal format	Display PSW	DEBUG PSW
	Display CAW contents	Display CAW	DEBUG CAW
	Display CSW contents	Display CSW	DEBUG CSW

Figure 11. Summary of VM/370 Debugging Tools (Part 2 of 4)

Function	Comments	CP Command	CMS Command
Store data	Store specified information into consecutive storage locations without alignment.	STore Shexloc hexdata...	DEBUG STore {symbol / hexloc} hexinfo
	Store specified words of information into consecutive fullword storage locations	STore {hexloc / Lhexloc} {hexword1[hexword2...]}	
	Store specified words of information into consecutive general registers	STore Greg hexword1 [hexword2...]	DEBUG SET GPR reg hexinfo[hexinfo]
	Store specified words of information into consecutive floating-point registers	STore Yreg hexword1 [hexword2...]	
	Store specified words of data into consecutive control registers	STore Xreg hexword1 [hexword2...]	
	Store information into PSW	STore PSW [hexword1] hexword2	DEBUG SET PSW hexinfo [hexinfo]
	Store information in CSW		DEBUG SET CSW hexinfo [hexinfo]
	Store information in CAW		DEBUG SET CAW hexinfo

Figure 11. Summary of VM/370 Debugging Tools (Part 3 of 4)

Function	Comments	CP Command	CMS Command
Trace execution	Trace all instructions, interrupts, and branches	TRace ALL	
	Trace SVC interrupts	TRACE SVC	SVCTrace ON
	Trace I/O interrupts	TRace I/O	
	Trace program interrupts	TRace PROgram	
	Trace external interrupts	TRace EXTernal	
	Trace privileged instructions	TRace PRIV	
	Trace all user I/O operations	TRace SIO	
	Trace virtual and real CCWs	TRace SIO TRace CCW	
	Trace all user interrupts and successful branches	TRace BRANCH	
	Trace all instructions	TRace INSTruct	
	End all tracing activity	TRace END	SVCTrace OFF
Trace real machine events	Trace events in real machine	MONitor STArt CPTRACE	
	Stop tracing events the real machine	MONitor STOP CPTRACE	

Figure 11. Summary of VM/370 Debugging Tools (Part 4 of 4)

COMPARISON OF CP AND CMS FACILITIES FOR DEBUGGING

If you are debugging problems while your virtual machine is running CMS, you can choose the CP or CMS debugging tools. See Figure 12 for a comparison of the CP and CMS debugging tools.

Function	CP	CMS
Setting address stops	Can set only one address stop at a time.	Can set up to 16 address stops at a time.
Dumping contents of storage to the printer	The dump is printed in hexadecimal format with EBCDIC translation. The storage address of the first byte of each line is identified at the left. The control blocks are formatted.	The dump is printed in hexadecimal format. The storage address of the first byte of each line is identified at the left. The contents of the general and floating-point registers are printed at the beginning of the dump.
Display the contents of storage and control registers at the terminal	The display occurs in hexadecimal format with EBCDIC translation. The CP command displays storage keys, floating-point registers and control registers.	The display occurs in hexadecimal format. The CMS commands do not display storage keys, floating-point registers or control registers as as the CP command does.
Storing information	The amount of information stored by the CP command is limited only by the length of the input line. The information can be fullword aligned when stored. CP stores data in the PSW, but not in the CAW or CSW. However, data can be stored in the CSW or CAW by specifying the hardware address in the STORE command. CP also stores the status of the virtual machine in the extended logout area.	The CMS command stores up to 12 bytes of information and can store data in the general registers but not in the floating-point or control registers. CMS stores data in the PSW, CAW, and CSW.
Tracing information	CP traces: • All interrupts, instructions and branches • SVC interrupts • I/O interrupts • Program interrupts • External interrupts • Privileged instructions • All user I/O operations • Virtual and real CCW's • All instructions The CP trace is interactive. You can stop and display other fields.	CMS traces all SVC interrupts. CMS displays the rupts. CMS displays the contents of general and floating-point registers before and after a routine is called. The parameter list is recorded before a routine is called.

Figure 12. Comparison of CP and CMS Facilities for Debugging

DEBUGGING CP ON A VIRTUAL MACHINE

Many CP problems can be isolated without standalone machine testing. It is possible to debug CP by running it in a virtual machine. In most instances, the virtual machine system is an exact replica of the system running on the real machine. To set up a CP system on a virtual machine, use the same procedure that is used to generate a CP system on a real machine. However, remember that the entire procedure of running service programs is now done on a virtual machine. Also, the virtual machine must be described in the real VM/370 directory. See the *VM/370: System Programmer's Guide* for directions for setting up the virtual machine.

ABEND DUMPS

There are three kinds of abnormal termination dumps possible when using CP. The first kind occurs when the problem program cannot continue. It terminates and in some cases attempts to issue a dump. The second occurs when the operating system for your virtual machine cannot continue. It terminates and in some cases attempts to issue a dump. In the VM/370 environment, both the problem program and the virtual machine's operating system dumps go to the virtual printer. A CLOSE must be issued to the virtual printer to have either dump print on the real printer.

A third kind of dump occurs when the CP system cannot continue. The CP abnormal termination dumps can be directed to a printer or tape or be dynamically allocated to DASD. If the dump is directed to a tape, the dumped data must fit on one reel of tape. Multiple tape volumes are not supported by VM/370. The historical data on the tape is in print line format and can be processed by user-created programs or via CMS commands.

Use the CP SET command to specify the output device for CP ABEND dumps. The format of the SET command is:

```
SET | DUmp { AUTO  } [ CP  ]
    |      { raddr } [ ALL ]
```

where:

DUMP — specifies the ABEND Dump.

AUTO — automatically directs the ABEND dump to disk.

raddr — directs the ABEND dump to the specified unit address (either a printer or a tape unit). If the address specifies a tape device, the dump data must fit on one reel; VM/370 does not support multiple tape volumes.

CP — dumps only the CP storage area.

ALL — dumps all of real storage.

USING THE VMFDUMP COMMAND

Use the CMS VMFDUMP command to print the dump on the real printer, when the CP ABEND dump is sent to a disk. The format of the VMFDUMP command is:

```
VMFDUMP | [ DUMPxx ]  [ ERASE  ]
        |             [ NOMAP  ]
        |             [ NOHEX  ]
        |             [ NOFORM ]
        |             [ NOVIRT ]
```

where:

DUMPxx — specifies the name of the CP dump file to be formatted and printed. xx may be any value from 00 to 09. Class D spool files contain only CP dump files. These files are searched for the indicated dump file. When the file is found, it is used to create a CMS file which, in turn, is formatted and printed.

ERASE — specifies that the CMS file which is being formatted and printed is to be erased at the conclusion of the program.

NOMAP — specifies that a load map is not to be printed.

NOHEX — specifies that a hexadecimal dump is not to be printed.

NOFORM — specifies that no formatted control blocks are to be printed.

NOVIRT — specifies that only the real machine control blocks are to be formatted. This option is ignored if NOFORM is also specified.

Use the VMFDUMP command to format and print a current or previous VM/370 system ABEND dump. Specify

VMFDUMP

to obtain a complete formatted, hexadecimal printout.

When the dump has been printed, one of two messages is printed:

DUMP FILE - DUMP xx - PRINTED AND KEPT

-- or --

DUMP FILE - DUMP xx - PRINTED AND ERASED

HOW TO PRINT A CP ABEND DUMP FROM TAPE

When the CP ABEND dump is sent to a tape, the records are 133 characters unblocked, and include carriage control characters.

To print the tape, first make sure the tape drive is attached to your system. Next, define the printer and tape file:

```
FILEDEF ddname1 PRINTER (RECFM F LRECL 133)

FILEDEF ddname2 {TAP2} (9-track DEN 1600
                {TAP1}
   RECFM F LRECL 133 BLOCK 133)
```

Then use the MOVEFILE command to print the tape:

```
MOVEFILE ddname2 ddname1
```

READING CP ABEND DUMPS

Two types of printed dumps occur when CP abnormally ends, depending on the options specified in the CP SET DUMP command. When the dump is directed to a direct access device, VMFDUMP must be used to format and print the dump. VMFDUMP formats and prints:

- Control blocks
- General registers
- Floating-point registers
- Control registers
- TOD (Time-of-Day) clock
- CPU timer
- Storage

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

If the CP SET DUMP command directed the dump to tape or the printer, the printed format of the dump is the same as with VMFDUMP, except that the control blocks are not formatted and printed.

When CP can no longer continue and abnormally terminates, you must first determine the condition that caused the ABEND, and then find the cause of that condition. You should know the structure and function of the Control Program. The following discussion on reading CP dumps includes many references to CP control blocks and control block fields. Refer to *VM/370: Data Areas and Control Block Logic* for a description of the CP control blocks. You will need the current load map for CP to be able to identify the modules from their locations. See "Load Map" later in this section for instructions for generating a load map.

REASON FOR THE ABEND

Determine the immediate reason for the ABEND. You need to examine several fields in the PSA (Prefix Storage Area) which is located in low storage, to find the reason for the ABEND.

- Examine the program old PSW and program interrupt code to find out if a program check occurred in CP. The program old PSW (PROPSW) is located at X'28' and the program interrupt code (INTPR) is at X'8E'. If a program check has occurred in supervisor mode, use the CP system load map to identify the module. If you cannot find the module using the load map, refer to "Identifying a Pageable Module."

- Examine the SVC old PSW, the SVC interrupt code, and the ABEND code to find out if a CP routine issued an SVC 0. The SVC old PSW (SVCOPSW) is located at X'20', the SVC interrupt code (INTSVC) is at X'8A', and the ABEND code (CPABEND) is at X'374'.

  The modules that may issue an SVC 0 are:

DMKBLD	DMKPSA
DMKCFG	DMKPTR
DMKCKS	DMKRGA
DMKCPI	DMKRNH
DMKCVT	DMKRPA
DMKDRD	DMKSCH
DMKDSP	DMKTDK
DMKFRE	DMKUDR
DMKHVD	DMKVDB
DMKIOS	DMKVDR
DMKNLD	DMKVIO
DMKPGS	DMKVMA
DMKPGT	DMKVSP
DMKPRG	

  The ABEND code (CPABEND) is a fullword in length. The first three bytes identify the module that issued the SVC 0 and the fourth byte is a binary field whose value indicates the reason for issuing an SVC 0. See "CP ABEND Codes, Reason and Action" in Section 3.

  Use the CP system load map to identify the module issuing the SVC 0. If you cannot find the module using the CP system load map, refer to "Identifying a Pageable Module" in this Section.

- Examine the old PSW at X'08'. If the operator has pressed the SYSTEM RESTART button on the CPU console, the old PSW indicates the instruction executing when the ABEND (caused by pressing the SYSTEM RESTART button) was recognized.

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

COLLECT INFORMATION

Examine several other fields in the PSA to analyze the status of the system. As you progress in reading the dump, you may return to the PSA to pick up pointers to specific areas (such as pointers to the real control blocks) or to examine other status fields.

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

- CP Running Status Field

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

CPSTAT Values	and Meaning
X'80'	CP is in wait state
X'40'	CP is running the user in RUNUSER
X'20'	CP is executing a stacked request

- Current User

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

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

The trace table is at least one page (4096 bytes) long. One page is allocated to the trace table for each block of 256K bytes of real storage available at IPL. Each trace table entry is 16 bytes long. The TRACSTRT field (location X'0C') contains the address of the start of the trace table. The TRACEND field (location X'10') contains the address of the byte following the end of the trace table. The address of the next available trace table entry is found in the TRACCURR field (location X'14').

Subtract 16 (X'10') bytes from the value at X'14' (TRACCURR) to find the address of the last trace table entry recorded.

REGISTER USAGE

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

The 16 general registers have many uses that vary depending upon the operation. The contents of some of the general registers follows:

Register	Contents
GR1	The virtual address to be translated
GR2	The real address or parameters
GR6,7,8	The address of the active VMBLOK and device control blocks
GR10	The address of the active IOBLOK
GR14,15	The external branch linkage

The following general registers always contain the same information:

Register	Contents
GR11	The address of the active VMBLOK
GR12	The base register for the module executing
GR13	The address of the current save area, if the module was called via an SVC

Use these registers, the CP control blocks, and the data in the prefix storage area to determine the error that caused the CP ABEND.

SAVE AREA CONVENTIONS

There are three save areas that may be helpful in debugging CP. If a module was called by an SVC, examine the SAVEAREA. SAVEAREA is not in the PSA; the address of the SAVEAREA is found in general register 13. If a module was called by a BALR, the general registers are saved in the PSA in an area called BALRSAVE (X'240'). The DMKFRE save area and work area is also in the PSA: these areas are only used by the DMKFREE and DMKFRET routines. The DMKFRE save area (FREESAVE) is at location X'280' and its work area (FREEWORK) follows at location X'2C0'.

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

- SAVEAREA

  An active save area contains the caller's return address in SAVERETN (displacement X'00'). The caller's base register is saved in SAVER12 (displacement X'04'), and the address of the save area for the caller is saved in SAVER13 (displacement X'08'). Using SAVER13, you can trace back again.

- BALRSAVE

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

- FREESAVE

  All the general registers are saved in FREESAVE before DMKFRE executes. Use the address of FREESAVE to trace module control backwards.

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

VIRTUAL AND REAL CONTROL BLOCK STATUS

Examine the virtual and real control blocks for more information on the status of the CP system.

VMBLOK

The address of the VMBLOK is in general register 11. Examine the following VMBLOK fields:

- The virtual machine running status is contained in VMRSTAT (displacement X'58'). The value of this field indicates the running status:

VMRSTAT Status	Meaning
X'80'	Waiting, executing console function
X'40'	Waiting, page operation
X'20'	Waiting, scheduled IOBLOK start
X'10'	Waiting, virtual PSW wait state
X'08'	Waiting, instruction simulation
X'04'	User not yet logged on
X'02'	User logging off
X'01'	Virtual machine in idle wait state

- The virtual machine dispatching status is contained in VMDSTAT (displacement X'59'). The value of this field indicates the dispatching status:

VMDSTAT Values	Meaning
X'80'	Virtual machine is dispatched RUNUSER
X'40'	Virtual machine is compute bound
X'20'	Virtual machine in-queue time slice end
X'10'	Virtual machine in TIO/SIO busy loop
X'08'	Virtual machine is runable
X'04'	Virtual machine in a queue

- Examine the virtual PSW and the last virtual machine privileged instruction. The virtual machine PSW is saved in VMPSW (displacement X'A8') and the virtual machine privileged or tracing instruction is saved in VMINST (displacement X'98').

- Find the name of the last CP command that executed in VMCOMND (displacement X'148').

- Check the status of I/O activity. The following fields contain pertinent information.

  -- VMPEND (displacement X'63') contains the interrupt pending summary flag. The value of VMPEND identifies the type of interrupt.

VMPEND Values	Meaning
X'40'	Virtual PER (Program Event Recording) interrupt pending
X'20'	Virtual program interrupt deferred
X'10'	Virtual SVC interrupt deferred
X'08'	Virtual pseudo page fault pending
X'02'	Virtual I/O interrupt pending
X'01'	Virtual external interrupt pending

  -- VMIOINT (displacement X'6A') contains the I/O interrupt pending flag. Each bit represents a channel (0-15). An interrupt pending is indicated by a 1 in the corresponding bit position.

VMIOINT Values	Meaning
10000000	00000000 Interrupt pending on channel 0
01000000	00000000 Interrupt pending on channel 1
.	.
.	.
.	.
00000000	00000001 Interrupt pending on channel 15

  -- VMIOACTV (displacement X'36') is the active channel mask. An active channel is indicated by a 1 in the corresponding bit position.

VCHBLOK

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

- The virtual channel address is contained in VCHADD (displacement X'00').

- The status of the virtual channel is found in the VCHSTAT field (displacement X'06'). The value of this field indicates the virtual channel status:

VCHSTAT

Values	Meaning
X'80'	Virtual channel busy
X'40'	Virtual channel class interrupt pending
X'01'	Virtual channel dedicated

- The value of the VCHTYPE field (displacement X'07') indicates the virtual channel type:

VCHTYPE

Values	Meaning
X'80'	Virtual selector channel
X'40'	Virtual block multiplexer

VCUBLOK

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

- The virtual control unit address is found in the VCUADD field (displacement X'00').

- The value of the VCUSTAT field (displacement X'06') indicates the status of the virtual control unit:

VCUSTAT

Values	Meaning
X'80'	Virtual subchannel busy
X'40'	Interrupt pending in subchannel
X'20'	Virtual control unit busy
X'10'	Virtual control unit interrupt pending
X'08'	Virtual control unit end pending

- The value of the VCUTYPE field (displacement X'07') indicates the type of the virtual control unit:

VCUTYPE

Values	Meaning
X'80'	Virtual control unit on shared subchannel
X'40'	Virtual control unit is a channel-to-channel adapter

VDEVBLOK

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

- The virtual device address is found in the VDEVADD field (displacement X'00').

- The value of the VDEVSTAT field (displacement X'06') describes the status of the virtual device:

VDEVSTAT

Values	Meaning
X'80'	Virtual subchannel busy
X'40'	Virtual channel interrupt pending
X'20'	Virtual device busy
X'10'	Virtual device interrupt pending
X'08'	Virtual control unit end
X'04'	Virtual device not ready
X'02'	Virtual device attached by console function
X'01'	VDEVREAL is dedicated to device RDEVBLOK

- The value of the VDEVFLAG field (displacement X'07') indicates the following device dependent information:

VDEVFLAG

Values	Meaning
X'80'	DASD--read-only device
X'80'	Virtual 2701/2702/2703 device--line enabled
X'40'	DASD--TDISK space allocated by CP
X'40'	Virtual 2701/2702/2703 device--line connected
X'40'	Console--activity spooled
X'20'	DASD--2311 device simulated on top half of 2314
X'10'	DASD--2311 device simulated on bottom half of 2314
X'10'	Console and spooling device--processing first CCW
X'08'	DASD--executing standalone seek
X'02'	RESERVE/RELEASE are valid CCW operation codes.
X'01'	Virtual device sense bytes present

- The VDEVCSW field (displacement X'08') contains the virtual channel status word for the last interrupt.

- The VDEVREAL field (displacement X'24') contains the pointer to the real device block, RDEVBLOK.

- The VDEVIOB field (displacement X'34') contains the pointer to the active IOBLOK.

- For console devices, the value of the VDEVCFLG field (displacement X'26') describes the virtual console flags:

VDEVCFLG

Values	Meaning
X'80'	User signaled attention too many times
X'40'	Last CCW processed was a TIC
X'20'	Data transfer occurred during this channel program
X'10'	Virtual console function in progress
X'08'	Automatic carriage return on first read

- For spooling devices, the value of the VDEVSFLG field (displacement X'27') describes the virtual spooling flags:

VDEVSFLG

Values	Meaning
X'80'	Spool reader--last command was a feed
X'80'	Spool output--transfered to VSPXXUSR
X'40'	Spool device--continuous operation
X'20'	Hold output--save input
X'10'	Spool output--for user and distribution
X'08'	Spool input -- set unit exception at EOF
X'08'	Terminal output required for spooled console
X'04'	Device closed by console function
X'02'	Spool output--purge file at close
X'02'	Spool input--device opened by DIAGNOSE
X'01'	Spool output--DMKVSP entered via SVC

- For output spooling devices, the VDEVEXTN field (displacement X'10') contains the pointer to the virtual spool extension block, VSPXBLOK.

RCHBLOK

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

- The real channel address is found in the RCHADD field (displacement X'00').

- The value of the RCHSTAT field (displacement X'04') describes the status of the real channel.

RCHSTAT

Values	Meaning
X'80'	Channel busy
X'40'	IOB scheduled on channel
X'20'	Channel disabled
X'01'	Channel dedicated

- The value of the RCHTYPE field (displacement X'05') describes the real channel type:

RCHTYPE

Values	Meaning
X'80'	Selector channel
X'40'	Block multiplexer channel
X'20'	Byte multiplexer channel
X'01'	System/370 type channel (System/370 instruction support)

- The RCHFIOB field (displacement X'08') is the pointer to the first IOBLOK in the queue and the RCHLIOB field (displacement X'0C') is the pointer to the last IOBLOK in the queue.

RCUBLOK

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

- The RCUADD field (displacement X'00') contains the real control unit address.

- The value of the RCUSTAT field (displacement X'04') describes the status of the control unit:

RCUSTAT

Values	Meaning
X'80'	Control unit busy
X'40'	IOB scheduled on control unit
X'20'	Control unit disabled
X'01'	Control unit dedicated

- The value of the RCUTYPE field (displacement X'05') describes the type of the real control unit:

RCUTYPE

Values	Meaning
X'80'	This control unit can attach to only one subchannel
X'01'	TCU is a 2701
X'02'	TCU is a 2702
X'03'	TCU is a 2703
X'04'	Subordinate control unit

- The RCUFIOB field (displacement X'08') points to the first IOBLOK in the queue and the RCULIOB field (displacement X'0C') points to the last IOBLOK in the queue.

RDEVBLOK

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

- The RDEVADD field (displacement X'00') contains the real device address.

- The values of the RDEVSTAT (displacement X'04') and RDEVSTA2 (displacement X'43') fields describe the status of the real device:

RDEVSTAT

Values	Meaning
X'80'	Device busy
X'40'	IOB scheduled on device
X'20'	Device disabled (offline)
X'10'	Device reserved
X'08'	Device in intensive error recording mode
X'04'	Device intervention required
X'02'	GRAF - IOBLOK pending; queue requests
X'01'	Dedicated device (attached to a user)

RDEVSTA2

Values	Meaning
X'80'	Active device is being reset
X'40'	Device is busy with the channel
X'20'	Contingent connection present

- The value of the RDEVFLAG field (displacement X'05') indicates device flags. The following flags are device dependent.

RDEVFLAG

Values	Meaning
X'80'	DASD--ascending order seek queueing
X'40'	DASD--volume preferred for paging
X'20'	DASD--volume attached to system
X'10'	DASD--CP owned volume
X'08'	DASD--volume mounted but not attached
X'80'	Console--terminal has print suppress feature
X'40'	Console--terminal executing prepare command
X'20'	Console--IOBLOK pending; queue request
X'10'	Console--2741 terminal code identified
X'08'	Console--device is enabled
X'04'	Console--next interrupt from a halt I/O
X'02'	Console--device is to be disabled
X'01'	Console--3704/3705 NCP resource in EP mode
X'80'	Spooling--device output drained
X'40'	Spooling--device output terminated
X'20'	Spooling--device busy with accounting
X'10'	Spooling--force printer to single space
X'08'	Spooling--restart current file
X'04'	Spooling--backspace the current file
X'02'	Spooling--print/punch job separator
X'01'	Spooling--UCS buffer verified
X'80'	Special--network control program is active
X'40'	Special--2701/2702/2703 emulation program is active
X'20'	Special--3704/3705 is in buffer slowdown mode
X'10'	Special--automatic dump/load is enabled
X'08'	Special--IOBLOK is pending; queue requests
X'04'	Special--emulator lines are in use by system
X'02'	Special--automatic dump/load process is active
X'01'	Special--basic terminal unit trace requested

- The value of the RDEVTYPC field (displacement X'06') describes the device type class and the value of the RDEVTYPE field (displacement X'07') describes the device type.
- The RDEVAIOB field (displacement X'24') contains the address of the active IOBLOK.
- The RDEVUSER field (displacement X'28') points to the VMBLOK for a dedicated user.
- The RDEVATT field (displacement X'2C') contains the attached virtual address.
- The RDEVIOER field (displacement X'48') contains the address of the IOERBLOK for the last CP error.
- For spooling unit record devices, the RDEVSPL field (displacement X'18') points to the active RSPLCTL block.
- For real 3704/3705 Communications Controllers, several pointer fields are defined. The RDEVEPDV field (displacement X'1C') points to the start of the free RDEVBLOK list for EP lines. The RDEVNICL field (displacement X'38') points to the network control list and the RDEVCKPT field (displacement X'3C') points to the CKPBLOK. Also, the RDEVMAX field (displacement X'2E') is the highest valid NCP resource name and the RDEVNCP field (displacement X'30') is the reference name of the active 3705 NCP.
- For terminal devices, additional flags are defined. The value of the RDEVTFLG field (displacement X'3A') describes the additional flags:

RDEVTFLG

Values	Meaning
X'80'	Terminal--logon process has been initiated
X'40'	Terminal--terminal in reset process
X'20'	Terminal--suppress attention signal
X'80'	Graphic--logon process initiated
X'40'	Graphic--screen full, more data waiting
X'20'	Graphic--screen in running status
X'10'	Graphic--read pending for screen input
X'08'	Graphic--last input not accepted
X'04'	Graphic--timer request pending
X'02'	Graphic--control function interruption pending
X'01'	Screen full, hold status

- For terminals, an additional flag is defined. The value of the RDEVTMCD field (displacement X'46') describes the line code translation to be used:

RDEVTMCD

Values	Meaning
X'10'	UASCII--8 level
X'0C'	APL correspondence
X'08'	APL PTTC/EBCD
X'04'	Correspondence
X'00'	PTTC/EBCD

IDENTIFYING A PAGEABLE MODULE

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

Go to the address indicated in the PSW. Backtrack to the beginning of that page frame. The first eight bytes of that page frame (the page frame containing the address pointed to by the PSW) contain the name of the failing module. If multiple modules exist within the same page frame, identify the module using the load map and failing address displacement within the page frame.

READING CMS ABEND DUMPS

When CMS abnormally terminates, the terminal operator must issue the DEBUG command and then the DUMP subcommand if an ABEND dump is desired. The DUMP formats and prints the following:

- General registers
- Extended control registers
- Floating-point registers
- Storage boundaries with their corresponding storage protect key
- Current PSW
- Selected storage

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

When CMS can no longer continue, it abnormally terminates. You must first determine the condition that caused the ABEND and then find why the condition occurred. In order to find the cause of a CMS problem, you must be familiar with the structure and functions of CMS. The discussion about reading CMS dumps refers to several CMS control blocks and fields in the control blocks. Refer to the _VM/370: Data Areas and Control Block Logic_ for a description of each CMS control block. Figure 13 shows the relationships of CMS control blocks. You also need a current CMS nucleus load map to analyze the dump.

REASON FOR THE ABEND

Determine the immediate reason for the ABEND and identify the failing module. The ABEND message DMSABN148T contains an ABEND code and failing address. "CMS ABEND Codes" in Section 3 lists all the CMS ABEND codes, identifies the module that caused the module to abnormally terminate, and describes the action that should be taken whenever CMS abnormally terminates.

You may have to examine several fields in the nucleus constant area (NUCON) of low storage.

1. Examine the program old PSW (PGMOPSW) at location X'28'. Using the PSW and current CMS load map, determine the failing address.

2. Examine the SVC old PSW (SVCOPSW) at location X'20'.

3. Examine the external old PSW (EXTOPSW) at location X'18'. If the virtual machine operator terminated CMS, this PSW points to the instruction executing when the termination request was recognized.

4. For a machine check, examine the machine check old PSW (MCKOPSW) at location X'30'.

COLLECT INFORMATION

Examine several other fields in NUCON to analyze the status of the CMS system. As you proceed with the dump, you may return to NUCON to find pointers to specific areas (such as pointers to file tables) or to examine other status fields. The complete contents of NUCON and the other CMS control blocks are described in the _VM/370: Data Areas and Control Block Logic_. The following areas of NUCON may contain useful debugging information.

NUCON AREAS

- Save Area For Low Storage.

  Before executing, DEBUG saves the first 160 bytes of low storage in a NUCON field called LOWSAVE. LOWSAVE begins at X'C0'.

- Register Save Area.

  DMSABN, the ABEND routine, saves the user's floating-point and general registers.

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

- Device.

  The name of the device causing the last I/O interrupt is in the DEVICE field at X'26C'.

- Last Two Commands or Procedures Executed.

Field	Location	Contents
LASTCMND	X'2A0'	Last CMS command issued
PREVCMND	X'2A8'	Next to last CMS command issued
LASTEXEC	X'2B0'	Last EXEC procedure invoked
PREVEXEC	X'2B8'	Next to last EXEC procedure invoked

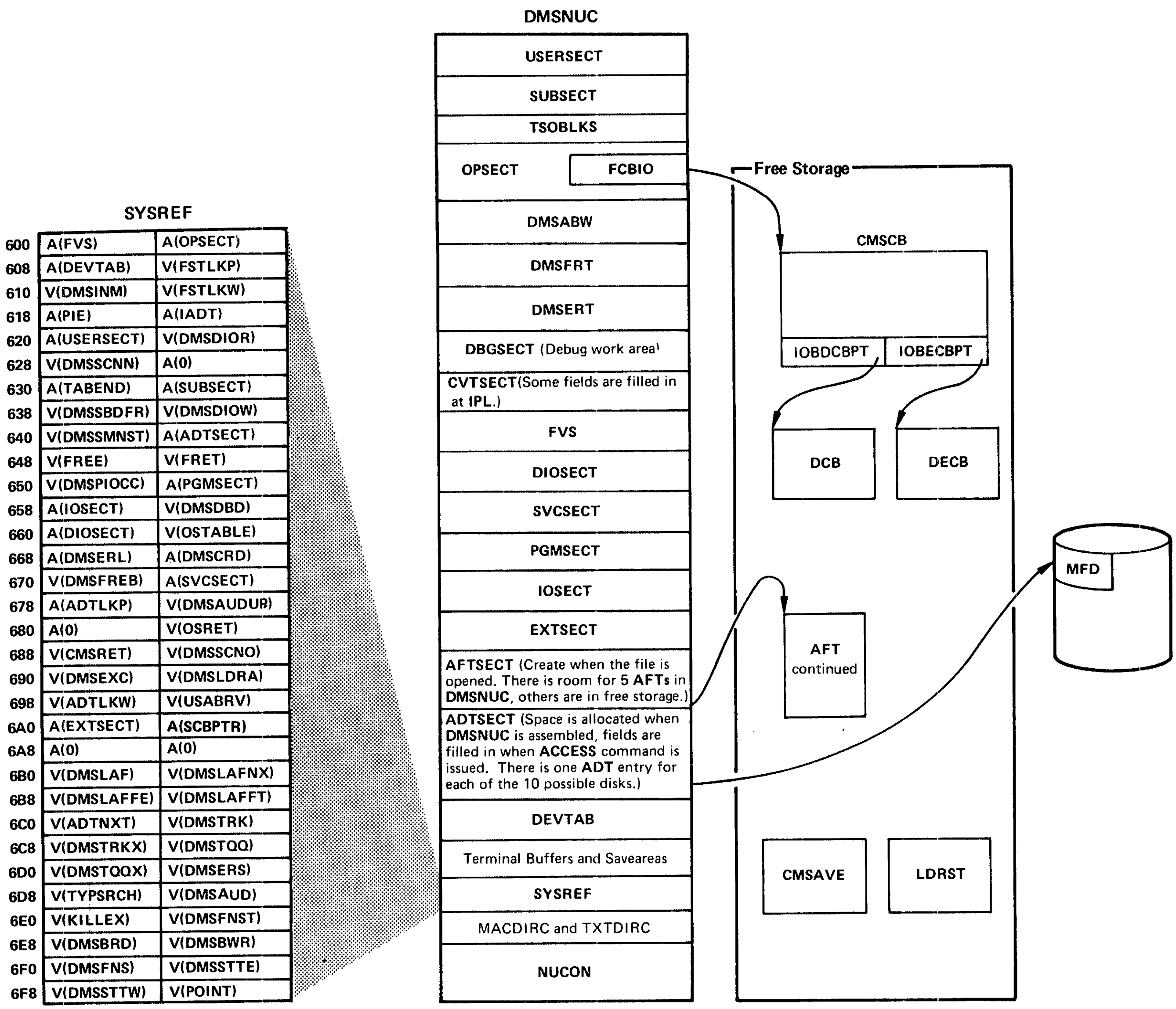


Figure 13. CMS Control Blocks

- Last Module Loaded Into Free Storage and Transient Area.

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

- Pointer to CMSCB.

  The pointer to the CMSCB is in the FCBTAB field located at X'5C0'. CMSCB contains the simulated OS control blocks. These simulated OS control blocks are in free storage. The CMSCB contains a PLIST for CMS I/O functions, a simulated job file control block (JFCB), a simulated data event block (DEB), and the first in a chain of I/O blocks (IOBs).

- The Last Command.

  The last command entered from the terminal is stored in an area called CMNDLINE (X'7A0'), and its corresponding PLIST is stored at CMNDLIST (X'848').

- External Interrupt Work Area.

  EXTSECT (X'1550') is a work area for the external interrupt handler. It contains:

-- The PSW, EXTPSW (X'15F8')
-- Register save areas, EXSAVE1 (X'15B8')
-- Separate area for timer interrupts, EXSAVE (X'1550')

- I/O Interrupt Work Area.

  IOSECT (X'1620') is a work area for the I/O interrupt handler. The oldest and newest PSW and CSW are saved. Also, there is a register save area.

- Program Check Interrupt Work Area.

  PGMSECT (X'16B0') is a work area for the program check interrupt handler. The old PSW and the address of register 13 save area are stored in PGMSECT.

- SVC Work Area.

  SVCSECT (X'1748') is a work area for the SVC interrupt handler. It also contains the first four register save areas assigned. The SFLAG (X'1758') indicates the mode of the called routine. The values have the following meanings:

Flag	Description
X'80'	SVC protect key is zero
X'40'	Transient area routine
X'20'	Nucleus routine
X'01'	Invalid re-entry flag

  Also, the SVC ABEND code, SVCAB, is located at X'175A'.

- Simulated CVT (Communications Vector Table).

  The CVT, as supported by CMS, is CVTSECT (X'1CC8'). Only the fields supported by CMS are filled in.

- Active Device Table and Active File Table.

  For file system problems, examine the ADT (Active Device Table), or AFT (Active File Table) in NUCON.

REGISTER USAGE

To trace control blocks and modules, it is important to know the CMS register usage conventions.

Register	Contents
GR1	Address of the PLIST
GR12	Program's entry point
GR13	Address of a 12-doubleword work area for an SVC call
GR14	Return address
GR15	Program entry point or the return code

The preceding information should help you to read a CMS dump. With a dump, the control block diagrams, and a CMS load map you should be able to find the cause of the ABEND.

NUCLEUS LOAD MAP

Each time the CMS resident nucleus is loaded on a DASD, and an IPL can be performed on that DASD, a load map is produced. Save this load map. It lists the virtual storage locations of nucleus-resident routines and work areas. Transient modules are not included in this load map. When debugging CMS, you can locate routines using this map.

The load map may be saved as a disk file and printed at any time. A copy of the nucleus load map is contained on the system with the file identification of 'filename NUCMAP'. Issue the

    LISTF * NUCMAPS

command to determine the filename. Then issue

    PRINT filename NUCMAP

to obtain a copy of the current nucleus load map.

Figure 14 shows a sample CMS load map. Notice that the debug work area (DBGSECT) and the DMSINM module have been located.

LOAD MAP

The load map of a disk-resident command module contains the location of control sections and entry points loaded into storage. It may also contain certain messages and card images of any invalid cards or it may replace cards that exist in the loaded files. The load map is contained in the third record of the MODULE file.

This load map is useful in debugging. When using the debug environment to analyze a program, use the program's load map to help in displaying information.

There are two ways to get a load map:

1. When loading relocatable object code into storage, make sure that the MAP option is in effect when the LOAD command is issued. Because MAP is the default option, be sure that NOMAP is not specified. A load map is then created on the primary disk each time a LOAD command is issued.

2. When generating the absolute image form of files already loaded into storage, make sure that the MAP option is in effect when the GENMOD command is issued. Because MAP is the default option, be sure that NOMAP is not specified. Issue the MODMAP command to type the load map associated with the specified MODULE file on the terminal. The format of the MODMAP command is:

MODmap	filename

<u>where</u>:

filename is the module whose map is to be displayed. The filetype must be MODULE.

```
INVALID CARD...:READ  DMSNUC    TEXT      C1 CMS191   9/21/72    9:01
           *         UPLIB     MACLIB    D1 CMS191   9/21/72   8:47
           *         CMSLIB    MACLIB    D1 CMS191   9/21/72   8:44
           *         OSMACRO   MACLIB    Y2 CMS19E   7/19/72  18:11
           *         DMSNUC    ASSEMBLE  C1 SOURCE   9/18/72  23:09
DMSNUC   AT 000000
DMSNUCU  AT 002800
NUCON    AT 000000
SYSREF   AT 000600
FEIBM    AT 000274
CMNDLINE AT 0007A0
SUBFLAG  AT 0005E9
IADT     AT 000644
DEVICE   AT 00026C
DEVTAB   AT 000C90
CONSOLE  AT 000C90
ADISK    AT 000CA0
DDISK    AT 000CD0
SDISK    AT 000D10
YDISK    AT 000D20
TABEND   AT 000DF0
ADTSECT  AT 000DF0
AFTSTART AT 001200
EXTSECT  AT 001500
EXTPSW   AT 0015A8
IOSECT   AT 0015D0
IONTABL  AT 001610
PGMSECT  AT 001660
PIE      AT 001668
SVCSECT  AT 0016F8
DIOSECT  AT 001998
FVS      AT 001A88
ADTFVS   AT 001B48
KXFLAG   AT 001C2F
UFDBUSY  AT 001C2E
CMSCVT   AT 001C80
DBGSECT  AT 001D80
DMSERT   AT 002098
DMSFRT   AT 002208
DMSABW   AT 002258
OPSECT   AT 002800
DMSERL   AT 002935
TSOBLKS  AT 0029B0
SUBSECT  AT 002A40
USERSECT AT 002AD8
INVALID CARD...:READ  DMSINA    TEXT      C1 CMS191   9/19/72   15:37
ABBREV   AT 003000
USABRV   AT 0030D0
INVALID CARD...:READ  DMSINM    TEXT      C1 CMS191   9/18/72   20:36
CMSTIMER AT 003200
GETCLK   AT 003200
DMSINM   AT 003200
INVALID CARD...:READ  DMSTIO    TEXT      C1 CMS191   9/19/72   10:33
TAPEIO   AT 003308
DMSTIO   AT 003308
```

Figure 14. Sample CMS Load Map

CP INTRODUCTION

The VM/370 Control Program (CP) manages the resources of a System/370 to provide virtual storage support by using virtual machines. With this support each terminal user appears to have the complete function of a dedicated System/370 at his disposal, even though many other users may be running batch, teleprocessing, time-sharing, testing, or production jobs at the same time.

A user defines the configuration he requires -- input/output (I/O) device addresses, and a storage size up to 16 million bytes -- regardless of whether they match the real machine's configuration. Virtual devices must have real counterparts, but not always in a one-for-one ratio. For example, many users' readers, punches, and printers can be mapped onto common spool disks, and their virtual disk devices may be mapped as minidisks onto different sections of common disk packs, effectively multiplying the number of logical disk devices that are available on the real machine.

Each user's virtual machine comprises:

- An operator's console (his local or remote terminal device)
- A virtual CPU either with or without virtual storage addressing.
- Virtual storage of up to 16 million bytes
- Virtual I/O devices

Note: If an operating system that manages virtual storage is running in the virtual machine, the CPU must have extended control (EC) mode.

Virtual I/O devices are controlled by the virtual machine's operating system, not by CP. Thus, for proper operation, the support for the correct number and type of I/O devices must be provided by the operating system of the virtual machine. CP monitors, translates, and schedules all real I/C operations to provide system integrity. It executes all virtual machine operations in a problem state by trapping, screening, and processing all the interrupts, and passing on the necessary information to the appropriate virtual machine. Only CP executes in the privileged state.

To increase the amount of real main storage available to the user's virtual machine, parts of CP that are infrequently used are not resident in main storage. Instead, they reside on part of the auxiliary paging storage used by the system, and are brought into main storage only when they are required.

Because CP nonresident modules are paged into main storage, CP also occupies virtual storage space. The system VMBLOK, assembled into the resident control program in the module DMKSYS, defines this space. The VMBLOK has a pointer to a segment table that references a set of page and swap tables that describe CP's virtual storage space.

The virtual space is divided into 2 parts; the first part (4 segments (256K)) is reserved for executable CP code, both resident and pageable; the second part (the remaining storage of at least 256K) is dynamically allocated for spooling buffers and for user directory functions. For a routine to be pageable, a number of restrictions must be observed.

When the system is loaded, resolved, and written onto the system residence volume, pagable modules must be loaded at addresses higher in main storage than the symbol DMKCPEND, which defines the last byte of the resident CP nucleus. This is done by reordering the LOAD-LIST EXEC that the VMFLOAD procedure uses when punching out the text decks that comprise the Any pageable modules are listed after the entry for DMKCPE. In addition, the set page boundary (SPB) loader control card must precede each pageable module. This SPB card forces the loader to start loading the succeeding module at the next higher 4k page boundary and ensures that the entire module is resident when it is paged in.

If several pageable modules perform similar or related functions and if they are likely to be resident at the same time, they may be included in the same page by omitting the SPB cards that would normally have preceded the second and subsequent modules. The group of modules to be loaded together must not exceed 4K as their total storage requirement; if they do, one or more must be loaded in separate pages, because no page boundary crossover in the pageable part of the control program is allowed. All currently pageable CP modules punch their own SPB card via an assembler PUNCH statement, except those that are designed to reside in a page along with other modules.

CP INITIALIZATION

System initialization (IPL) prepares VM/370 for operation. IPL performs the following tasks:

- Initializes main storage
- Mounts devices
- Reads spool file checkpoint records, on a warm start from the warm start cylinder; reads spool file checkpoint records on a checkpoint or force start, from the checkpoint cylinders.
- Allocates space for the system dump file
- Logs on the system operator

In the case of a system restart that follows a failure, active files and the system log message are written on the warm start cylinder before the CP nucleus can be brought into main storage. The user can now log on.

VIRTUAL MACHINE MANAGEMENT

A virtual machine is created for a user when he logs on VM/370, on the basis of information stored in his directory entry. The entry for each user identification includes a list of the virtual I/O devices associated with his virtual machine and the real device equivalents.

The directory file contains additional information about the virtual machine. Included are the VM/370 command privilege classes for the user, accounting data, normal and maximum virtual storage sizes, and optional virtual machine characteristics such as extended control mode.

CP supervises virtual machine execution by (1) permitting only problem state execution except in its own routines, and (2) receiving control after all interruptions occur on the real system. CP intercepts each privileged instruction and simulates it if the current PSW of the issuing virtual machine indicates a virtual supervisor state. If the virtual machine is running in the problem state, an attempt to execute a privileged instruction is reflected back to the virtual machine as a program interruption. All virtual machine interruptions (including those caused by attempting privileged instructions) are first handled by CP, and are reflected to the virtual machine if an equivalent interruption would have occured on the real machine.

Virtual Machine Time Management

The real CPU uses time-slicing to simulate multiple virtual CPUs. Virtual machines executing in a conversational mode are given access to the real CPU 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 interruptions. The virtual machine is queued for subsequent CPU usage according to whether it is a conversational or nonconversational user of system resources.

A virtual machine can gain control of the CPU 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 CPU if it is waiting for a page of storage, an I/O operation to be translated and started, or a CP command to finish execution.

A virtual machine can be assigned a priority of execution. Priority is a parameter affecting the execution privilege of a particular virtual machine in comparison to other virtual machines that have the same general execution characteristics. Priority may be assigned by the real machine operator, but is more frequently determined by the virtual machine's directory entry.

Virtual Machine Storage Management

The normal and maximum storage sizes of a virtual machine are defined in the virtual machine configuration in the VM/370 directory. The virtual storage size can be temporarily redefined to any value that is a multiple of 4K and not greater than the value stated as the maximum allowable in the directory. VM/370 uses this storage as virtual storage. The storage can appear as paged or nonpaged to the virtual machine, depending upon whether the extended control (EC) mode option was specified for that virtual machine. EC mode is required if operating systems that control virtual storage, such as OS/VS1 or VM/370, are to be 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 describe the storage of each virtual machine. These tables are maintained by CP and reflect the allocation of virtual storage pages to blocks of real storage. The System/370 machine uses these tables to address virtual storage. Storage in the real machine is logically and physically divided into 4096 byte areas called page frames or blocks.

Only referenced virtual storage pages are kept in real storage and, therefore, use real storage more efficiently. A page can be brought into any available page frame; the necessary relocation is done during program execution by a combination of VM/370 software and the dynamic address translation hardware of 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 starts suitable operations to write them out on a paging device (paging out).

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

Paging is done on demand by CP. This means that a page of virtual storage is not read from the paging device and written to a real storage block until it is needed for virtual machine execution. No attempt is made by CP to anticipate what pages might be required by a virtual machine. While a paging operation is being performed for one virtual machine, another virtual machine can be executing. Any paging operation started by CP is transparent to the virtual machine.

If the virtual machine is executing in EC mode with translate on, two additional sets of segment and page tables are maintained. The virtual machine operating system is responsible for the equivalency of the virtual storage created by it to the virtual 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.

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

During the process of CP system generation, the installation may specify that a single virtual machine is to be given an option called virtual=real. With this option, the virtual machine's storage is allocated directly from real storage at the time CP is initially loaded, and remains allocated until released by an operator command. All pages except page zero are allocated to the corresponding real storage locations. 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 because it removes the need for CP to perform paging operations for the selected virtual machine. The virtual=real option is necessary whenever programs that contain dynamically modified channel programs (excepting those of OS ISAM) are to execute under control of CP.

Virtual Machine I/O Management

A real disk device can be shared among multiple virtual machines. Virtual device sharing is specified in the directory entry or by a user command. If sharing is requested by a user command, an appropriate password must be supplied before gaining access to the virtual device. A particular virtual machine can be assigned read-only or read/write access to a shared disk device. CP verifies each virtual machine I/O operation against the parameters in the virtual machine configuration to ensure device integrity.

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 directory entry of the virtual machine, or they may be attached to (or detached from) the virtual machine's configuration while it remains logged on. 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 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 are both virtual. It is the responsibility of CP to translate the virtual specifications to real.

In addition, the interruptions caused by the I/O operation 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. These are the responsibility of the virtual machine operating system.

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

SPOOLING

A virtual unit record device, which is mapped directly to a real unit record device, is 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. Because virtual machines controlled by CMS ordinarily have modest requirements for unit record I/O, such device sharing is quite advantageous, and it is the standard mode of system operation.

Spooling operations stop if the direct access storage space assigned to spooling is exhausted, and the virtual unit record devices are in a not ready status. The system operator can 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, backspacing any number of printer pages, and defining spool classes for scheduling real output.

Remote Spooling

The Remote Spooling Communications Subsystem (RSCS), a component of VM/370, provides support for the automatic transfer of spool files generated by VM/370 virtual machines to remote locations. It also supports the transmission of files from remote locations to virtual users.

RSCS uses the CP spooling facilities of VM/370 to:

- Gain access to files spooled to RSCS by VM/370 users for transmission to remote locations.
- Transfer files, received from remote locations, to the intended VM/370 virtual machines.

This support is fully described in the IBM VM/370: RSCS User's Guide.

CONSOLE FUNCTIONS

The CP console functions allow the user to control the virtual machine from the terminal, much as an operator controls a real machine. Virtual machine execution can be stopped at any time by the terminal's attention key; it can be restarted by typing in the appropriate CP command. External, attention, and device ready interruptions can be simulated on the virtual machine. Virtual storage, virtual machine registers, the PSW and CSW can be inspected and modified. Extensive trace facilities are provided for the virtual machine, as well as a single-instruction mode. Commands control the spooling and disk sharing functions of CP.

Console functions are divided into privilege classes. The directory entry for each user assigns one or more privilege classes. The classes are:

- Primary system operator
- System resource operator
- System programmer
- Spooling operator
- Systems analyst
- IBM field engineer or PSR
- General user

Commands in the system analyst class can inspect real storage locations, but they cannot make modifications to real storage. Commands in the operator class control real resources. System operator commands include all those relating to virtual machine performance options, such as assigning a set of reserved page frames to a selected virtual machine.

PROGRAM STATES

When instructions in CP are being executed, the real computer is in the supervisor state; at all other times, when running virtual machines, it is in the problem state. Therefore, privileged instructions can only be executed by CP. Programs running on a virtual computer can issue privileged instructions; such an instruction causes an interruption that is handled by CP. CP examines the operating status of the virtual machine PSW. If the virtual machine indicates that it is functioning in supervisor mode, then the privileged instruction is simulated according to its type. If the virtual machine is in problem mode, then the privileged interrupt is reflected to the virtual machine.

Only CP 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 interruptions, including those caused by attempted privileged operations, are handled by CP, which then reflects to the user program only those interruptions that the user program would expect from a real machine. A problem program executes on the virtual machine in a manner identical to its execution on a real System/370 CPU, as long as it does not violate the CP restrictions.

PREFERRED VIRTUAL MACHINE

CP supports four special virtual machine operating environment functions. Each function can be applied to one virtual machine at a time. Although each function could be applied to a different virtual machine, optimum performance would not be achieved. Each function is discussed separately following.

Favored Execution

CP attempts to provide up to a specified percentage of CPU time to a particular virtual machine, provided that the virtual machine is functioning in a way that fully utilizes CPU time. At regular time intervals the CP dispatcher checks the CPU time used by the particular virtual machine. If the specified percentage is exceeded, the machine becomes the lowest priority user in the system. If the percentage used is lower than that specified, the virtual machine has highest priority execution for the remainder of the interval. The percentage of CPU time assured is specified in the privileged class command that invokes the function.

CP can also assure that a designated user is never dropped from the active (in queue) subset by the scheduler. When the user is runnable, he is placed in the dispatchable list at his normal priority.

Reserved Page Frames

CP uses chained lists of table entries for 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. Paging proceeds using demand paging with a "reference bit" algorithm to select the best page for swapping. The reserved page frames option gives a particular virtual machine an essentially "private" set of pages. The pages are not locked, that is, they can be swapped, but usually only for the specified virtual machine. The number of reserved pages for the virtual machine are specified as a maximum. The page selection routine selects an available page for a reserved user and marks that page reserved if the maximum specified for the user has not been reached. If an available, unreferenced reserved page is encountered during page replenishment for the reserved user, it is used whether or not the maximum has been reached. If the page selection routine cannot locate an available page for other users because they are all reserved, the routine may steal the reserved pages.

Virtual=Real

This feature requires that the CP nucleus be reorganized to provide a "hole" in real storage large enough to contain the entire storage area of the virtual machine. For the virtual machine, each page from page one to the last page (n) is in its true real storage location; only page zero is relocated. The virtual machine runs in relocate mode, but because the virtual page address is the same as the real page address, no CCW translation is required for the virtual machine. Because no CCW translation is performed, no check is made of the I/O data addresses. The virtual machine must ensure that no I/O data transfer is specified into page zero or into any page not in the virtual machine's domain.

There are several considerations for the virtual=real option of preferred machine support 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 that machine.
- While allocated as such, the storage reserved for the virtual=real machine can be used only by a virtual machine with that option. It is not available to other users for paging space nor for VM/370 usage, even when the virtual=real machine is nct logged on. For this reason, the virtual=real machine should be a high availability, high throughput machine. The virtual=real storage can be released by the operator. That storage is then available for paging. Once virtual=real storage space is released by the operator, a VM/370 IPL is necessary to reallocate that storage to that virtual=real machine.
- The virtual machine with the virtual=real option operates in the pre-allocated storage area with normal CCW translation in effect until the execution of the SET NOTRANS ON command. At that time, all subsequent I/O operations are performed from the virtual CCWs in the virtual=real space without translation. In this mode, the virtual machine must not perform I/O operations into page zero nor beyond its addressable limit. Violation of this requirement may cause destruction of the VM/370 system and/or other virtual machines.
- If the virtual=real machine performs a virtual reset or IPL, the normal CCW translation is performed until the issuance of the SET NOTRANS ON command. Only the virtual=real virtual machine can issue the command. A message is issued if normal translation mode is entered.

Virtual Machine Assist Feature

The virtual machine assist feature is available with System/370 Models 135, 145, and 158 and as an RPQ on the System/370 Model 168. It improves the performance of VM/370. It intercepts and handles interruptions caused by SVCs, invalid page conditions and the following privileged instructions:

LRA	(load real address)
STCTL	(store control)
RRB	(reset reference bit)
ISK	(insert storage key)
SSK	(set storage key)
IPK	(insert PSW key)
STNSM	(store then and system mask)
STOSM	(store then or system mask)
SSM	(set system mask)
LPSW	(load PSW)
SPKA	(set PSW key from address)

Although virtual machine assist feature is designed tc improve the performance of VM/370, the virtual machines that do not have virtual machine assist feature available may see a performance improvement because the virtual machines with virtual machine assist feature are using less of the system resources leaving more resources available for the other users.

<u>VIRTUAL MACHINE CONTROL</u>: Real control register 6 (see Note 1) and a MICBLOK control virtual machine assist feature. The MICBLOK is a list of pointers to areas within VM/370 control blocks. The control register 6 format follows:

Bit	Meaning
0	1=virtual machine assist feature on for this virtual machine 0=virtual machine assist feature disabled (VM/370 mode)

Bit	Meaning
1	1=Virtual machine is in problem state 0=Virtual machine is in supervisor state (see Note 2)
2	1=ISK and SSK not handled by virtual machine assist feature 0=ISK and SSK handled by virtual machine assist feature
3	1=360 operations and 370 non-DAT operations only 0=370 DAT operations allowed (see Note 3)
4	1=SVC interruptions not handled by virtual machine assist feature 0=SVC interruptions handled by virtual machine assist feature
5	1=Shadow table mode: Shadow page fixup done by virtual machine assist feature 0=Shadow Table fixup not allowed
6	Reserved (must be zero)
7	Reserved (must be zero)
8-28	Real address of virtual machine pointer list
29-31	Unused (must be zero)

Notes:

1. Control register 6 is loaded before each virtual machine is dispatched.
2. Bit 1 of control register 6 may be changed by virtual machine assist feature during a virtual machine status change.
3. Bit 3 affects instructions that only a virtual machine with the EC option may execute. Specifically, they are: LRA, RRB, IPK, STNSM, STOSM, and SPKA. Bit 3 also affects STCTL even though it can be executed by a virtual machine without the EC option.

Virtual machine assist feature uses the list of pointers, or MICBLOK, to access virtual machine control information. The list must start on a doubleword boundary. A MICBLOK is obtained for each user when he logs on. The entries in this list are as follows:

- Real segment table pointer and length, page size, and segment size.
- Pointer to the real address of virtual control register 0.
- Pointer to the real address of the virtual PSW currently in effect.
- Pointer to the 64 byte workspace area reserved for virtual machine assist feature.

INTERACTION WITH PROGRAM EVENT RECORDING (PER): For all instructions in virtual machine assist feature except SVC and LPSW, PER monitoring events are indicated normally as if the instructions were being executed in supervisor state. Changes made to the virtual PSW or swap table entries in VM/370 real storage are indicated as storage alteration events, because those locations are considered to be internal registers to the virtual machine. A virtual instruction that attempts to change the state of the virtual PSW PER mask causes a privileged instruction interruption, and the instruction is suppressed.

For virtual SVC interruption, PER monitoring specified in the current real PSW, current virtual PSW, or new virtual PSW causes a real SVC interrupt, regardless of the values specified in real or virtual control registers 9, 10, and 11. For virtual LPSW, similar conditions result in a real privileged instruction interruption.

PER monitoring specified in the real PSW causes the VM/370 page invalid interruption to be inactive.

Privileged instruction interruptions resulting from the virtual instructions may show a PER event for instruction fetching, just as they would without the feature. Real SVC interruptions may be followed by a program interruption for an instruction fetch PER event.

INTERACTION WITH DOS EMULATOR: On machines with both virtual machine assist feature and the DOS Emulation feature installed, local execution (LEX) mode inactivates virtual machine assist feature; privileged instruction interruptions and SVC interruptions occur according to DOS emulation architecture. When the machine is not in LEX mode, the machine performs as described for virtual machine assist feature.

RESTRICTED USE OF VIRTUAL MACHINE ASSIST FEATURE: Certain interruptions must be handled by VM/370. Consequently, the virtual machine assist feature is not on in a virtual machine if the machine has instruction address stop set on.

VM/370 turns SVC handling off when instruction address stop is set on, and turns it back on after the stop occurs.

VM/VS HANDSHAKING

The VM/VS Handshaking feature provides a communication path between CP and a virtual machine operating system (OS/VS1) that makes each system control program aware of certain capabilities or requirements of the other. VM/VS Handshaking 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/370 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/370 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/370. If VS1 finds a version code of X'FF', it then issues a DIAGNOSE (X'00') instruction to store the VM/370 extended-identification code. If an extended-identification code is returned to VS1, VS1 knows that VM/370 supports handshaking; if nothing is returned to VS1, VM/370 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/370 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/370 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/370 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/370 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/370 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 tasks.

When a page fault occurs for a VS1 virtual machine, VM/370 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/370 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/370 indicates the same program interruption to VS1, except the high order 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/370, it executes in nonpaging mode if:

* Its virtual storage size is equal to the size of the VM/370 virtual machine
* Its virtual machine size is at least one megabyte and no more than four megabytes.
* 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/370 provides. For example, VS1 avoids:

* ISK (Insert Storage Key) instructions and uses a key table
* Seek separation for 2314 direct access devices
* ENABLE/DISABLE sequences in the VS1 I/O Supervisor (IOS)

- TCH (Test Channel) instructions preceding SIO (Start I/O) instructions.

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 15.

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 transparent to the virtual machine. Most other program interruptions result from virtual machine processing and are reflected to the virtual machine for handling.

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.

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 of the following actions being taken:

- System termination
- Selective virtual user termination
- 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/370 startup (before the system is set up well enough to permit RMS to operate successfully), the CPU goes into a disabled wait state and places a completion code of X'00B' in the high-order bytes of the current PSW.

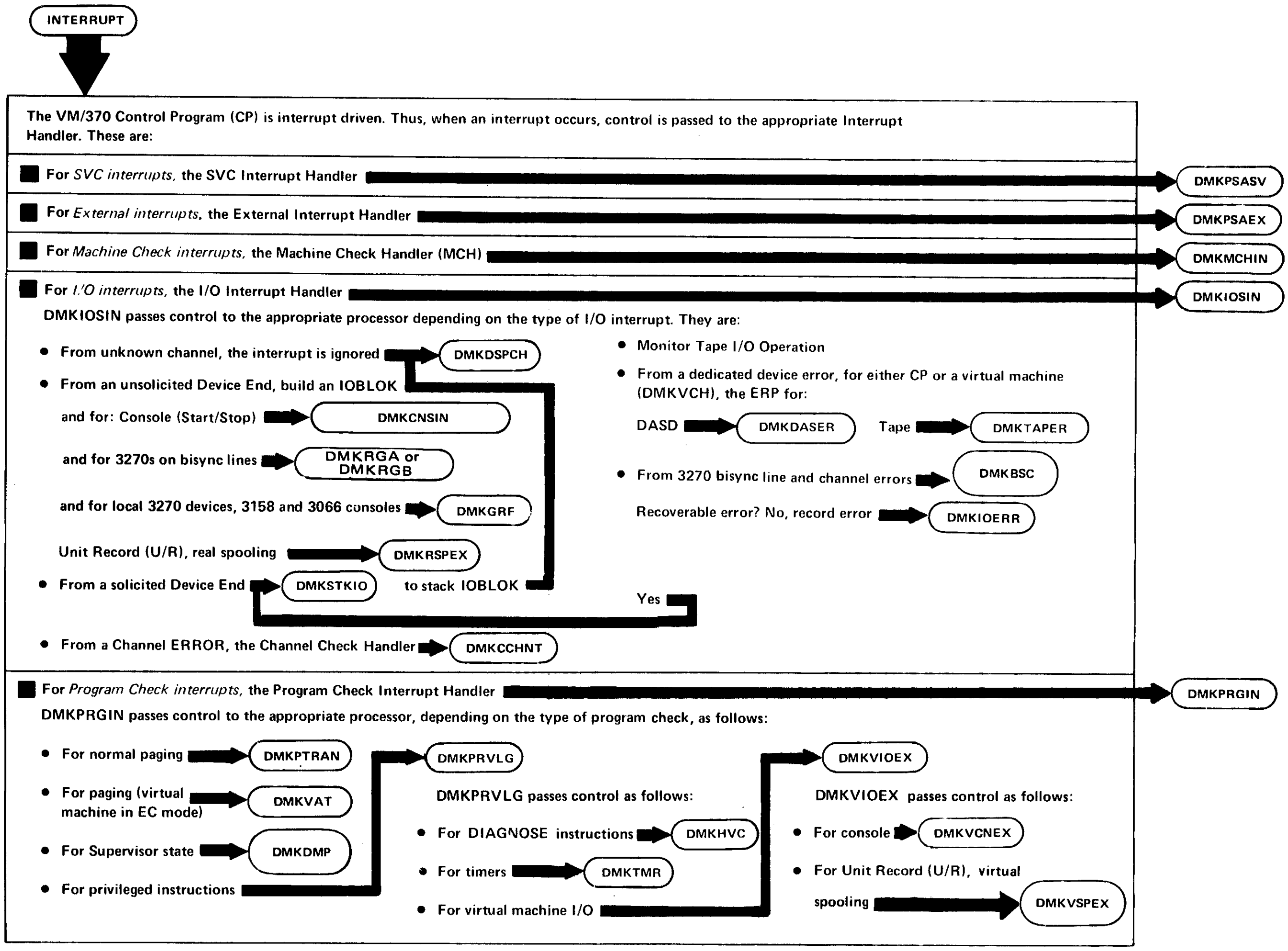


Figure 15. Overview of Interruption Handling

SVC Interruption

When an SVC interruption occurs, the SVC interruption routine is entered. If the machine is in problem state, the type of interruption is usually reflected back to the pseudo-supervisor (that is, the supervisor operating in the user's virtual machine). If the machine is in supervisor state, the SVC interruption code is determined, and a branch is taken to the appropriate SVC interruption handler.

External Interruption

If a timer interruption occurs, CP processes it according to type. The interval timer indicates time-slice end for the running user. The clock comparator indicates that a specified timer event has occurred, such as the time of day, a scheduled shutdown, or a reached user event. The CPU timer indicates that a virtual machine's allowed execution interval (time in queue) has expired.

The external console interruption invokes CP processing to switch from the primary console to an alternate operator's console.

FREE STORAGE MANAGEMENT

During its execution, CP occasionally requires small blocks of storage that are used for the duration of a task. CP obtains this storage from the free storage area. The free storage area is divided into various size subpools. The requestor informs the free storage manager of the size of the block required and the smallest available subpool that fulfills the request is allocated to the requestor. When the block is no longer needed, the requestor informs the free storage manager and CP returns the block to free storage.

If the request for free storage cannot be fulfilled, the free storage manager requests the temporary use of a page of storage from the dynamic paging area. If a page is obtained, the page is chained to the free storage area and used for that purpose until it is no longer needed and subsequently returned to the dynamic paging area.

If the request for a page cannot be fulfilled, the requestor waits until free storage becomes available.

STORAGE PROTECTION

VM/370 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 of 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 CPU accesses storage, and protection applies, the protection key of the current PSW is used as the comparand. The protection key of the CPU is bit positions 8-11 of the PSW.

If the CPU 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 referring to the data areas to be protected, with the fetch protect bit in the key. VM/370 subsequently:

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

A special case of storage protection occurs when the CMS nucleus resides in a shared segment. The nucleus must be protected and still be shared by many CMS users. The program interruption handler, DMKPRG, manipulates the real storage key and real PSW key to ensure that user programs and disk-resident commands run with a different key than the nucleus code.

EXECUTING THE PAGEABLE CONTROL PROGRAM

Calls to pageable routines are recognized at execution time by the SVC 8 linkage manager in DMKPSA. 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 reentrant, 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 used by page selection tends to make highly used control program routines, even when not locked, remain resident. The called routine is locked into real storage until it exits. Thus, it can request asynchronously scheduled function, 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, the fact that the module is executing at an address different from that at which it was loaded is transparent. Although part of CP is pageable, it never runs in relocate mode. Thus, the CPU 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 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 CPU 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/370 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/370 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 should be completely reenterable. Any messages to be modified, temporary work or scratch areas, or program switches must be allocated from system free storage or from the caller's save area.

- The module must be entered by the standard SVC 8 CALL linkage. Modules entered by BALR or GOTO cannot be pageable.

- 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 four 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/370 Maintenance Procedures" of the _VM/370: Service Routines Program Logic_. Additional information can be found in the _VM/370: Planning and System Generation Guide_.

DATA AREA MODULES

In addition to the executable resident and pageable modules, 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
DMKSYS	System constants
DMKTBL	Terminal translate table

Pageable Module	Contents
DMKBOX	Output separator table
DMKEMS	Error message data module
DMKFCB	3211 Forms Control Buffer (FCB) load tables
DMKSNT	System name table
DMKSYM	System symbol table
DMKUCB	3211 Universal Character Set Buffer (UCSB) load tables
DMKUCS	1403 Universal Character Set (UCS) load tables
DMKTBM	Terminal translate tables

INTERRUPTION HANDLING

SVC INTERRUPTIONS

When an SVC interruption occurs, the SVC interruption routine (DMKPSASV) is entered. If the machine is in the problem state, DMKPSASV takes the following action:

EXECUTABLE MODULES

Executable Resident Modules

DMKBSC	DMKFRE	DMKPGT	DMKRSP
DMKCCH	DMKGRF	DMKPRG	DMKSCH
DMKCCW	DMKHVC	DMKPRV	DMKSCN
DMKCFC	DMKHVD	DMKPSA	DMKSTK
DMKCFM	DMKIOE	DMKPTR	DMKTMR
DMKCNS	DMKIOS	DMKQCN	DMKUNT
DMKCVT	DMKLOC	DMKRGA	DMKVAT
DMKDAS	DMKMCH	DMKRGB	DMKVCN
DMKDGD	DMKMSW	DMKRGF	DMKVIO
DMKDMP	DMKOPR	DMKRNH	DMKVMA
DMKDSP	DMKPAG	DMKRPA	DMKVSP

Executable Pageable Modules

DMKACO	DMKCSO	DMKLOG	DMKTAP
DMKBLD	DMKCSP	DMKMCC	DMKTDK
DMKCDB	DMKCST	DMKMID	DMKTHI
DMKCDS	DMKCSU	DMKMON	DMKTRA
DMKCFD	DMKDEF	DMKMSG	DMKTRC
DMKCFG	DMKDIA	DMKNEM	DMKTRM
DMKCFP	DMKDRD	DMKNES	DMKUDR
DMKCFS	DMKEIG	DMKNET	DMKUSO
DMKCFT	DMKEMA	DMKNLD	DMKVCA
DMKCKP	DMKEMB	DMKPGS	DMKVCH
DMKCKS	DMKERM	DMKRSE	DMKVDB
DMKCPB	DMKGIO	DMKSAV	DMKVDR
DMKCPI	DMKIOC	DMKSEP	DMKVDS
DMKCPV	DMKIOF	DMKSEV	DMKVER
DMKCQG	DMKIOG	DMKSIX	DMKVMI
DMKCQP	DMKISM	DMKSNC	DMKWRM
DMKCQR	DMKLNK	DMKSPL	

- 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), DMKPSA 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 was in EC mode or its 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 was in BC mode and if page 0 is in main storage, an appropriate SVC old PSW is stored in page 0 and the interruption is reflected to the virtual machine, bypassing unnecessary register saving. If the new virtual PSW indicates the wait state, all registers are saved in the VMBLOK and control transfers to DMKDSPB for PSW validation.

If the machine is in the supervisor state, the SVC interruption code is determined 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.

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 in the pageable module DMKPTRUL, then DMKPGSUL 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 SVCCPSW.

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.

There are 35 SAVEAREAs initially set up by DMKCPINT for use by the SVC linkage handlers. If all the save areas are used, the linkage handlers call DMKFREE to obtain additional save areas.

EXTERNAL INTERRUPTIONS

Timer Interruption

If DMKPSAEX is entered because of a timer interruption, the state of the machine must be determined. If the machine was in wait state, control is transferred to DMKDSPCH, and the machine stays idle until another interruption occurs. If the 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."

External Interruption

If DMKPSAEX is entered because the operator pressed the console interrupt button (INTERRUPT), the following steps are taken:

- The current system operator's VMBLOK (DMKSYSOP) is referenced.
- The virtual machine is disconnected.

The operator can now log on from another terminal. Pressing the console interrupt button activates an alternate operator's console. For a description of the processing of the external interruption command, refer to module DMKCPB in Section 2.

Extended Virtual External Interruptions

To reflect external interruptions to a virtual machine, an XINTBLOK is queued 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 low-order 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.

PROGRAM INTERRUPTIONS

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.
- Program errors.

For information paging requests, see "Allocation Management" in this section.

Privileged Instructions

If a program interruption is caused by the virtual machine issuing a privileged instruction when it is running in supervisor state, DMKPRVLG 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 (DMKVIOEX).

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, and DIAGNOSE.

Although the CS and CDS instructions are non-privileged, they are not part of the standard instruction set on IBM System/370 Models 135 and 145. VM/370 simulates these instructions on Models 135 and 145 that do not have the optional hardware feature installed.

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

DIAGNOSE Interface (DMKHVC)

The DIAGNOSE command communicates between a virtual machine and CP. Correct CP execution of DIAGNOSE requires that the operand storage addresses passed to CP via the DIAGNOSE interface be real addresses to the virtual machine using the DIAGNOSE instruction. In VM/370, the machine-coded format for the DIAGNOSE command is:

Bits 0–7	8–11	12–15	16–31
83	rx	ry	code

Content	Explanation
83	DIAGNOSE operation code
rx	User-specified register number
ry	User-specified register number
code	Hexadecimal value that selects a particular VM/370 control program function. The codes and their associated functions are:

Code	Class	Function
0000	Any	Store extended-identification code.
0004	C,E	Examine data from real storage
0008	G	Execute CP console function
000C	G	Pseudo-timer facility
0010	G	Release virtual storage pages
0014	G	Manipulate input spool files
0018	G	Standard DASD I/O
001C	F	Clear I/O and machine check recording
0020	G	General virtual I/O without interrupts
0024	G	Virtual device type information
0028	G	Dynamic TIC modification
002C	C,E,F	Return DASD start of LOGREC area
0030	C,E,F	Read one page of LOGREC data
0034	C,E	Read system dump spool file
0038	C,E	Read system symbol table
003C	A,B,C	Dynamically update system user directory
004C	Any	Generate accounting cards for virtual user
0050	A,B,C	Save 3704/3705 control program image
0054	G	Enable or disable for external interruption
0058	G	Virtual console interface for 3270
005C	G	Error message editing
0060	G	Provide virtual machine storage size
0064	G	Load, find, or purge a named system

Rules for DIAGNOSE codes:

1. The DIAGNOSE code must always be a multiple of 4.
2. Virtual machines issuing DIAGNOSE codes should run with interruptions disabled to prevent loss of status information (condition code, sense, etc.) pertaining to the DIAGNOSE operation.

X'00' through X'FC' --Reserved for IBM use
X'100' through X'1FC' --Reserved for users

DIAGNOSE CODE 0: Allows a virtual machine to examine the VM/370 extended-identification code. For example, an OS/VS1 virtual machine issues a DIAGNOSE code 0 instruction to determine if the version of VM/370 it is running in supports the VM/VS Handshaking feature. If the extended-identification code is returned to VS1, VM/370 supports handshaking; otherwise, it does not.

where:

rx contains the virtual storage address where the VM/370 extended-identification code is to be stored.
ry is the number of bytes to be stored (an unsigned binary number).

If the VM/370 system currently running does not support the DIAGNOSE code 0 instruction, no data is returned to the virtual machine. If it does support the DIAGNOSE code 0 instruction, the following data is returned to the virtual machine (at the location specified by rx):

Field	Description	Attributes
System Name	VM/370	8 bytes, EBCDIC
Version Number	The first byte is the version number, the second is the level, and the third is the program level change (PLC) number.	3 bytes, hex
Version Code	VM/370 executes the STORE CPU ID (STIDP) instruction to determine the version code.	1 byte, hex
MCEL	VM/370 executes the STIDP instruction to determine the maximum length of the machine check extended logout area (MCEL).	2 bytes, hex
CPU Address	VM/370 executes the STORE CPU ADDRESS (STAP) instruction to determine the processor address.	2 bytes, hex
userid	The userid of the virtual machine issuing the DIAGNOSE.	8 bytes, EBCDIC

The condition code remains unchanged by the DIAGNOSE code 0 instruction.

DIAGNOSE CODE 4: Examines real storage.

where:

rx contains the virtual address of a list of CP (real) addresses.

ry contains a count of entries in the list.

ry+1 contains the virtual address of the result field that holds the values retrieved from CP locations.

DIAGNOSE CODE 8: Allows a virtual machine to perform CP console functions.

where:

rx contains the address (virtual) of the CP console function command and parameters.

ry contains the length of the associated console function input, up to 132 characters.

The following example illustrates the virtual console function:

```
         LA    R6,CPFUNC
         LA    R10,CPFUNCL
         DC    X'83',X'6A',XL2'0008'
         .
         .
         .
CPFUNC   DC    C'QUERY FILES'
CPFUNCL  EQU   *-CPFUNC
```

The console function output goes to the user's terminal, and then execution continues. Any valid and authorized console function can be executed in this manner.

A completion code is returned to the user as a value in the register specified in ry. A completion code of 0 signifies normal completion. If an error message is issued, the completion code is equal to the numeric portion of the error message.

DIAGNOSE CODE C: Obtains total and virtual CPU time.

where:

rx contains the virtual address of a 32-byte data area that does not cross a page boundary, into which the following data is stored:

Bytes 0 – 7	8 – 15	16 – 23	24 – 31
MM/DD/YY	HH:MM:SS	Virt CPU	Total CPU

Virtual and total CPU time used (in microseconds) is returned as a doubleword logical value.

DIAGNOSE CODE 10: Releases pages.

where:

rx contains the virtual address of the first page to be released.

ry contains the virtual address of the last page to be released.

Any of the virtual pages in real or auxiliary storage are released.

DIAGNOSE CODE 14: Performs input spool file manipulation.

where:

rx contains either a buffer address, a copy count, or a spool file identifier, depending on the value of the function subcode in ry+1.

ry (cannot be register 15) contains either the virtual address of a spool-input card reader or, if ry+1 contains X'0FFF', a spool file ID number.

ry+1 contains a hexadecimal function code interpreted by DMKDRDER as a command to do the following:

Code	Function
0000	Read next spool buffer (data record)
0004	Read next print SFBLOK
0008	Read next punch SFBLOK
000C	Select a file for processing
0010	Repeat active file nn times
0014	Restart active file at beginning
0018	Backspace one record
0FFF	Retrieve successor file descriptor

Note: ry+1 on return, may contain error codes which further define a returned condition code of 3.

The file manipulation is performed by DMKDRDER.

DIAGNOSE CODE 18: Performs limited disk I/O.

where:

rx contains the device address of the disk.

ry points to a CCW chain to read or write a limited number of disk records.

Each read or write must specify no more than 2048 bytes (usually 800 is used), and the CCW chain is of a standard form, as shown below. For a 3330 or 3350 a SET SECTOR command would precede each SRCH command.

Register 15 contains the number of reads or writes in the CCW chain (the number is two in the following example for a typical CCW string (to read or write two 800-byte records):

```
    SEEK,A,CC,6
    SRCH,A+2,CC,5
    TIC,*-8,0,0
    RD or WRT,DATA,CC+SILI,800
    SEEK HEAD,B,CC,6 (Omitted if HEAD No.
    SRCH,B+2,CC,5                unchanged)
    TIC,*-8,0,0
    RD or WRT,DATA+800,SILI,800
A   SEEK and SRCH arguments for first RD/WRT
B   SEEK and SRCH arguments for second RD/WRT
```

The condition codes (cc) and completion codes that are returned are as follows:

cc=0 I/O complete with no errors

cc=1 Error condition. Register 15 contains one of the following:
- 1 Device not attached
- 2 Device not 2314, 2319, 3330, 3340, or 3350
- 3 Attempt to write on a read-only disk
- 4 Cylinder number is not in the range of user's disk
- 5 Virtual device is busy or has an interrupt pending

cc=2 Error condition. Register 15 contains one of the following:
- 5 Pointer to CCW string not doubleword aligned.
- 6 SEEK/SEARCH arguments are not within range of user's storage
- 3 READ/WRITE CCW is neither read (06) nor write (05)
- 8 READ/WRITE byte count=0
- 9 READ/WRITE byte count greater than 2048
- 10 READ/WRITE buffer not within user's storage

cc=3 Uncorrectable I/O error. Register 15 contains the following:
- 13 CSW (8 bytes) returned to user.

Sense bytes are available if user issues a SENSE command.

DIAGNOSE CODE 1C: Calls the DMKIOEFM routine to clear the I/O error recording data on disk.

where:

rx contains the code value 1, 2, or 3 to clear and reformat the I/O error recording, M/C recording, or both.

ry is ignored.

DIAGNOSE CODE 20: Performs general I/O without interruptions.

where:

rx contains a virtual tape or DASD device address.

ry contains the address of the string of CCWs to be executed.

The CCW string is processed by DMKCCWTR through DMKGIOEX. This provides full virtual synchronous I/O to any virtual tape or DASD device specified (self-modifying CCW strings are not permitted, however). Control returns to the virtual machine only after completion of the operation or detection of a fatal error condition. Condition codes and error codes are returned to the virtual system. Unit record devices are not supported.

The condition codes (cc) and completion codes that are returned follow:

cc=0 I/O complete with no errors

cc=1 Exception conditions. Register 15 contains the following:
- 1 Device not attached
- 5 Virtual device is busy or has an interrupt pending

cc=2 Exception conditions. Register 15 contains one of the following:
- 2 Unit exception bit in device status byte=1
- 3 Wrong length record detected

cc=3 Error condition. Register 15 contains the following:
- 13 A permanent I/O error occurred. The two low order positions of the user's R2 register contain the first two sense bytes.

Note: Support is provided for DASD and tape devices only. All other devices have a return code of 13 and a condition code of 3 in the virtual machine's PSW.

DIAGNOSE CODE 24: Provides virtual device type information.

where:

rx contains a virtual device address, or a value of -1 indicating a virtual console whose address is not known.

If rx contained a value of -1 upon entry and a virtual console was found, the register contains the virtual device address in the two low order bytes, upon return.

ry contains virtual device information.

ry+1 contains real device information. If ry is register 15, then only virtual device information is supplied.

Bits	0 — 7	8 — 15	16 — 23	24 — 31
ry	VDEVTYPC	VDEVTYPE	VDEVSTAT	VDEVFLAG
ry+1	RDEVTYPC	RDEVTYPE	RDEVMDL	see Note

Note: The low order byte of ry+1 contains the current device line length (RDEVLLEN) for a virtual console, or the device feature code (RDEVFTR) for a device other than a virtual console. The condition codes (cc) and completion codes that are returned follow:

cc=0 Data transfer successful

cc=2 Real device does not exist

cc=3 Device address invalid; or device does not exist

DIAGNOSE CODE 28: Modifies a real TIC or NOP CCW in a channel program when the associated virtual TIC or NOP has been modified after a START I/O but before a channel or device end interruption.

where:

rx contains the address of the TIC or NOP CCW that has been modified. The address in rx, the new address in the modified TIC CCW, and the addresses in the new CCW list pointed to by the modified TIC, must be "real" with respect to the virtual machine; they must be "second level" virtual storage addresses to CP.

ry contains the virtual device address in bits 16 through 31. (Must be a different register than rx.)

When DMKHVC has analyzed the DIAGNOSE, the real CCW string and appropriate TIC or NOP is located by a call to DMKCCWTC. If a virtual TIC had been changed to a NOP, a corresponding change is made to the real TIC. If a virtual TIC had been changed to point to a new list of CCWs or a virtual NOP had been changed to a TIC, the program translates the new list via a call to DMKCCWTR and modifies the existing channel program to include the new real CCWs. If an error was detected in the DIAGNOSE information, or if it was too late to change the real CCW list because channel or device end had already occurred, a condition code and return code are returned to the virtual machine to notify it that the real CCW string was not successfully modified. The condition codes are as follows:

Condition Code	GR15	Explanation
0	0	Successfully modified channel program
1	1	rx and ry registers are the same.
1	2	Device specified by ry register was not found.
1	3	Address given to rx register was not within user's storage space.
1	4	Address given by rx register was not doubleword aligned.
1	5	CCW string corresponding to ry device and rx address was not found.
1	5	CCW string corresponding to ry and rx address was not found.
1	6	CCW at address specified by rx is not a TIC or a NOP, or CCW in channel program is not a TIC or a NOP.
1	7	New address in TOC is not within user's storage space.
1	8	New address in TOC is not doubleword aligned.
2	9	Too late to change the CCW string (channel end or device end has already occurred).

DIAGNOSE CODE 2C: Returns the DASD start location of the LOGREC area.

where:

rx on return contains the DASD location, in CP format, of the first record of the system I/O and machine check error recording area.

ry is ignored.

DIAGNOSE CODE 30: Reads one page of LOGREC data.

where:

rx contains the DASD location, in CP format, of the desired record.

ry contains the virtual address of a page-size buffer to receive the data.

The page of data is provided to the virtual machine via DMKRPAGT. The condition codes are as follows:

cc=0 Successful read, data available.
cc=1 End of cylinder, no data.
cc=2 Invalid cylinder, outside recording area

DIAGNOSE CODE 34: Reads the system dump spool file.

where:

rx contains the virtual address of a page-size buffer to accept the requested data.

ry (cannot be register 15) contains the virtual device address of a spool-input card reader.

ry+1 on return, may contain error codes which further define a returned condition code of 3.

The system chain of spool input files is searched for a dump file belonging to the user issuing the DIAGNOSE by DMKDRDMP. The first (or next) record from the dump file is provided to the virtual machine via DMKRPAGT and the condition code is set to zero. The dump file is closed by the CLOSE command issued from the console.

DIAGNOSE CODE 38: Reads the system symbol table.

where:

rx contains the start address of the page buffer that is to contain the symbol table.

ry is ignored.

The system symbol table (DMKSYM) is read into storage by DMKDRDSY at the location specified by rx.

DIAGNOSE CODE 3C: Dynamically updates the system user directory by DMKUDRDS.

where:

rx contains the first 4 bytes of the volume serial label.

ry the first 2 bytes of the register specified by ry contain the last 2 bytes of the volume serial label.

DIAGNOSE CODE 4C: Generates accounting cards for the virtual user. This code can be issued only by a user with the account option (ACCT) in his directory.

where:

rx contains the virtual address of either a 24-byte parameter list identifying the "charge to" user, or a variable length data area that is to be punched into the accounting card. The interpretation of the address is based on a hexadecimal code supplied in ry. If the virtual address represents a parameter list, it must be doubleword aligned; if it represents a data area, the area must not cross a page boundary. If rx is interpreted as pointing to a parameter list and the value in rx is zeros, the accounting card is punched with the identification of the user issuing the DIAGNOSE.

ry contains a hexadecimal function code interpreted by DMKHVC as follows:

Code	Meaning
0000	rx points to a parameter list containing only a userid.
0004	rx points to a parameter list containing a userid and account number.
0008	rx points to a parameter list containing a userid and distribution number.
000C	rx points to a parameter list containing a userid, account number, and distribution number.
0010	rx points to a data area containing up to 70 bytes of user information to be transferred to the accounting card starting in column nine.

Note: If ry contains X'0010', ry cannot be register 15.

ry+1 contains the length of the data area pointed to by rx. If rx points to a parameter list (ry not equal to X'0010'), ry+1 is ignored.

The following condition codes are returned to the user by DMKHVC:

CC	Meaning
0	Successful operation
1	User does not have account option privileges
2	Invalid userid in the parameter list
3	Invalid function hexadecimal code in ry or an error occurred in trying to read in the user machine block (UMACBLOK)

DMKHVC checks the VMACCOUN flag in VMPSTAT to verify that the user has the account option and if not, returns control to the user with a condition code of 1.

If ry contains a code of X'0010', DMKHVC performs the following checks:

- If the address specified in rx is negative or greater than the size of the user's virtual storage, an addressing exception is generated.
- If the combination of the address in rx and the length in ry+1 indicates that the data area crosses a page boundary, a specification exception is generated.
- If the value in ry+1 is zero, negative or greater than 70, a specification exception is generated.

If both the virtual address and the length are valid, DMKFREE is called to obtain storage for an account buffer (ACNTBLOK) which is then initialized to blanks. The userid of the user issuing the DIAGNOSE is placed in columns 1 through 8 and an accounting card identification code of "C0" is placed in columns 79 and 80. The user data pointed to by the address in rx is moved to the accounting card starting at column 9 for a length equal to the value in ry+1. A call to DMKACOQU queues the ACNTBLOK for real output. If a real punch is available, DMKACOPU is called to punch the card; otherwise, the buffer is stored in main storage until a punch is free. DMKHVC then returns control to the user with a condition code of 0.

If ry contains other than X'0010' code, control is passed to DMKCPV to generate the card. DMKCPV passes control to DMKACO to complete the "charge to" information; either from the user accounting block (ACCTBLOK), if a pointer to it exists, or from the user's VMBLOK. DMKCPV then punches the card and passes control back to DMKHVC to release the storage for the ACCTBLOK, if one exists. DMKHVC then checks the parameter list address for the following conditions:

- If zero, control is returned to the user with a condition code of 0.
- If invalid, an addressing exception is generated.
- If not aligned on a doubleword boundary, a specification exception is generated.

For a parameter list address that is nonzero and valid, the userid in the parameter list is checked against the directory list and if not found, control is returned to the user with a condition code of 2. If the hexadecimal code in ry is invalid, control is returned to the user with a condition code of 3. If both userid and hexadecimal code are valid, the user accounting block (ACCTBLOK) is built and the userid, account number, and distribution number are moved to the block from the parameter list or the user machine block belonging to the userid in the parameter list. Control is then passed to the user with a condition code of 0.

DIAGNOSE CODE 50: Saves the 3704/3705 control program image.

When a 3704/3705 control program module has been created, the CMS-based service routine (SAVENCP) builds a parameter list (see CCPARM in SAVENCP data areas) of control information required by CP to load the module into the user's virtual storage. It passes this information to CP by a DIAGNOSE code X'0050'.

where:

rx contains the virtual address of the parameter list (CCPARM)

ry is ignored on entry.

DIAGNOSE code X'0050' invokes DMKSNC to validate the parameter specifications and write the page-format image of the control program to the appropriate system volume.

ry upon return, contains the following error codes:

Code	Explanation
044	"ncpname" was not found in system name table.
171	System volume specified not currently available.
176	Insufficient space reserved for program and system control information.
179	System volume specified is not a CP owned volume.
435	Paging error while writing saved system.

DIAGNOSE CODE 54: Sets a flag in the VMQSTAT to reflect an external interruption to the virtual machine when the PA2 key is pressed on a 3270 keyboard with the APL feature activated.

DIAGNOSE CODE 58: Virtual console interface for 3270.

Execution of DIAGNOSE code 58 allows a virtual machine to quickly display large amounts of data on a 3270 in a very rapid fashion. The interface can display up to 1760 characters on the screen with one write operation instead of up to 22 individual writes, if each line was limited to 80 characters.

where:

rx contains the address of the console CCW string. The format of the special display CCW is:

CCW X'19', dataddr, flags, ctl, count

where:

dataddr	is the address of the first byte to be displayed.
flags	is the standard CCW flag field.
ctl	is a control byte indicating the starting output display line (0-22). If the high-order bit is on, the entire 3270 output display area is erased before the new data is displayed. A value of X'FF' clears the screen, but writes nothing.
count	is the number of bytes to be displayed (maximum is 1760).

ry contains the address of the virtual console device in bits 16-31.

If this CCW is issued to a virtual console that is not simulated on a real 3270, a virtual command reject is generated. Otherwise, a buffer is built in free storage and the data pointed to by 'dataddr' is loaded into it. Data chaining may be specified in the CCW to link noncontiguous data areas; however, command chaining is an end-of-data indication for the current buffer.

The starting line specified in 'ctl' is correlated with the data 'count' to ensure that the data does not overflow the allowed area. Any invalid specification will generate a command reject.

CP then processes the display CCW returning a condition code of zero if the display was successful or a nonzero code if an I/O error occurred.

DIAGNOSE CODE 5C: Edits error messages. Execution of DIAGNOSE Code 5C edits an error message according to the user's setting of the EMSG function.

where:

rx contains the address of the message to be edited.

ry contains the length of the message to be edited.

DMKHVC tests the VMMLEVEL field of the VMBLOK and returns to the caller with rx and ry modified as described in Figure 16.

VMMLEVEL		Registers Upon Return	
VMMCODE	VMMTEXT	rx	ry
On	On	No change	No change
On	Off	No change	10 (length of code)
Off	On	Pointer to text part of message	Length of text alone
Off	Off	N/A	0

Figure 16. DIAGNOSE X'5C'/VMMLEVEL Field Analysis

Note that DIAGNOSE code X'5C' does not write the message; it merely rearranges the starting pointer and length. For CMS error messages, a console write is performed following the DIAGNOSE, unless ry is returned with a value of 0.

DIAGNOSE CODE 60: Returns the virtual machine storage size to the user. CMS issues this DIAGNOSE during initialization.

where:

rx contains the virtual storage size.

DIAGNOSE CODE 64: Allows any virtual machine to dynamically load, purge, or find a named system in its virtual storage. CMS uses this DIAGNOSE to support DOS simulation.

rx contains the address of the NAMED system. The name must occupy eight bytes, be left justified and padded with trailing blanks.

ry the contents must be a multiple of 4 and its value cannot be greater than decimal 12.

ry=00 loads the named system in shared mode and attach to the user's virtual storage.

ry=04 loads the named system in nonshared mode and attach to the user's virtual storage.

ry=08 release the named system from virtual storage.

ry=0C finds the starting address of the named system.

If the address in the rx register is invalid an addressing exception occurs. If the code in the ry register is invalid, a specification exception occurs.

LOADSYS DIAGNOSE function: Execution of the LOADSYS DIAGNOSE function (ry=00 or 04) causes the control program to locate the name and location of the named system and builds the necessary page/swap tables. All virtual storage pages into which the system is to be loaded are released prior to loading the named system. When the LOADSYS DIAGNOSE is invoked, the virtual machine's storage is expanded dynamically, if necessary, and is completely transparent to the virtual machine. Whenever the LOADSYS function is invoked, an automatic PURGESYS function occurs prior to building new page/swap tables. The automatic PURGESYS allows virtual machines to switch back and forth from shared systems to nonshared systems.

When the LOADSYS function is executed in shared mode and the virtual machine has the CP trace facility active, the following options are reset if they are active: instruction trace and branch trace. All other options remain in effect. If no other tracing options are active, the user receives the message: TRACE ENDED.

Note: If the LOADSYS function is executed in shared mode, the virtual machine assist feature is reset.

Return Codes for LOADSYS

- Successful Execution

PSW CONDITION CODE = 0
User's rx = address of where the named system has been loaded.

PSW CONDITION CODE = 1
User's rx = address of where the named system has been loaded and also the starting address of virtual storage released prior to loading the named system.

User's ry = ending address of virtual storage released prior to loading the named system.

Note: A condition code of one in the user's PSW is reflected only when the named system to be loaded resides within the virtual machine size.

- Unsuccessful Execution

PSW CONDITION CODE = 2
User's rx = Return code of 44 if the named system does not exist.
User's ry = Return code of 174 if paging I/O errors occur.

PURGESYS DIAGNOSE function: The PURGESYS function releases storage made addressable to the virtual machine when the LOADSYS function was executed. The PURGESYS function releases page/swap tables associated with the named system. If the area released occupied a storage address range greater than the virtual machine storage size, this area is now made nonaddressable to the virtual machine. If the named system was being operated on in a nonshared mode, the storage which contained the named system is cleared to binary zeros. If the PURGESYS function is executed for a named system which had not been loaded by the LOADSYS function, no action takes place and the command is treated as a NOP. The PURGESYS function is invoked dynamically by the control program when a LOADSYS function is executed. The name of the purged named system is the same as that requested via the LOADSYS function.

Return Codes for PURGESYS

- Successful Execution

  PSW CONDITION CODE = 0

- Unsuccessful Execution

  PSW CONDITION CODE = 1
  This occurs if the named system was not found in the user's virtual storage.

  PSW CONDITION CODE = 2
  User's ry = Return code of 44 if the named system does not exist or is inactive.

FINDSYS DIAGNOSE function: The FINDSYS function determines where the named system will be loaded into storage, without actually loading it. FINDSYS also determines whether or not the named system has already been invoked by this virtual machine. FINDSYS is executed by CMS prior to issuing the LOADSYS DIAGNOSE instruction. This ensures that the named system to be loaded does not overlay any part of the CMS nucleus and that the named system is not already active (loaded) in the virtual machine. If the named system is active, no subsequent LOADSYS DIAGNOSE is issued, therby keeping the current copy of the named system active. The address of where the named system resides is returned in the user's rx register.

Return Codes for FINDSYS

- Successful Execution

  PSW CONDITION CODE = 0
  User's rx = address of where the named system resides in virtual storage.
  User's ry = ending address of where the named system resides in virtual storage.

  PSW CONDITION CODE = 1
  User's rx = beginning address of where the named system is loaded into virtual storage.
  User's ry = ending address of where the named system is loaded into virtual storage.

- Unsuccessful Execution

  PSW CONDITION CODE = 2
  User's rx = return code of 44 if the named system does not exist.
  User's ry = return code of 174 if paging I/O errors occur.

  Note: Condition code 0 indicates that the named system already resides in main storage. Condition code 1 indicates that the named system exists but has not been previously loaded in virtual storage.

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/370 uses the timing facilities of the real machine.

1. The location X'50' interval timer is used only for timeslicing. The value placed in the timer is the maximum length of time that the dispatched virtual machine is allowed to execute.

2. 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.

3. 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.

4. The CPU timer facility performs three functions:

   - Accumulates CP overhead
   - Detects in-queue time slice end
   - Simulates virtual CPU 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 CPU timer. When CP is finished with the service for that virtual machine the CPU timer, which has been decremented by the amount of CPU time used, is stored back into VMTTIME. GR 11 is then loaded with a new VMBLOK pointer and the CPU 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 CPU 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 non-interactive) and on the CPU 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 (non-interactive) it is 2 seconds. Each time the user is dispatched, the value in VMTMOUTQ is entered into the CPU timer; whenever the user is interrupted, the decremented CPU timer is stored into VMTMOUTQ prior to being set from the new VMTTIME. When the problem state time slice has been exhausted; a CPU 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 CPU timer simulation is handled for EC mode virtual machines if the value in the virtual CPU timer is less than that in VMTMOUTQ. In this case, the VMBLOK is flagged as "tracking CPU timer" and a CPU 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 CPU 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 CPU timer place the new value into the ECBLOK. Requests to store the new value update the ECBLOK field with the virtual CPU 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 CPU 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 CPU 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 CPU 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 CPU timer contains a large negative value, then a real timer request is scheduled to occur when the virtual machine 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, 3270 remote equipment, 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 "Programming for 3270 Remote Terminals - an Introduction" in this section.

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. 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 DMKVIOEX.

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. DMKVIOEX then branches to handle the request based on the operation requested.

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 DMKVIOEX 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, DMKVIOEX 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, DMKVIOEX passes control to DMKFSPEX for additional processing. When control returns to DMKVIOEX, 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. DMKVIOEX 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.

DMKVIOEX stores the contents of the user's CAW in IOECAW 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 DMKVIOEX. DMKVIOEX 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 DMKVIOEX, which passes control to DMKDSPCH.

Other Privileged I/O Instructions

Other privileged I/O instructions are handled directly by DMKVIOEX. DMKVIOEX 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 Adapter

The virtual channel-to-channel adapter (CTCA) simulates data transfer and control communication between two selector channels, 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 CTCA. Each part of the CTCA is identical and the operation of the unit is completely symmetrical. The CTCA occupies an entire control unit slot on each of the two channels attached. The low-order four bits of the unit address (device address) are ignored completely and are not available for use.

The VM/370 control program support for virtual CTCA includes all status, sense data, and interruption logic necessary to simulate the operation of the real CTCA. Data transfer, command byte exchange, sense data, and status data presentation for the virtual CTCA is accomplished via storage-to-storage operations (MVCL, etc.). No real I/O operations (excluding paging I/O) nor I/O interruptions 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 (DMKVIOEX) 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 is. 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/370 control program to allow virtual machines to use this access method. The particular CCWs that require special handling have the following general format:

```
     0        2        4        6        8
    ┌──────────────────────────────────────┐
A   │      READDATA  C+7  10 bytes         │
    │────────│────────│────────│───────────│
B   │             TIC to E                 │
    │────────│────────│────────│───────────│
C   │        │        │        │           │
    │────────│────────│────────│───────────│
D   │        │        │        │           │
    │────────│────────│────────│───────────│
E   │       SEEK: SEEK head on D           │
    │────────│────────│────────│───────────│
F   │          SEARCH on D+2               │
    └──────────────────────────────────────┘
```

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/370 control program, it usually is in the following format:

```
     0        2        4        6        8
    ┌──────────────────────────────────────┐
1   │      READDATA  C+7  10 bytes         │
    │────────│────────│────────│───────────│
2   │             TIC to 3                 │
    │──────────────────────────────────────│
3   │      SEEK: SEEK head on 6            │
    │────────│────────│────────│───────────│
4   │          SEARCH on D+2               │
    │────────│────────│────────│───────────│
5   │        │      etc.       │           │
    │────────│────────│────────│───────────│
6   │        │        │    ISAM word       │
    └──────────────────────────────────────┘
```

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:

```
    ┌──────────────────────────────────────────┐
7   │ Address of Read     │  Address of TIC    │
    │ at 1                │  at 2              │
    │─────────────────────│────────────────────│
8   │  Unused             │   Unused           │
    │─────────────────────│────────────────────│
9   │        Data area for READ at 1           │
    │──────────────────────────────────────────│
10  │             SEEK HEAD on 9               │
    │──────────────────────────────────────────│
11  │               TIC to 4                   │
    │──────────────────────────────────────────│
12  │        Image of READ CCW at 1            │
    │──────────────────────────────────────────│
13  │        Image of TIC CCW at 2             │
    └──────────────────────────────────────────┘
```

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" 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 that 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 type 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 non zero 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 virtual machine and exit is taken to the 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.

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/370 control program function such as for spooling or paging, the entry point DMKIOSQR is called. If the requestor 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 moveble head DASD devices 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.

Regardless of 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 requestor'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.

<u>Ordered</u> <u>Seek</u> <u>Queuing</u>: Requests to start I/O on system devices are normally handled FIFO. However, requests to moveable 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. DMKIOS 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 SEEKs to virtual cylinder 0
- Channel SEEKs, relocates to the virtual cylinder

The IOBLOK queuing subroutine of DMKIOS recognizes that a request is being queued on a moveable head DASD device 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 a DASD device is recorded in the field IOBCYL. 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, DMKIOS 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 devices, this is the first IOBLOK queued. However, for moveable head DASD devices, the queued requests are dequeued in either ascending or descending order, depending on 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 IOBLOK is found, it is dequeued and started. The direction of motion is indicated by an RDEVFLAG bit and the next request is dequeued in the down direction 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 FIFO queuing.

Dedicated Channel Support: One of the facilities of the VM/370 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 they could be both 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, DMKVIO. 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, VDEVBLOCK, 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 VCCNCTL 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*, GA27-2749, and *General Information - Binary Synchronous Communications*, GA27-3004, is required.

A digest of some of this essential information as it applies to VM/370 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/370 system generation procedures employing TERMINAL and CLUSTER macros. For more details about terminal list generation, see the *VM/370: Planning and System Generation Guide*.
- 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' NO-OP
X'09' POLL
X'23' SET MODE
X'27' ENABLE
X'2F' DISABLE

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

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

Table
: The appropriate location in the table of data-link control characters provided in the module DMKGRF (Example: RVI, EOT, ENQ).

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

List
: The appropriate entry in terminal list (NICBLOKS) associated with the READ or WRITE operation. 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 AREA, TABLE, RESPONSE, and LIST are used, refer to the CCW sequences described in "I/O Program Routines for Bisync Lines and 3270 Remote Devices" in this section.

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) and PCI (program controlled interrupt).

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, DMKRGFIN, 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 the 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 BISYNC 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 control unit hardware recognizes a unique string of characters transmitted on the line from CP. 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'	X'40'	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 (for a write disable and write continuous) 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 (write continue) CCW. The format of the CCW write initial and write continue operation follows.

Write Initial

Operation	Command Code	Address	Flags	TP Op Code	Count
Write an EOT	01	Table	CC, SILI	02	1
Write addressing char.	01	List	CC, SILI	03	LIST
Read Response	02	Response	SILI	05	2

Write Continue

Operation	Command Code	Address	Flags	TP Op Code	Count
Write text	01	Area	CC, SILI	10	variable
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 bisync 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	Table	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 second CCW.

Write Enq

Operation	Command Code	Address	Flags	TP Op Code	Count
Write ENQ	01	Table	CC, SILI	03	1
Read Response	02	Response	SILI	11	2

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, then the list (referred to in the poll CCW) of enabled devices is scanned for any messages that are queued and ready for transmission. A positive acknowledgement (yes, I have a message to transmit) from any of the attached devices causes the next 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, all remote stations respond with negative acknowledgement (no messages queued) or any station queried for a response fails to respond, 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	Table	CC, SILI	02	1
Poll	09	List	CC, SILI	03	LIST
I/O No-operation	03	0	SILI	07	1
Read Text	02	Area	SILI	10	162

After CP receives a message from a remote station, it may reissue the initial read sequence to poll the remaining stations on the list (assuming the list of enabled devices was not exhausted on the first pass of the initial read sequence). In the event that the list was exhausted on either the first or a subsequent initial read sequence, CP starts the poll delay, then allows the poll delay interval to expire before starting another read scan 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	Table	CC, SILI	06	1
Read Text	02	Area	SILI	10	162

Once CP message processing receives an error-free message from a remote station, CP sends an RVI control character to the remote station before processing the message. The remote station, upon recognition of the RVI character, halts the sending of additional queued data and responds with EOT (instead of the normal ACK0/ACK1 response). The second CCW of the read interruption sequence processes the EOT response from the remote station as shown in the format below.

Read Interruption

Operation	Command Ccde	Address	Flags	TP Op Code	Count
Write RVI	X'01'	Table	CC, SILI	06	2
Read Response	X'02'	Response	SILI	11	2

DATA FORMATS - BISYNC LINES AND REMOTE 3270

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, Order No. GA27-2749.

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'F7') and COPY (X'F7').

Write Data Stream

```
|STX|ESC|CMD|WCC|BSA| Buffer | Orders |SBA|
|   |   |   |   |   |Address | & Text |   |

  1   1   1   1   1     2     variable  1

                      -----------//------
                      | Buffer |     |ETX|
                      |Address |     |   |
                      -----------//------
                          2            1
```

Write Text Messages for the Copy Command

The COPY command is limited to remote terminal display devices and 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 follow:

Copy Data Stream - 3271

```
|STX|ESC|  CMD   |CCC| From   |ETX|
|   |   | X'F7'  |   |Address |   |
```

Copy Data Stream - 3275

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

Read Text and Read Header Message Formats

The following is representative of typical input-to-processor message formats. The format of a multiline read operation follows.

Read Text Data Stream

```
| Index |STX|CU |Dev|AID|Cursor |SBA|Buff    |
| Byte  |   |Adr|Adr|   |Address|   |Address |

   ---//----------------------//---------
   | Text |SBA| Buff | Text |      |ETX  |
   |      |   | Adr  |      |      |     |
   ---//----------------------//---------
```

Error Status Data Stream

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

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

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 | SOH | % | / | STX | Text | ETX  |
| BYTE  |     |   |   |     |      |      |
```

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. NEXT storage, the management of real storage, and the management of auxiliary storage (DASD paging devices).

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 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 it 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 which indicates to the control program those page frames that are to be released. An entire virtual storage is released when a user IPLs 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.

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 CPU 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 highly 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 case may cause CP to use excessive CPU 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 non-pageable page must remain resident in real storage for some finite period of time; thus, the page frame cannot be taken from its current owner to give it to someone else. Pages can be either permanently or temporarily non-pageable, depending on their use.

Temporary loks 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 allccate 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 which 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 one of three lists if its real page frame is available for paging; however, if the page referenced is locked or it is in transit, its entry is not in any list and is not referenced when available page frames are being searched for swap candidates. The lists are known as the freelist, the flushlst, and the userlist, and they represent various levels of page frame availability.

- The freelist 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 freelist. 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 flushlist contains page frames that belong to those virtual machines that have been dropped from an active dispatching queue. The flushlst is the first place that the page frame selection routine looks to find a page to swap out or to assign to the freelist for a virtual machine that requires real storage space.
- The userlist contains the cortable entries for all other pageable pages in the system that belong to active virtual machines.

<u>Requests for Real Storage Page Frames</u>

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 if 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 fifo 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 fifo 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 not, it is placed in the freelist by DMKPTRFT; otherwise, it is scheduled for a swap-out by dequeueing 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.

If the flushlst is exhausted, select must take a page frame from an active user by examining the page frames represented by the entries in the userlist to locate the least recently used user page frame. This list is scanned from top to bottom, and each page frame is tested to see if its hardware referenced bits have been set. If a page frame has been referenced, its bits are reset and it is queued to the end of the userlist. This process is continued until either an unreferenced page frame is found or the list is exhausted. An unreferenced page frame is immediately selected. However, if the list is exhausted, it is rescanned from the top. An unreferenced page frame is always found; in the worst case it is the first one tested on the userlist at initial entry. However, if this occurs, it indicates that the rate of entry to select is too low to permit differentiation between high- and low-usage page frames.

Once a page frame has been selected and page-out is scheduled, control is returned to DMKPTRFR, which then passes control back to DMKPTRAN with the address of the cortable entry that was allocated. In most cases, page-outs are completely overlapped with page-ins. Approximately one half of all page-ins require a corresponding page-out.

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. 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.

After the required page has been read in or the page frame has been set to zero, DMKPTRAN queues the appropriate cortable entry to the end of the userlist, where it eventually is available for page selection. After developing the real storage address that corresponds to the requested virtual address, DMKPTRAN tests to see if the caller has requested that the page be locked. If LOCK is requested, the cortable entry is de-queued from the userlist and is not available for selection. A resident page can also be locked by removing it from the USERLIST. In addition, a LOCK count is maintained in the cortable entry so that when all locks have been satisfied the page frame can again be made available for paging.

Some requests for main storage page frames are handled differently than 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 tc 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) use 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 the Depth of Multiprogramming" under "Dispatcher/Scheduler" describes the use to which the scheduler puts these counts.

__VM/370 Virtual=Real Option__: The VM/370 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 log on 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 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.

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.

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.

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/370 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/370: Planning and System Generation Guide__. The size of the area must be large enough to contain the entire addressing space of whatever virtual machine wishes to occupy that space. A virtual 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 RDEVBLOK 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.
- It must have at least one cylinder of temporary space marked as available in the cylinder allocation block which is located on cylinder 0, head 0, record 3.

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 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 device, provided that one exists on the system and that it has page space available. Preferred paging devices are specified by the installation at system generation time, and generally should be devices on which excessive seek times do not occur. A typical preferred paging device would be the IBM 2305 Fixed Head Storage facility. If the 2305 is assigned as a preferred device, 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 moveable head DASD devices such as the 3330, 3340, 3350 and 2314/2319 Direct Access Storage facilities as preferred paging devices. 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 moveable head preferred paging device with only the lowest usage data files.

If one or more preferred devices are defined on the system, CP allocates all of the page space available space on these before it allocates on any other available owned volumes. Within the class of preferred devices, space is allocated first on the fastest devices, and these are spead out across channels and devices. Allocation on nonpreferred devices is spread out in the same manner. Cylinders for spooling space are not allocated from preferred devices. Allocation on a given device is done from the relative center of the volume outward, a cylinder at a time in a zig-zag 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, 3330, 3340, or 3350) 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 that controls the operation is unstacked to the interruption return address, waitpage, in DMKPAGIO. waitpage then unchains the CPEXBLOKs that are queued to DMKPAGQ, and then 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" in this section, while errors associated with virtual storage paging operations are discussed later in section "Virtual Storage Paging Error Recovery".

DMKPAGIO maintains its own subpool 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. DMKPAGIO also periodically calculates system paging overhead. After 200 pages have been moved (read or written), the elapsed time for the 200 page moves is computed, and the paging rate is calculated in page moves per second. The recent paging load, expressed as the percentage of time that more than one half of the system's pages were idle due to page-wait, is averaged with the previous load and re-projected as the expected load for the next interval.

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 of 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 shut-down 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 LARA) is performed by DMKPRV. 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 all of 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, it sets a bit in the VMBLOK indicating that all of the shadow page tables must be invalidated. Invalidation of all of the tables is required since CP does not know which third-level storage pages map into the second-level page that is being swapped out. The actual invalidation is handled by DMKVATAB, called from DMKDSP when the virtual machine is on the verge of being dispatched.

The other situations which cause shadow table invalidation arise from the simulation of privileged instructions in DMKPRV. 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 must re-copy the virtual segment table into first-level storage and invalidate the entire shadow segment table. 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 also causes the virtual segment tables to be re-copied and all of the shadow page tables to be invalidated because the shadow tables are the logical equivalent of the translation look-aside buffer.

DMKPRV provides virtual interrogation of the reference and change bits in the virtual storage keys, which involve the privileged instructions ISK, SSK, and RRB. 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.

FREE STORAGE MANAGEMENT

DMKFRE is responsible for the management of free storage, and CP uses it to obtain free storage for I/O tasks, CCW strings, various I/O buffers, etc. It is used, in fact, for practically all such applications except real channel, control unit, and device blocks, and the cortable.

Block sizes of 30 doublewords or less, constituting about 99 per cent of all calls for free storage, are grouped into 10 subpool sizes (3 doublewords each), and are handled by LIFO (push-down stack) logic. Blocks of greater than 30 doublewords are strung off a chained list in the classic manner.

When subpools are exhausted, small size blocks are generally obtained from the first larger sized block at the end of available free storage. Large size 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:

<u>Calling DMKFREE for a Subpool</u>

<u>Subpool Available</u>: 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.

<u>Subpool Not Available</u>: 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 no equal or

larger block is found, all the subpool blocks currently not in use are returned to the main free storage chain, and then the free storage chain is again searched for a block large enough to satisfy the call. If there still is no block large enough to satisfy the request, then DMKPTRFR is called to obtain another page frame of storage from the dynamic paging area, 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 30 doublewords, the chained list of free storage is searched for a block of equal or larger size. 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, and the above process is repeated (as necessary) to obtain the needed block.

Calling DMKFRET for a Subpool

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, however, the block was in a page within the dynamic paging area, the block is returned to the regular free storage chain instead.

Calling DMKFRET for a Large Block

If a block larger than 30 doublewords is returned via DMKFRET, it is merged appropriately into the regular free storage chain. Then, unless the block was returned by DMKFRETR (see "Initialization") 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, it is DMKPTRFT returns it to the dynamic paging area for subsequent use.

Free Storage Page Frame Allocation

The number of page frames allocated to free storage depends upon the number of storage boxes upon which the VM/370 control program is running, and is initialized by DMKCPINT (3 pages for the first 256K and 1 page for each 64K thereafter not including V=R size if any). DMKFRETR is called by DMKCPINT to merge available blocks of storage into the regular free storage chain regardless of their size.

CP INITIALIZATION

System initialization starts when the operator selects the DASD device address of CP system's residence volume (SYSRES) and presses the IPL button. The System/370 hardware reads 24 bytes from record 1 of cylinder 0 on SYSRES into location 0 of main storage. This record consists of an initial PSW and a channel program. The channel program reads the module DMKCKP into location X'800' and gives it control. DMKCKP checks location CPID in module DMKPSA.

If CPID contains the value CPCP or WARM, DMKCKP saves the spool file control blocks, system log messages, accounting information, status of spool devices, spool hold queue blocks, and spool record allocation blocks and writes them on the warm start cylinders. If CPID contains the value CPCP, DMKCKP loads a disabled wait state code X'008'.

If location CPID does not contain the value CPCP, DMKCKP now loads DMKSAV and passes control to it at entry point DMKSAVRS. DMKSAV reloads a page image copy of the CP nucleus into real storage starting at page 0. When DMKSAV is finished, control is transferred to DMKCPI. DMKCPI performs the main initialization function. This includes calling DMKWRM to retrieve the information stored on the warm start cylinder. This also includes calling DMKCKS to initialize the dynamic checkpoint cylinders and to checkpoint the current status of the spool file system. When DMKCPL has finished, it passes control to DMKDSPCH. DMKDSPCH loads a wait state PSW to wait for work.

INITIALIZATION AND TERMINATION

Attaching a Virtual Machine to the System

After CP has been initialized, DMKCPVEN enables the communication lines. Then an individual virtual machine is attached to the system using the following steps:

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 typed 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 and translate tables for input and output are set; if not, flags are set to indicate 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:

- Ensures that the maximum number of virtual machines allowed on the system is not being exceeded.
- Obtains the userid from the command line, and checks for a possible password and other optional operands.
- Checks the userid and password (entered separately if not on the LOGON command line) 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 the 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).
- 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 devices included in the directory, the accessibility of any disk being determined by the user access mode in the directory, and whether any other user(s) 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 places 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.

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 DMKVDB and DEFINE 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 queueing. 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 read/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 different virtual address than its real address; however, the virtual machine may not already have a virtual device at the attached address. A real device cannot be attached (1) if it is currently dedicated to another virtual machine, (2) if it contains mini-disks that are in use by other virtual machines, or (3) if it is a system owned volume that is in use for spooling or paging.

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 adapters, 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.

The DEFINE command 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. Also, any channel with a channel-to-channel adapter (CTCA) defaults to selector mode of operation regardless of the channel mode selected. 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 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.

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 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 informed that the operator has logged off his machine. A virtual machine may also be logged off the system:

- If the time for a read of a system password expires (28 seconds).
- If the user makes a connection to the system but does not logon within a given period.
- 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. DMKDSP calls DMKUSOFF directly by DMKDSP to force the logoff of a disconnected user as previously described.

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 his disconnected machine is still running he 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 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. DMKQCNRD 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. The entered command is matched against a list of valid commands. The list contains a 16-byte entry for each command. Each entry contains 8 bytes for the name, 2 bytes for class mask, 2 bytes for an abbreviation count, and 4 bytes containing the routine address. If the entered command matches an entry in the list, it is then checked to ensure that a valid abbreviation for the command has been used. If this test is not successful, DMKSCN continues to scan the list for a valid command. Should the abbreviation be valid, a check is then made to determine if this user is of the proper class to use the command entered. If this is successful, DMKCFM then calls the appropriate routine to process the command.

After the command has been processed, control is returned to DMKCFM. There are three possible returns. (1) 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 DMKQCNRD to read the next command from the terminal. (2) 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. (3) On a return plus 8, the operation is the same as plus 4 except 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 non-interactive virtual machine, 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 CPU 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.

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.

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, not in dispatch list)
3	Interactive and eligible for queue1, but queue1 is full (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, not in dispatch list)
7	Non-interactive and eligible for queue2, but queue2 is full (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 and 5, and represent the virtual machines that can be run at any given time. The dispatch list is sorted by dispatching priority, which is the ratio of CPU time to wait time over the length of the current virtual machine task. 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. Virtual machines entering state 1 always have a priority of 0.

The eligible list are 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 CPU. 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 (non-interactive). Each section of the list is sorted by scheduling priority, which is determined at the time the virtual machine is added to the eligible list, as follows:

1. The virtual machine's projected working set size, calculated the last time it was dropped from a queue, is expressed as a percentage of the amount of main storage available for paging. This percentage, usually between 0 and 100, is multiplied by the paging bias factor (stored at DMKSCHPB).

2. The virtual machine's priority (the priority set by the directory or the class A SET PRIORITY command) is multiplied by the user bias factor (stored at DMKSCHUB), and is added to the paging bias calculated in step 1.

3. The sum of paging and user bias is divided by the sum of the bias factors to obtain a weighted average.

4. A base priority is obtained by storing the TOD clock and using the high order word, which increments by 1 approximately once per second. This word is then modified by shifting it left or right based on the priority delay factor (stored at DMKSCHPD). If DMKSCHPD is positive, it indicates a right shift, thereby increasing the delay interval of the base priority. A negative value indicates a left shift.

5. The weighted average obtained in step 3 is then logically added to the adjusted base obtained in step 4.

6. If the virtual machine is entering Q2 for the first time after being dropped from Q1, the interactive bias factor (stored at DMKSCHIB) is subtracted from the priority obtained in step 5. If the virtual machine is entering Q1, or if it was last dropped from Q2, the interactive bias is not applied.

7. The result of steps 1 through 6 is the scheduling or eligible list priority, and is stored in the VMEPRIOR field of the VMBLOK.

The VMBLOK is then sorted into the appropriate section of the eligible list in ascending value of VMEPRIOR. The effects of the various biases and the delay factor are illustrated by the following examples.

<u>Example 1</u>
Assume that two virtual machines are to be added to the eligible list for Q2. The paging bias factor is 1, the user bias factor is 1, and the priority delay factor is 0. Virtual machine A has a projected working set size of 80 percent of available storage and a user priority of 50. Virtual machine B has a projected working set size of 20 percent of available storage and also has a user priority of 50. The biases are obtained as follows:

User	Paging Bias		User Bias		Weighted Bias
A	80 X 1	+	50 X 1	=	130/2 = 65
B	20 X 1	+	50 X 1	=	70/2 = 35

If A is added to the eligible list at base time 0, its eligible list priority witll be 65. If the priority delay factor is 0, B is added ahead of A provided that B is eligible for entry to the list within the next (65-35) 30 seconds. If the priority delay factor is set to +1, the base is incremented once every two seconds. Therefore, although the bias difference is still 30, the delay time is now 60 seconds.

Example 2
To force A to be given a weighted bias equal to B, a priority differential is calculated as follows:

$$\frac{80 + A}{2} = \frac{20 + B}{2}$$

$$A = B - 60$$

Therefore, for the biases to be equal, A must have a priority of 60 less than B. For example, if A is given a priority of 10 and B is given a priority of 70, the biases would compute as follows:

User	Paging Bias		User Bias		Weighted Bias
A	80 X 1	+	10 X 1	= 90/2	= 45
B	20 X 1	+	70 X 1	= 90/2	= 45

Example 3
The large difference in priorities could be lessened by increasing the user bias factor. If the user bias factor is set to 3 instead of 1, the calculated priority differential is as follows:

$$\frac{80 + 3A}{4} = \frac{20 + 3B}{4}$$

$$3(B - A) = 60$$

$$A = B - 20$$

Now, A requires a priority of only 20 less then B to achieve parity. For example:

User	Paging Bias		User Bias		Weighted Bias
A	80 X 1	+	30 X 3	= 170/4	= 42
B	20 X 1	+	50 X 3	= 170/4	= 42

The above examples illustrate the following general points about the use of the bias factors, the delay factor, and the user priority value:

1. The paging and user bias factors are a measure of the relative importance of the bias value. A high user bias allows greater discrimination via the assigned priority; while a high paging bias makes storage requirement the primary scheduling parameter.

2. The virtual machine priority value, in the directory, may be overridden, and is the means through which selected users obtain improved performance.

3. The priority delay factor is the measure of the impact that the paging and user biases are to have. The greater the delay value, the greater is the maximum delay that can be experienced by a given user.

4. The interactive bias factor is a tool that enhances command response to conversational commands that require disk I/O, and that may be partially executed in Q2.

If the paging bias factor is nonzero, the net effect of the priority scheme is to discriminate against virtual machines that require large amounts of real storage. This discrimination results in a higher level of multiprogramming and increased CPU utilization; however, it must be traded off against poorer throughput for large storage users. The distributed scheduler is not biased; the bias factors are as follows:

Paging bias factor	(DMKSCHPB)	= 0
User bias factor	(DMKSCHUB)	= 1
Priority delay factor	(DMKSCHPD)	= 0
Interactive bias factor	(DMKSCHIB)	= 0

Thus, the basic VM/370 scheduler schedules virtual machines FIFO within user priority.

Figure 17 is a graphic breakdown of the user states, showing the relationship between interactive and non-interactive states, in-queue and not-in-queue states, and in-list and not-in-list states.

	In-Queue		Not-in-Queue	
	Dispatch List	No List	Eligible List	No List
Interactive	1	2	3	4
Non-Interactive	5	6	7	8

Figure 17. User Dispatching States

Figure 18 shows the possible user-state changes and the reasons for them; any changes not described are not possible.

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

Status Change		Reason for Status Change
From	To	
1	2	Pagewait, SIO-WAIT, or enabled wait for any busy channel
1	4	Enabled wait for interactive terminal read or write
1	5	Exceeds in-queue time slice
1	7	Same as 1 to 5 except that queue 2 is full
1	8	Wait without active I/O, disabled WAIT or hit ATTN
2	1	Wait condition complete
2	5,7	Wait completes, but in-queue time slice exceeded
3	1	Another user drops from queue1 and now there is room
4	1	Terminal I/O completes while user is waiting
4	3	Terminal I/O completes, but queue1 is full
5	1	Terminal I/O completes while user is active in queue2
5	4	User puts up terminal read or write and enters wait
5	6	Pagewait, SIO-WAIT, or enabled wait for busy channel
5	7	Dropped from queue2 due to in-queue time-slice end
5	8	Wait without active I/O, disabled WAIT, or hit ATTN
6	5	Wait condition completes
7	5	Room is found in queue2
8	5,7	Asynchronous I/O or external interruption or BEGIN

Figure 18. User Status Changes

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. Individual virtual machine's working sets are calculated and projected at queue drop time according to one of the following formulas:

$$P=(A+P)/2$$

If (LP-LA) r (P-A) < 0

-- or --

$$P=A$$

If (LP-LA) r (P-A) $\geq$ 0

The working set is added to the current system load, which consists of the sum of the working sets for all virtual machines currently in a queue. The sum is compared to the system maximum, which is equal to the number of dynamically assignable pages in the system. If the virtual machine's projected working set will not push the system load over tthe virtual machine maximum, he is placed in the queue and added to the dispatchable list.

__where__:

A Actual working set at queue drop time

LA Last actual working set

LP Last projected working set

P Current projected working set

The actual working set, A, is the smaller of the two values determimined at queue drop time by the following formula:

$$A = \left[\begin{array}{c} \sum_{i=1}^{N} PRi \;/\; N + \text{Steals} \\ \text{-- or --} \\ \text{Pages referenced} \end{array} \right]$$

__where__:

N Number of page reads while in queue.

PR Number of pages resident at the _ith_ page read.

Steals Number of times page wait was entered because of a stolen page.

The number of referenced pages is determined by scanning the virtual machine's page tables for software referenced bits. These bits are set by DMKPTRAN when the page is taken from the virtual machine by CP. Thus the actual working set is generally the average number of pages resident at each page read. However, this estimate is sensitive to the overall system paging activity for the following reasons:

1. If there is no paging load on the system, there is one page read for each resident page, and no steals; the working set therefore tends to be equal to about one half of the resident page total.

2. As paging activity increases, and the working set location shifts, the working set tends to increase toward the average number of resident pages.

3. If paging activity becomes excessive, the number of page steals increases to the extent that the working set expands to the maximum of the total number of pages referenced while in the queue.

In summary, the scheduler selects the subset of logged-on virtual machines that are allowed to compete for the resources of the CPU, with the constraint that a new virtual machine is not added to the active subset if its projected main storage requirement, added to that of the other active virtual machines, causes the current capacity of the system to be exceeded. Selection within scheduling priority simply means that a executable virtual machine of high priority is always added to the active subset (to a queue) before a executable virtual machine of lower priority. If the paging bias mechanism is activated by setting the paging bias factor to a nonzero value, scheduler selection is in favor of smaller virtual machines; otherwise, selection is within priority. Once the active subset (the set of in-queue virtual machines) has been selected, the dispatcher allocates resources of the CPU among them.

The list of executable virtual machines in a queue is sorted by dispatching (as opposed to scheduling) priority. The dispatching priority is a running average of a given virtual machine's CPU time/wait-time ratio. Thus, virtual machines who are most likely to go into wait state, based on past performance, are dispatched ahead of those whose demands on the CPU are more extensive. This simple ratio priority is normally altered if a virtual machine is identified as compute bound by means of the fact that it has executed for at least 50 ms. without entering the wait state. In this case, it is placed at the bottom of the dispatchable list. On the other hand, virtual machines identified as interactive by virtue of the frequency their requests for terminal I/O are placed at the top of the dispatchable list.

DMKDSP also provides a fast dispatch path for virtual machines that have issued specific privileged instructions that are not handled by the Virtual Machine Assist feature.

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.

Favored Execution Options

When the resources of the CPU (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 its resources to a given virtual machine than would ordinarily be the case. The options provided are:

1. The favored execution option.
2. 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 sytem'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 CPU with many other tasks of the same type, an installation can define the CPU allocation to be made. In this case, the favored execution percentage option can be selected for one virtual machine. This option specifies that the selected virtual machine, in addition to remaining in queue, receives a given minimum percentage of the total CPU time, if he can use it. The percentage is assured in the following manner:

1. The in-queue time slice is multiplied by the requested percentage and added to the virtual machine's current total CPU time usage.
2. When the favored virtual machine, is executable, it is always placed at the top of the dispatchable list until it has obtained his guarantee.
3. If the virtual machine obtains its guarantee before the interval has elapsed, it is placed in the dispatchable list according to its caluculated dispatching priority.
4. In any case, at the end of the in-queue time slice, the guarantee is recomputed as in step 1 and the process repeated.

These options can impact the response time of interactive virtual machines and only one favored percentage virtual machine is allowed at any given time.

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 (DMKSTKCP). 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 PSW's, CSW's 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. The first two doublewords of each buffer contain linkage information described below, followed by the data and CCWs.

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. 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/370 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.

The format of the tag record is a NOP CCW, followed by a TIC to the next CCW and a 136 byte data field. 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 transfered 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 SFBLCK also contains information such as the starting and ending buffer addresses for the file, the record size, certain file status flags, 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/370 (the storage specified at system generation or actual real storage, whichever is less). Allocation is as follows:

Storage Size Available	Virtual Buffers Allocated
256K to 655,360 bytes	128
655,361 bytes to 1.1 megabytes	320
over 1.1 megabytes	640

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 placed in a wait state until a buffer is freed by another virtual machine closing a file. 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. The location of the spool buffer in real storage is transparent to the virtual spooling executive, because all references to the data therein are accomplished through the DAT feature of the CPU.

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 semi-permanent 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 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).

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 must have the universal character set feature. No checking is done on the spooled printer data. 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 feature described below. For the 3211 printer, forms control buffer (FCB) commands are accepted and simulated by means of a virtual FCB maintained by the executive. 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. When the spooled file is directed to a real 3211 or 1403, the operator is responsible for loading the FCB or mounting the proper carriage tape.

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.

Output File Processing

DMKVSP receives control from the virtual I/O executive, DMKVIO, when the user's machine issues a SIO to a spooled unit record device. DMKVIO does not pass control until it has been determined that the device is available (that is, non-busy and with 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 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 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 XFER is in effect, is queued for input to another virtual machine.

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, READERS, 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/370 is provided as an option for the VM/370 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 at DMKVSPBP 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. The data in the virtual buffer, not yet written out to the spool device 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" under "Virtual Spooling Manager."

REAL SPOOLING MANAGER (DMKRSP)

The real spooling manager operates the real unit record devices that are attached to the system and that are used to transcribe input data into reader spool files and user output spool files onto the real printers and punches. The executive optimizes the use of main storage and the CPU rather than running the system unit record devices at their rated speeds. DASD input files are not double buffered and under periods of peak load, input and output devices tend to run in bursts. However, 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.

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 matches that of the device that was made ready. 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 page of real main storage is then obtained for use as a buffer, and the output separator routine (DMKSEP) is called to print output identifier pages. 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. The real SIO supervisor (DMKIOSQR) is then called to start the channel program, and DMKRSP exits to the dispatcher (DMKDSPCH) to await the interruption.

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.

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 is not found, an exit is taken to DMKDSPCH to wait for another ready interruption.

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 "Error Recovery."

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, an open subroutine 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.

Accounting Card Processing

Various routines in CP accumulate, format, and punch account cards that contain system usage information for certain users. These routines format the information into an 80-column card image preceded by a punch CCW and call DMKACOAQ to queue the card for real output. DMKACOAQ calls DMKACOPU to punch the card on a real punch, if one is available; otherwise, the card is queued in main storage until a punch is free. When a punch finishes processing its last file, a test is made to see if any accounting cards have been queued. If they have, DMKACOPU is called to process them.

In addition to the cards generated by CP to account for a virtual machine's use of system resources, the user may request cards to be punched 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.

To punch an accounting card, the user must 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 punched into the accounting card. DMKHVC validates the instruction operands, builds an account buffer (ACNTBLOK), and DMKACOQU is called to queue the card for real output. For additional information about this user option, see "DIAGNOSE Interface (DMKHVC)" under "Privileged Instructions."

When the user accounting option is being utilized, the user must keep in mind that each additional accounting record requested is occupying real storage space. Degradation of system performance occurs if available storage becomes filled with accounting data.

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 99. 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.

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.

After they are closed and are awaiting output, their class 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.

<u>Virtual Device Spooling Commands</u>

These commands affect the status of a user's virtual spooling devices:

<u>Command</u>	<u>Meaning</u>
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 CLASS, 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 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.

<u>Real Device Spooling Commands</u>

The operator can use these commands to control the activity of the real spooling devices:

<u>Command</u>	<u>Meaning</u>
BACKSPAC	Backspaces an active 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

for the output device and output separator records.

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 99 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 followed by a valid spool class 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 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, and the class of the specified file. Any of the above attributes of a file can be determined via the QUERY command.
HOLD	Places, via the system operator, the specified file in a hold status. The file is not printed or punched 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, or by any combination of the two.
TRANSFER	Transfers a virtual reader to another user's virtual reader without any processing. The TRANSFER command causes a changing in the owning userid field in the file's SFBLOK.

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 executive 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.

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. When the systme 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 had 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) 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

DMKCPI calls 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/370 is running in a virtual machine environment, or running standalone on the real machine. If VM/370 is running in a virtual machine, the version code is set to a hexadecimal 'FF' by DMKPRV. If the version code returned is hexadecimal '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 and channel errors (other than channel data checks) are not reflected to any virtual machine. VM/370, 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 and the CP executing block (CPEXBLOK). DMKIOG saves the pointers for the machine check record and the CPEXBLOK in DMKMCH. DMKIOG obtains the storage for the I/O extended logout area and initializes the logout area and the ECSW to 1s. 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 CPU to produce an extended logout or I/O extended logout and the length of the logouts are both model and channel dependent. If VM/370 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/370 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 MCH/CCH recording cylinder for error recording is initialized. DMKIOE passes control back to DMKCPI 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 CPU, real storage or control storage. When any of these fails to work properly, the CPU attempts to correct the malfunction.

When the malfunction is corrected, the machine check handler (MCH) is notified by a machine check interruption and the CPU 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 four categories: functional recovery, system recovery, system-supported 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 CPU retry, the ECC facility, or the machine check handler. CPU 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 a system-supported restart.

SYSTEM-SUPPORTED RESTART: When the machine check occurs in a critical routine, the primary system operator is notified that the system cannot continue to operate. An automatic reload of the system occurs. This type of recovery is tried when functional and system recovery have failed or could not be tried.

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. An example of this type of error is when a hard error occurs so frequently that system-supported restart is not successful.

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.

CPU Retry

CPU 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 CPU 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 CPU. 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 19). 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	Synch. 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	Asynch. MCEL control	Mch-Chk handling
14	9	Asynch. fixed log control	Mch-Chk handling
15	8-28	MCEL address	Mch-Chk handling

Figure 19. RMS Control Register Assignments

Machine Check Handler Subroutines

VM/370 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 is shutdown followed by an automatic restart function.
- 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 CPU was in supervisor state the error is considered uncorrectable and the VM/370 system is terminated. If the CPU was in problem state, the virtual machine is reset or terminated and the system continues operation.
- If CPU 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/370 system to be terminated. An automatic restart reinitializes VM/370.

If the damage is not in the CP nucleus, main storage is exercised to determine if the failure is solid or intermittent. 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/370 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/370 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 interrruption 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/370 is a potentially catastrophic failure. Namely, VM/370 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. The system is automatically restarted and VM/370 is reinitialized. 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. (1) The virtual machine is selectively reset or terminated by the virtual machine termination subroutine; (2) 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 (DMKMCHMS) allows the service representative to change the mode that CPU retry and ECC recording are operating in. This subroutine receives control when a user with privilege class F issues some form of the SET MODE command. A check is initially made to determine if this is VM/370 running under VM/370. If this is the case, the request is ignored and control is returned to the calling routine. The format of the MODE command is as follows:

```
SET MODE {RETRY|MAIN} {QUIET|RECORD}
```

RETRY and MAIN imply CPU retry and main storage, respectively.

QUIET causes the specified facility to be placed in quiet mode. RECORD causes the count of soft errors to be reset to zero and the specified facility to be placed in record mode.

Operator Communication Subroutine

The operator communciation 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 CPU in a disabled wait state with a recognizable wait state code in the CPU 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 CPU 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 CPU retry and corrected ECC events.

CPU Retry Recording Mode: Recording mode (bit 4 of control register 14 set to one) is the initialized state, and normal operating state of VM/370 for CPU 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 CPU 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/370 error recording cylinder, but the primary systems operator is not informed of these occurrences.

CPU Retry Quiet Mode: Quiet mode (bit 4 of control register 14 set to 0) can be entered in one of two ways: (1) when 12 soft machine checks have occurred, or (2) when the SET MODE RETRY QUIET command is executed by a class F user. In this mode, both CPU retry and ECC reporting are disabled. The CPU 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.

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 CPU model. For the IBM System/370 Models 135, 145, 158, and 168, the hardware initialized state (therefore the normal operational state for VM/370) is quiet mode. For the IBM System/370 Models 155 II and 165 II, the hardware initialized state (the normal operational state for VM/370) is record mode. An automatic restart incident due to a VM/370 failure does not reset the ECC recording mode in effect at the time of failure.

The change from record to quiet mode for ECC recording can be initiated in either of the following ways: (1) by issuing the SET MODE {MAIN|RETRY} QUIET command, or (2) 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 CPU 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 CPU models equipped with a high speed buffer (155 II, 158, 165 II, 168) or a data lookaside table (DLAT) (165 II, 168) 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 CPEREP programs. A message about the channel error is issued 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, 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, CMKCCH 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 CPU in a disabled wait state with a recognizable wait code in the CPU instruction counter.

Recovery is not initiated for channel errors associated with I/O events initiated by a virtual machine, however these causes termination of the virtual machine after it has been notified of the failure. The error is recorded by DMKIOECC on the error recording cylinder.

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.

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:

1. The device dependent error recording program (ERP) 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.
2. DMKIOECC uses the IOERBLOK to record the channel check record on the error recording cylinder.

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 (DMKCCH603W) is sent to him advising him of the channel error. A LPSW is then executed to place the CPU in a disabled wait state with a wait state code of 002 in the CPU 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, DMKIOECC is called to record the channel check record on the error recording cylinder. The system operator is then sent a message (DMKCCH601I or DMKCCH602I) informing him of the error and control is then returned to DMKIOS to continue system operation.

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 IOERBLCK. 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, 145, 155 II, 158)

Since all of these systems have integrated channels one common subroutine is used to handle all of these CPU types. This subroutine:

- Indicates CP termination if the ECSW is not complete, the channel has been reset, or reset codes are invalid
- 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 1s
- 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, set 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 logout completely, at least part of the logout area is all 1s. 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 9 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 zero to five 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/370. If the virtual operating system is VM/370, it must be the Release 2.0 version or later. An SVC 76 issued by a virtual machine is used to signal VM/370 that error recording is required. The SVC interruption handler in DMKPSA 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/370. If the record is valid, it is modified by changing virtual information to real. The actual recording is accomplished by using existing modules in 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/370 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 four modules. One module (DMKIOE) is resident and the other three (DMKIOC, DMKIOF, and DMKIOG) are pageable.

The error recording modules record temporary errors (statistical data recording) for CP generated I/O except for DASD devices 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 program. 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 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 cylinder using DMKRPAGT and DMKRPAPT. The error recording cylinders have page sized records (4096 bytes). Each page contains a header (8 bytes) which signifies: the cylinder and page number of the page (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.

Two cylinders are used for error recording: one cylinder is used exclusively for recording the I/O errors and the other cylinder for recording MCH/CCH errors. The cylinders that are used for error recording are specified by the user at system generation time. If either error recording cylinder becomes 90 per cent full, a message is issued to the operator using DMKQCNWT to warn him of the condition. If either cylinder becomes full, another message is issued to inform the operator and recording is stopped on that cylinder. Recording continues on the cylinder that is not full.

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 cylinder 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 the CPEREP program via a DIAGNOSE instruction. DMKIOEFM is invoked to reset the specified error recording cylinders (if CLEARALL, CLEARIO, or CLEARMC was specified). The clear is performed by resetting each page-header, space-available field. A pointer in storage is then updated to point to the first page on the error recording cylinder available for recording MCH and CCH records and the first page available on the other error recording cylinder for recording outboard errors. Control is then returned to the calling routine.

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 cylinder's 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 either error recording cylinder is in an unrecognizable format, that cylinder is automatically reformatted by CP.

DASD ERROR RECOVERY, ERP (DMKDAS)

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

- 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, 3350, and 2305 devices; twice for the 2314 and 2319.

No record found and missing address marks: Recalibrate and retry the channel program 10 times (2314/2319).

No record found: 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: Retry the operation 10 times except that 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 3330/3350, the operation is retried by hardware.

Overrun: Retry the operation 10 times.

Missing address marker: Retry the operation 10 times.

Command reject: 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: This error should not occur. CP does not use alternate tracks in its paging or spooling management. When a disk pack is formatted, any track that is marginal is marked as permanently allocated and, therefore, made unavailable for use by CP.

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 exeeded 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 fails to correct the error situation, the operator may or may not be notified of the error condition. 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 and DMKIOE determine that the error condition 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 IOBSTAT 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 IOBSTAT 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 self explanatory and are explained in Figure 20.

Field			Action to be
IOBSTAT IOBERP	IOBFLAG IOBRSTRT	IOBSTAT IOBFATAL	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 successfull

Figure 20. 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.

TAPE ERROR RECOVERY, ERP (DMKTAP)

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. 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, DMKTAP 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 by DMKTAP, the temporary error is not recorded and control is returned to DMKIOS with error flags all off. 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 IOBSTAT 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 (IOBSTAT=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 23 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 bisync 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 _IBM 3270 Information Display Component Description_, Order No. GA27-2749) is processed by DMKRGF. 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.

Depending on the error encountered, ERP receives control and either DMKBSC or DMKGRA and DMKGRB 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 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.

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 23 for a summary of the IOB indicators). Also, the statistical counters for 3270 are updated.

THE CONVERSATIONAL MONITOR SYSTEM (CMS)

- Introduction to CMS
- Interruption handling
- Functional information (how CMS works)
  - -- Register usage
  - -- DMSNUC structure
  - -- Storage structure
  - -- Free storage management
  - -- SVC handling
- OS macro simulation

The Conversational Monitor System (CMS), the major subsystem of VM/370, provides a comprehensive set of conversational facilities to the user. Several copies of CMS may run under CP, thus providing several users with their own time-sharing system. CMS is designed specifically for the VM/370 virtual machine environment.

Each copy of CMS supports a single user. This means that the storage area contains only the data pertaining to that user. Likewise, each CMS user has his own machine configuration and his own files. Debugging is simpler because the files and storage area are protected from other users.

Programs can be debugged from the terminal. The terminal is used as a printer to examine limited amounts of data. After examining program data, the terminal user can enter commands on the terminal to alter the program.

CMS, operating with CP, is a time-sharing system suitable for problem solving, program development, and general work. It includes several programming language processors, file manipulation commands, utilities, and debugging aids. Additionally, CMS can simplify the operation of other operating systems in a virtual machine environment when controlled from a remote terminal. For example, CMS can create and modify job streams, and to analyze virtual printer output.

Part of the CMS environment is related to the virtual machine environment created by CP. Each user is completely isolated from the activities of all other users, and each machine in which CMS executes has virtual storage available to it and managed for it. The CP commands are recognized by CMS. For example, the commands allow messages to be sent to the operator or to other users, and virtual devices to be dynamically detached from the virtual machine configuration.

THE CMS COMMAND LANGUAGE

The CMS command language offers terminal users a wide range of functions. It supports a variety of programming languages, service functions, file manipulation, program execution control, and general system control. The CMS commands that are useful in debugging are discussed in "Debugging with CMS" in this section. For detailed information on all other CMS commands, refer to the _VM/370: CMS Command and Macro Reference_. Figure 23 describes CMS command processing.

THE FILE SYSTEM

CMS interfaces with virtual disks, tapes, and unit record equipment. The CMS residence device is kept as a read-only, shared, system disk. Permanent user files may be accessed from up to nine active disks. CMS controls logical access to those virtual disks, while CP facilities manage the device sharing and virtual-to-real mapping.

User files in CMS are identified with three designators: the filename, the filetype, which can imply specific file characteristics to the CMS file management routines, and the filemode, which describes the location and access mode of the file.

The compilers available under CMS default to particular input filetypes, such as ASSEMBLE, but the file manipulation and listing commands do not. Files of a particular filetype form a logical data library for a user; for example, the collection of all COBOL source files, or of all object (TEXT) decks, or of all EXEC procedures. This allows selective handling of specific groups of files with minimum input by the user.

User files can be created directly from the terminal with the CMS EDIT facility. EDIT provides extensive context editing services. File characteristics such as record length and format, tab locations, and serialization options can be specified. The system includes standard definitions for certain filetypes.

CMS automatically allocates compiler work files at the beginning of command execution on whichever active disk has the greatest amount of available space, and deallocates them at completion. Compiler object decks and listing files are normally allocated on the same disk as the input source file or on the primary read/write disk, and are identified by combining the input filename with the filetypes TEXT and LISTING. These disk locations may be overridden by the user.

A single user file is limited to a maximum of 65533 records and must reside on one virtual disk. The file management system limits the number of files on any one virtual disk to 3400. All CMS disk files are written as 800-byte records, chained together by a specific file entry that is stored in a table called the master file directory; a separate master file directory is kept for, and on, each virtual disk. The data records may be discontiguous, and are allocated and deallocated automatically. A subset of the master file directory (called the user file directory) is made resident in virtual storage when the disk directory is made available to CMS; it is updated on the virtual disk at least once per command if the status of any file on that disk has been changed.

Virtual disks may be shared by CMS users; the facility is provided by VM/370 to all virtual machines, although a user interface is directly available in CMS commands. Specific files may be spooled between virtual machines to accomplish file transfer between users. Commands allow such file manipulations as writing from an entire disk or from a specific disk file to a tape, printer, punch, or the terminal. Other commands write from a tape or virtual card reader to disk, rename files, copy files, and erase files. Special macro libraries and text or program libraries are provided by CMS, and special commands are provided to update and use them. CMS files can be written onto and restored from unlabeled tapes via CMS commands.

Caution: Multiple write access under CMS can produce unpredictable results.

Problem programs which execute in CMS can create files on unlabeled tape in any record and block size; the record format can be fixed, variable, or undefined. Figure 21 describes the CMS file system.

PROGRAM DEVELOPMENT

CMS includes commands to create and compile source programs, to modify and correct source programs, to build test files, to execute test programs and to debug from the terminal. The commands of CMS are especially useful for OS and DOS/VS program development, but also may be used in combination with other operating systems to provide a virtual machine program development tool.

CMS utilizes the OS and DOS/VS compilers via interface modules; the compilers themselves normally are not changed. To provide suitable interfaces, CMS includes a certain degree of OS and DOS/VS simulation. The sequential, direct, and partitioned access methods are logically simulated; the data records are physically kept in the chained 800-byte blocks which are standard to CMS, and are processed internally to simulate OS data set characteristics. CMS supports VSAM catalogs, data spaces, and files on OS and DOS disks using the DOS/VS Access Method Services. OS SVC functions such as GETMAIN/FREEMAIN and TIME are simulated. The simulation restrictions concerning what types of OS object programs can be executed under CMS are primarily related to the OS/PCP, MFT, and MVT Indexed Sequential Access Method (ISAM) and the telecommunications access methods, while functions related to multitasking in OS and DOS/VS are ignored by CMS.

INTERRUPTION HANDLING IN CMS

CMS receives virtual SVC, input/output, program, machine, and external interruptions and passes control to the appropriate handling program.

SVC Interruptions

The Conversational Monitor System is SVC (supervisor call) driven. SVC interruptions are handled by the DMSITS resident routines. Two types of SVCs are processed by DMSITS: internal linkage SVC 202 and 203, and any other SVCs. The internal linkage SVC is issued by the command and function programs of the system when they require the services of other CMS programs. (Commands entered by the user from the terminal are converted to the internal linkage SVC by DMSINT). The OS SVCs are issued by the processing programs (for example, the Assembler).

INTERNAL LINKAGE SVCS: When DMSITS receives control as a result of an internal linkage SVC (202 or 203), it saves the contents of the general registers, floating-point registers, and the SVC old PSW, establishes the normal and error return addresses, and passes control to the specified routine. (The routine is specified by the first 8 bytes of the parameter list whose address is passed in register 1 for SVC 202, or by a halfword code following SVC 203.)

For SVC 202, if the called program is not found in the internal function table of nucleus (resident) routines, then DMSITS attempts to call in a module (a CMS file with filetype MODULE) of this name via the LOADMOD command.

If the program was not found in the function table, nor was a module successfully loaded, DMSITS returns an error indicator code to the caller.

To return from the called program, DMSITS restores the calling program's registers, and makes the appropriate normal or error return as defined by the calling program.

OTHER SVCS: The general approach taken by DMSITS to process other SVCs supported under CMS is essentially the same as that taken for the internal linkage SVCs. However, rather than passing control to a command or function program, as is the case with the internal linkage SVC, DMSITS passes control to the appropriate routine. The SVC number determines the appropriate routine.

In handling non-CMS SVC calls, DMSITS refers first to a user-defined SVC table (if any -- set up by the DMSHDS program).

If the user-defined SVC table is present, any SVC number (other than 202 or 203) is looked for in that table. If it is found, control is transferred to the routine at the specified address.

If the SVC number is not found in the user-defined SVC table (or if the table is nonexistent), the standard system table of OS calls is searched for that SVC number. If the SVC number is found, control is transferred to the corresponding address in the usual manner. If the SVC number is not in either table, then the supervisor call is treated as an ABEND call.

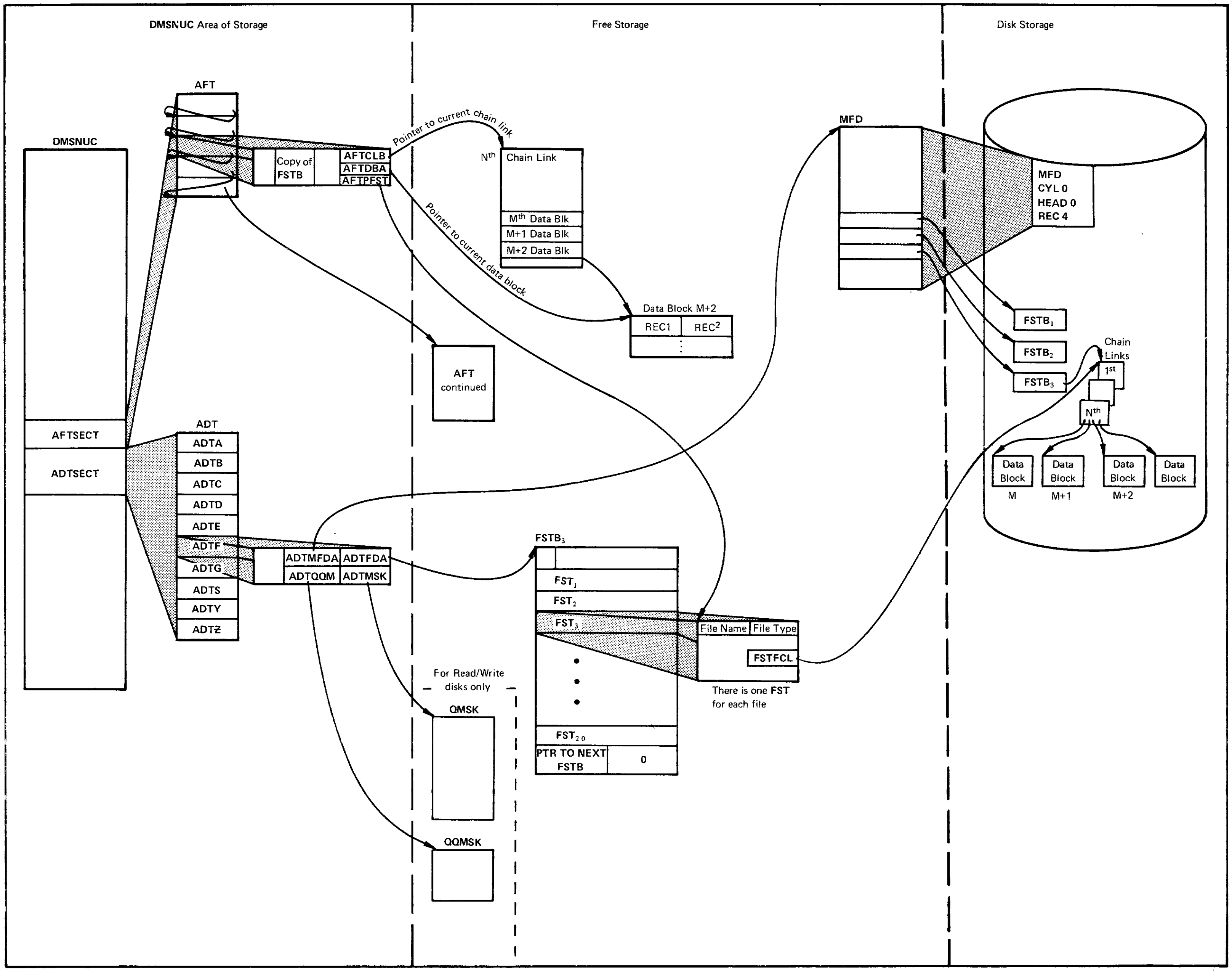


Figure 21. CMS File System

The DMSHDS initialization program sets up the user-defined SVC table. It is possible for a user to provide his own SVC routines.

Input/Output Interruptions

All input/output interruptions are received by the I/O interrupt handler, DMSITI. DMSITI saves the I/O old PSW and the CSW (channel status word). It then determines the status and requirements of the device causing the interruption and passes control to the routine that processes interruptions from that device. DMSITI scans the entries in the device table until it finds the one containing the device address that is the same as that of the interrupting device. The device table (DEVTAB) contains an entry for each device in the system. Each entry for a particular device contains, among other things, the address of the program that processes interruptions from that device.

When the appropriate interrupt handling routine completes its processing, it returns control to DMSITI. At this point, DMSITI tests the wait bit in the saved I/O old PSW. If this bit is off, the interruption was probably caused by a terminal (asynchronous) I/O operation. DMSITI then returns control to the interrupted program by loading the I/O old PSW.

If the wait bit is on, the interruption was probably caused by a nonterminal (synchronous) I/O operation. The program that initiated the operation most likely called the DMSIOW function routine to wait for a particular type of interruption (usually a device end.) In this case, DMSITI checks the pseudo-wait bit in the device table entry for the interrupting device. If this bit is off, the system is waiting for some event other than the interruption from the interrupting device; DMSITI returns to the wait state by loading the saved I/O old PSW. (This PSW has the wait bit on.)

If the pseudo-wait bit is on, the system is waiting for an interruption from that particular device. If this interruption is not the one being waited for, DMSITI loads the saved I/O old PSW. This will again place the machine in the wait state. Thus, the program that is waiting for a particular interruption will be kept waiting until that interruption occurs.

If the interruption is the one being waited for, DMSITI resets both the pseudo-wait bit in the device table entry and the wait bit in the I/O old PSW. It then loads that PSW. This causes control to be returned to the DMSIOW function routine, which, in turn, returns control to the program that called it to wait for the interruption.

TERMINAL INTERRUPTIONS: Terminal input/output interruptions are handled by the DMSCIT module. All interruptions other than those containing device end, channel end, attention, or unit exception status are ignored. If device end status is present with attention and a write CCW was terminated, its buffer is unstacked. An attention interrupt causes a read to be issued to the terminal, unless attention exits have been queued via the STAX macro. The attention exit with the highest priority is given control at each attention until the queue is exhausted, then a read is issued. Device end status indicates that the last I/O operation has been completed. If the last I/O operation was a write, the line is deleted from the output buffer and the next write, if any, is started. If the last I/O operation was a normal read, the buffer is put on the finished read list and the next operation is started. If the read was caused by an attention interrupt, the line is first checked for the commands RT, HO, HT, or HX, and the appropriate flags are set if one is found. Unit exception indicates a canceled read. The read is reissued, unless it had been issued with ATTREST=NO, in which case unit exception is treated as device end.

READER/PUNCH/PRINTER INTERRUPTIONS: Interruptions from these devices are handled by the routines that actually issue the corresponding I/O operations. When an interruption from any of these devices occurs, control passes to DMSITI. Then DMSITI passes control to DMSIOW, which returns control to the routine that issued the I/O operation. This routine can then analyze the cause of the interruption.

USER CONTROLLED DEVICE INTERRUPTIONS: Interrupts from devices under user control are serviced the same as CMS devices except that DMSIOW and DMSITI manipulate a user created device table, and DMSITI passes control to any user written interrupt processing routine that is specified in the user device table. Otherwise, the processing program regains control directly.

Program Interruptions

The program interruption handler, DMSITP, receives control when a program interruption occurs. When DMSITP gets control, it stores the program old PSW and the contents of the registers 14, 15, 0, 1, and 2 into the program interruption element (PIE). (The routine that handles the SPIE macro instruction has already placed the address of the program interuption control area (PICA) into PIE.) DMSITP then determines whether or not the event that caused the interruption was one of those selected by a SPIE macro instruction. If it was not, DMSITP passes control to the DMSABN ABEND recovery routine.

If the cause of the interruption was one of those selected in a SPIE macro instruction, DMSITP picks up the exit routine address from the PICA and passes control to the exit routine. Upon return from the exit routine, DMSITP returns to the interrupted program by loading the original program check old PSW. The address field of the PSW was modified by a SPIE exit routine in the PIE.

External Interruptions

An external interruption causes control to be passed to the external interrupt handler DMSITE. If the user has issued the HNDEXT macro to trap external interrupts, DMSITE passes control to the user's exit routine. If the interrupt was caused by the timer, DMSITE resets the timer and types the BLIP character at the terminal. The standard BLIP timer setting is two seconds, and the standard BLIP character is upper case, followed by the lower case (it moves the typeball without printing). Otherwise, control is passed to the DEBUG routine.

Machine Check Interruptions

Hard machine check interruptions on the real CPU are not reflected to a CMS virtual user by CP. A message prints on the console indicating the failure. The user is then disabled and must IPL CMS again in order to continue.

FUNCTIONAL INFORMATION

The most important thing to remember about CMS, from a debugging standpoint, is that it is a one-user system. The supervisor manages only one user and keeps track of only one user's file and storage chains. Thus, everything in a dump of a particular machine relates only to that virtual machine's activity.

You should be familiar with register usage, save area structuring, and control block relationships before attempting to debug or alter CMS.

Register Usage

When a CMS routine is called, R1 must point to a valid parameter list (PLIST) for that program. On return, R0 may or may not contain meaningful information (for example, on return from a call to FILEDEF with no change, R0 will contain a negative address if a new FCB has been set up; otherwise, a positive address of the already existing FCB). R15 will contain the return code, if any. The use of Registers 0 and 2 through 11 varies.

On entry to a command or routine called by SVC 202:

Register	Contents
1	The address of the PLIST supplied by the caller
12	The address entry point of the called routine
13	The address of a work area (12 doublewords) supplied by SVCINT
14	The return address to the SVCINT routine
15	The entry point (same as register 12)

On return from a routine, register 15 contains:

Return Code	Meaning
0	No error occurred
<0	Called routine not found
>0	Error occurred

If a CMS routine is called by an SVC 202, registers 0 through 14 are saved and restored by CMS.

Most CMS routines use register 12 as a base register.

Structure of DMSNUC

DMSNUC is the portion of storage in a CMS virtual machine that contains system control blocks, flags, constants, and pointers.

The CSECTs in DMSNUC contain only symbolic references. This means that an update or modification to CMS, which changes a CSECT in DMSNUC, does not automatically force all CMS modules to be recompiled. Only those modules that refer to the area that was redefined must be recompiled.

USERSECT (User Area)

The USERSECT CSECT defines space that is not used by CMS. A modification or update to CMS can use the 18 fullwords defined for USERSECT. There is a pointer (AUSER) in the NUCON area to the user space.

DEVTAB (Device Table)

The DEVTAB CSECT is a table describing the devices available for the CMS system. The table contains the following entries:

- 1 console
- 10 disks
- 1 reader
- 1 punch
- 1 printer
- 4 tapes

You can change some existing entries in DEVTAB. Each device table entry contains the following information:

- Virtual device address
- Device flags
- Device types
- Symbol device name
- Address of the interrupt processing routine (for the console)

The virtual address of the console is defined at IPL time. The virtual address of the user disks can be altered dynamically with the ACCESS command. The virtual address of the tapes can be altered in the device table. Changing the virtual address of the reader, printer, or punch will have no effect.

STRUCTURE OF CMS STORAGE

Figure 22 describes how CMS uses its virtual storage. The pointers indicated (MAINSTRT, MAINHIGH, FREELOWE, and FREEUPPR) are all found in NUCON (the nucleus constant area).

The sections of CMS storage have the following uses:

- DMSNUC (X'00000' to approximately X'03000')

  This area contains pointers, flags, and other data updated by the various system routines.

- Low Storage DMSFREE Free Storage Area (Approximately X'03000' to X'0E000'

  This area is a free storage area, from which requests from DMSFREE are allocated. The top part of this area contains the File Directory for the System Disk (SSTAT). If there is enough room (as there will be in most cases), the FREETAB table also occupies this area, just below the SSTAT.

- Transient Program Area (X'0E000' to X'10000')

  Because it is not essential to keep all nucleus functions resident in storage all the time, some of them are made "transient." This means that when they are needed, they are loaded from the disk into the Transient Program Area. Such programs may not be longer than two pages, because that is the size of the Transient Area. (A page is 4096 bytes of virtual storage.) All transient routines must be serially reusable since they are not read in each time they are needed.

- CMS Nucleus (X'10000' to X'20000')

  Segment 1 of storage contains the reenterable code for the CMS Nucleus routines. In shared CMS systems, this is the "protected segment." That is, this segment must consist only of reenterable code, and may not be modified under any circumstances. This fact implies certain system restrictions for functions which require that storage be modified, such as the fact that DEBUG breakpoints or CP address stops cannot be placed in this segment, in a saved system.

- User Program Area (X'20000' to Loader Tables)

  User programs are loaded into this area by the LOAD command. Storage allocated by means of the GETMAIN macro instruction is taken from this area, starting from the high address of the user program. In addition, this storage area can be allocated from the top down by DMSFREE, if there is not enough storage available in the low DMSFREE storage area. Thus the usable size of the User Program Area is reduced by the amount of free storage which has been allocated from it by DMSFREE.

- Loader Tables (Top pages of storage)

  The top of storage is occupied by the Loader Tables that are required by the CMS loader. These tables indicate which modules are currently loaded in the User Program Area (and the Transient Program Area after a LOAD COMMAND). The size of the Loader Tables can be varied by the SET LDRTBLS command. However, to successfully change the size of the Loader Tables, the SET LDRTBLS command must be issued immediately after IPL.

FREE STORAGE MANAGEMENT

Free storage can be allocated by the GETMAIN or DMSFREE macros. Storage allocated by the GETMAIN macro is taken from the user program area, beginning after the high-address of the user program.

Storage allocated by the DMSFREE macro can be taken from several areas.

If possible, DMSFREE requests are allocated from the low-address free storage area. Otherwise, DMSFREE requests are satisfied from the storage above the user program area.

There are two types of DMSFREE requests for free storage: requests for USER storage and NUCLEUS storage. Because the two types of storage are kept in separate 4K pages, it is possible for storage of one type to be available in low storage, while no storage of the other type is available.

<u>GETMAIN Free Storage Management</u>

All GETMAIN storage is allocated in the user program area, starting after the end of the user's actual program. Allocation begins at the location pointed to by the NUCON pointer MAINSTRT. The location MAINHIGH in NUCON is the "high-extend" pointer for GETMAIN storage.

Before issuing any GETMAIN macros, user programs must use the STRINIT macro to set up user free storage pointers. The STRINIT macro is issued only once, preceding the initial GETMAIN request. The format of the STRINIT macro is:

```
┌─────────────────────────────────────────────────────┐
|         |           | ┌         ┌  ┐┐                |
| [label] | STRINIT   | |TYPCALL=|SVC ||               |
|         |           | |         |BALR||               |
|         |           | └         └  ┘┘               |
└─────────────────────────────────────────────────────┘
```

<u>where</u>:

```
        ┌    ┐
TYPCALL=|SVC |
        |BALR|
        └    ┘
```

indicates how control is passed to DMSSTG, the routine that processes the STRINIT macro. Because DMSSTG is a

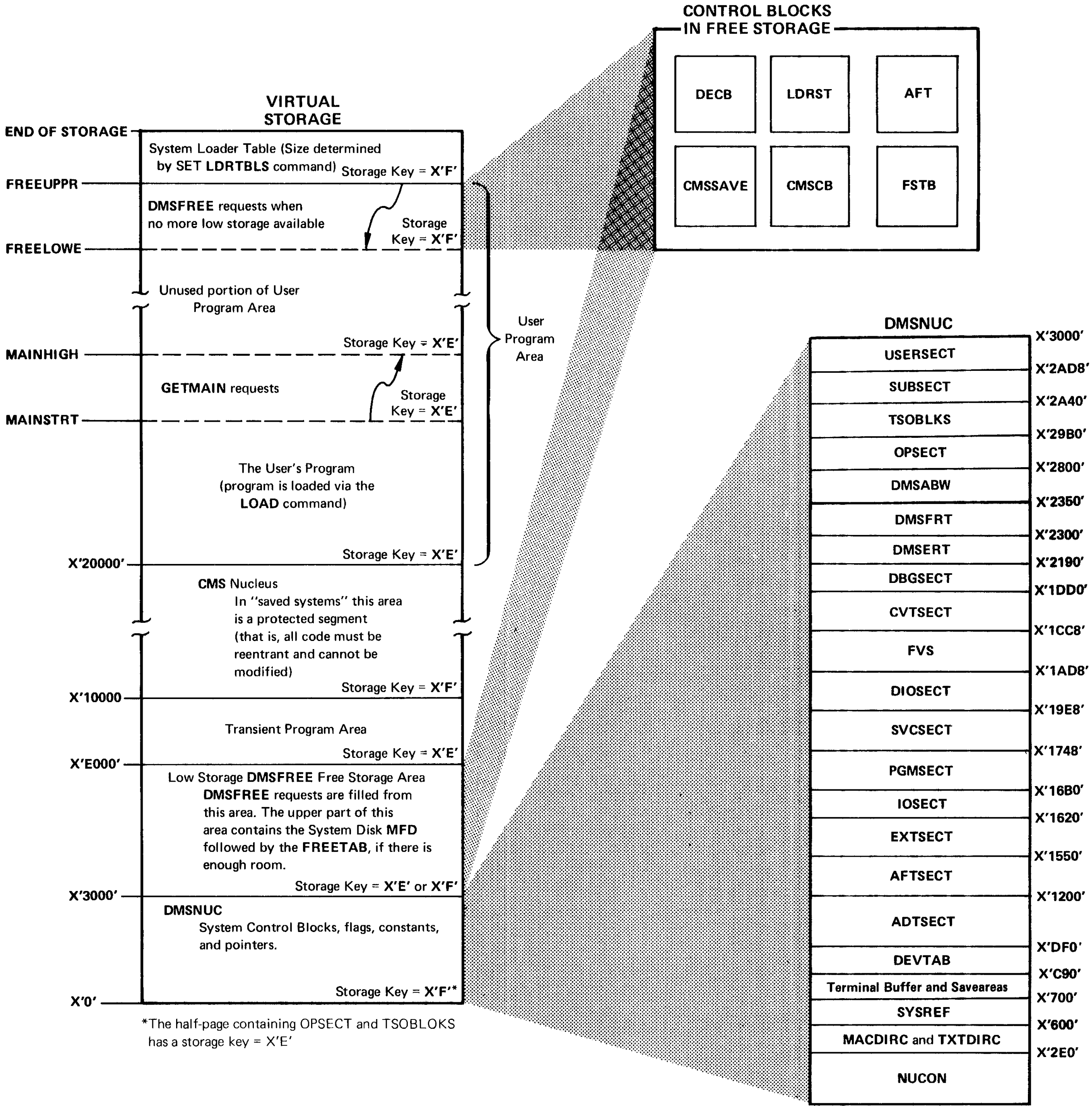


Figure 22. CMS Storage Map

nucleus-resident routine, other nucleus-resident routines can branch directly to it (TYPCALL=BALR) while routines that are not nucleus-resident must use linkage SVC (TYPCALL=SVC). If no operands are specified the default is TYPCALL=SVC.

When the STRINIT macro is executed, both MAINSTRT and MAINHIGH are initialized to the end of the user's program, in the user program area. As storage is allocated from the user program area to satisfy GETMAIN requests, the MAINHIGH pointer is adjusted upward. Such adjustments are always in multiples of doublewords, so that this pointer is always on a doubleword boundary. As the allocated storage is released, the MAINHIGH pointer is adjusted downward.

The pointer MAINHIGH can never be higher than FREELOWE, the "low-extend" pointer for DMSFREE storage allocated in the user program area. If a GETMAIN request cannot be satisfied without extending MAINHIGH above FREELOWE, GETMAIN takes an error exit, indicating that insufficient storage is available to satisfy the request.

The area between MAINSTRT and MAINHIGH may contain blocks of storage that are not allocated, and that are therefore available for allocation by a GETMAIN instruction. These blocks are chained together, with the first one pointed to by the NUCON location MAINSTRT. Refer to Figure 22 for a description of CMS virtual storage usage.

The format of an element on the GETMAIN free element chain is as follows:

```
      ┌────────┬────────┬────────┬────────┐
      | FREPTR -- pointer to next free    |
0(0)  |     element in the chain, or 0    |
      |     if there is no next element   |
      |--------|--------|--------|--------|
      | FRELEN -- length, in bytes, of    |
4(4)  |    this element                   |
      |                                   |
      |--------|--------|--------|--------|
      |  Remainder of this free element   |
      <                                   >
      <                                   >
      <                                   >
```

When issuing a variable length GETMAIN, six and a half pages are reserved for CMS usage; this is a design value. A user who needs additional reserved pages (for example, for larger directories) should free up some of the variable GETMAIN storage from the high end.

DMSFREE Free Storage Management

The DMSFREE macro allocates CMS free storage. The format of the DMSFREE macro is:

```
| [label] | DMSFREE | DWORDS={ n  } [,MIN={ n  }]
|         |         |        { (0)}       { (1)}
|         |         |
|         |         | [,TYPE=[USER   ]] [,ERR=[laddr]]
|         |         |        [NUCLEUS]        [ *   ]
|         |         |
|         |         | [,AREA=[LOW ]] [,TYPCALL=[SVC ]]
|         |         |        [HIGH]           [BALR]
```

where:

label a valid assembler language label.

DWORDS={ n | (0) }
is the number of doublewords of free storage requested. DWORDS=n specifies the number of doublewords directly and DWORDS=(0) indicates that register 0 contains the number of doublewords requested.

MIN={ n | (1) }
indicates a variable request for free storage. If the exact number of doublewords indicated by the DWORDS operand is not available, the largest block of storage that is greater than or equal to the minimum is returned. MIN=n specifies the minimum number of doublewords of free storage directly while MIN=(1) indicates that the minimum is in register 1. The actual amount of free storage allocated is returned to the requesting routine via general register 0.

TYPE=[USER | NUCLEUS]
indicates the type of CMS storage with which this request for free storage is filled: USER or NUCLEUS.

```
ERR=┌      ┐
    │laddr│
    │  *  │
    └      ┘
```

is the return address if any error occurs. "laddr" is any address that can be referred to in a LOAD ADDRESS (LA) instruction. The error return is taken if there is a macro coding error or if there is not enough free storage available to fill the request. If * is specified for the return address, the error return is the same as a normal return.

```
AREA=┌     ┐
     │LOW  │
     │HIGH │
     └     ┘
```

indicates the area of CMS free storage from which this request for free storage is filled. LOW indicates the low storage area between DMSNUC and the the transient program area. HIGH indicates the area of storage between the user program area and the CMS loader tables. If AREA is not specified, storage is allocated wherever it is available.

```
TYPCALL=┌     ┐
        │SVC  │
        │BALR │
        └     ┘
```

indicates how control is passed to DMSFREE. Because DMSFREE is a nucleus-resident routine, other nucleus-resident routines can branch directly to it (TYPCALL=BALR) while routines that are not nucleus-resident must use linkage SVC (TYPCALL=SVC).

The pointers FREEUPPR and FREELOWE in NUCON indicate the amount of storage that DMSFREE has allocated from the high portion of the user program area. These pointers are initialized to the beginning of the loader tables.

The pointer FREELOWE is the "low-extend" pointer of DMSFREE storage in the user program area. As storage is allocated from the user program area to satisfy DMSFREE requests, this pointer is adjusted downward. Such adjustments are always in multiples of 4K bytes, so that this pointer is always on a 4K boundary. As the allocated storage is released, this pointer is adjusted upward.

The pointer FREELOWE can never be lower than MAINHIGH, the "high-extend" pointer for GETMAIN storage. If a DMSFREE request cannot be satisfied without extending FREELOWE below MAINHIGH, DMSFREE takes an error exit, indicating that insufficient storage is available to satisfy the request. Figure 22 shows the relationship of these storage areas.

The FREETAB free storage table is kept in free storage, usually in low-storage, just below the master file directory for the system disk (S-disk). However, the FREETAB may be located at the top of the user program area. This table contains one byte for each page of virtual storage. Each such byte contains a code indicating the use of that page of virtual storage. The codes in this table are as follows:

Code	Meaning
USERCODE (X'01')	The page is assigned to user storage.
NUCCODE (X'02')	The page is assigned to nucleus storage.
TRNCODE (X'03')	The page is part of the transient program area.
USARCODE (X'04')	The page is part of the user program area.
SYSCODE (X'05')	The page is none of the above. The page is assigned to system storage, system code, or the loader tables.

Other DMSFREE storage pointers are maintained in the DMSFRT CSECT, in NUCON. The four chain header blocks are the most important fields in DMSFRT. The four chains of unallocated elements are:

- The low-storage nucleus chain
- The low-storage user chain
- The high-storage nucleus chain
- The high-storage user chain

For each of these chains of unallocated elements, there is a control block consisting of four words, with the following format:

Offset				
0(0)	POINTER -- pointer to the first free element on the chain; it is zero, if the chain is empty.			
4(4)	NUM -- the number of elements on the chain.			
8(8)	MAX -- a value equal to or greater than the size of the largest element.			
12(C)	FLAGS-Flag byte	SKEY -Storage key	TCODE -FREETAB code	Unused

where:

POINTER points to the first element on this chain of free elements. If there are no elements on this free chain, the POINTER field contains all zeros.

NUM contains the number of elements on this chain of free elements. If there are no elements on this free chain, this field contains all zeros.

MAX avoids failing searches. It contains a number not exceeding the size, in bytes, of the largest element on the free chain. Thus, a search for an element of a given size is not be made if that size exceeds the MAX field. However, this number may actually be larger than the size of the largest free element on the chain.

FLAGS The following flags are used:

- FLCLN (X'80') -- Clean-up flag. This flag is set if the chain must be updated. This is necessary in the following circumstances:
  - If one of the two high-storage chains contains a 4K page that is pointed to by FREELOWE, then that page can be removed from the chain, and FREELOWE can be increased.
  - All completely unallocated 4K pages are kept on the user chain, by convention. Thus, if one of the nucleus chains (low-storage or high-storage) contains a full page, then this page must be transferred to the corresponding user chain.
- FLCLB (X'40') -- Destroyed flag. Set if the chain has been destroyed.
- FLHC (X'20') -- High-storage chain. Set for both the nucleus and user high-storage chains.
- FLNU (X'10') -- Nucleus chain. Set for both the low-storage and high-storage nucleus chains.
- FLPA (X'08') -- Page available. This flag is set if there is a full 4K page available on the chain. This flag may be set even if there is no such page available.

SKEY contains the 1-byte storage key assigned to storage on this chain.

TCODE contains the 1-byte FREETAB table code for storage on this chain.

Allocating User Free Storage

When DMSFREE with TYPE=USER (the default) is called, one or more of the following steps are taken in an attempt to satisfy the request. As soon as one of the following steps succeeds, then user free storage allocation processing terminates.

1. Search the low-storage user chain for a block of the required size.
2. Search the high-storage user chain for a block of the required size.
3. Extend high-storage user storage downward into the user program area, modifying FREELOWE in the process.
4. For a variable request, put all available storage in the user program area onto the high-storage user chain, and then allocate the largest block available on either the high-storage user chain or the low-storage

user chain. The allocated block will not be satisfactory unless it is larger than the minimum requested size.

Allocating Nucleus Free Storage

When DMSFREE with TYPE=NUCLEUS is called, the following steps are taken in an attempt to satisfy the request, until one succeeds:

1. Search the low-storage nucleus chain for a block of the required size.
2. Get free pages from the low-storage user chain, if any are available, and put them on the low-storage nucleus chain.
3. Search the high-storage nucleus chain for a block of the required size.
4. Get free pages from the high-storage user chain, if they are available, and put them on the high-storage nucleus chain.
5. Extend high-storage nucleus storage downward into the user program area, modifying FREELOWE in the process.
6. For variable requests, put all available pages from the user chains and the user program area onto the nucleus chains, and allocate the largest block available on either the low-storage nucleus chains, or the high-storage nucleus chains.

Releasing Storage

The DMSFRET macro releases free storage previously allocated with the DMSFREE macro. The format of the DMSFRET macro is:

```
┌──────────────────────────────────────────────────────────────────────────┐
│ [label] │ DMSFRET │ DWORDS={ n  },LOC={laddr}                           │
│         │         │        {(0) }     { (1) }                           │
│         │         │ ┌      ┌      ┐┐ ┌           ┌    ┐┐                │
│         │         │ │,ERR=│laddr││ │,TYPCALL=│SVC ││                │
│         │         │ │      │  *   ││ │           │BALR││                │
│         │         │ └      └      ┘┘ └           └    ┘┘                │
└──────────────────────────────────────────────────────────────────────────┘
```

where:

label is any valid assembler language label.

DWORDS={ n | (0) }

is the number of doublewords of storage to be released. DWORDS=n specifies the number of doublewords directly and DWORDS=(0) indicates that register 0 contains the number of doublewords being released.

LOC={ laddr | (1) }

is the address of the block of storage being released. "laddr" is any address that can be refereed to in a LOAD ADDRESS (LA) instruction. LOC=laddr specifies the address directly while LOC=(1) indicates the address is in register 1.

ERR=[laddr | *]

is the return address if an error occurs. "laddr" is any address that can be referred to by a LOAD ADDRESS (LA) instruction. The error return is taken if there is a macro coding error or if there is a problem returning the storage. If * is specified, the error return address is the same as the normal return address.

TYPCALL=[SVC | BALR]

indicates how control is passed to DMSFRET. Because DMSFRET is a nucleus-resident routine, other nucleus-resident routines can branch directly to it (TYPCALL=BALR) while routines that are not nucleus-resident must use SVC linkage (TYPCALL=SVC).

When DMSFRET is called, the block being released is placed on the appropriate chain. At that point, the final update operation is performed, if necessary, to advance FREELOWE, or to move pages from the nucleus chain to the corresponding user chain.

Similar update operations are performed, when necessary, after calls to DMSFREE, as well.

Releasing Allocated Storage

Storage allocated by the GETMAIN macro instruction may be released in any of the following ways:

1. A specific block of such storage may be released by means of the FREEMAIN macro instruction.
2. The STRINIT macro instruction releases all storage allocated by any previous GETMAIN requests.

3. Almost all CMS commands issue a STRINIT macro instruction. Thus, executing almost any CMS command causes all GETMAIN storage to be released.

Storage allocated by the DMSFREE macro instruction may be released in any of the following ways:

1. A specific block of such storage may be released by means of the DMSFRET macro instruction.

2. Whenever any user routine or CMS command abnormally terminates (so that the routine DMSABN is entered), and the ABEND recovery facility of the system is invoked, all DMSFREE storage with TYPE=USER is released automatically.

Except in the case of ABEND recovery, storage allocated by the DMSFREE macro is never released automatically by the system. Thus, storage allocated by means of this macro instruction should always be released explicitly by means of the DMSFRET macro instruction.

DMSFREE Service Routines

The system uses the DMSFRES macro instruction to request certain free storage management services.

The format of the DMSFRES macro is:

```
[label] DMSFRES  INIT1    [,TYPCALL= {SVC }]
                 INIT2               {BALR}
                 CHECK
                 CKON
                 CKOFF
                 UREC
                 CALOC
```

where:

label is any valid assembler language label.

INIT1 invokes the first free storage initialization routine, so that free storage requests can be made to access the system disk. Before this routine is invoked, no free storage requests may be made. After this routine has been invoked, free storage requests may be made, but these are subject to the following restraints until the second free storage management initialization routine has been invoked:

- All requests for USER type storage are changed to requests for NUCLEUS type storage.
- Error checking is limited before initialization is complete. In particular, it is sometimes possible to release a block which was never allocated.
- All requests that are satisfied in high storage must be of a temporary nature, since all storage allocated in high storage is released when the second free storage initialization routine is invoked.

When CP's saved system facility is used, the CMS system is saved at the point just after the A-Disk has been made accessible. It is necessary for DMSFRE to be used before the size of virtual storage is known, since the saved system can be used on any size virtual machine. Thus, the first initialization routine initializes DMSFRE so that limited functions can be requested, while the second initialization routine performs the initialization necessary to allow the full functions of DMSFRE to be exercised.

INIT2 invokes the second initialization routine. This routine is invoked after the size of virtual storage is known, and it performs initialization necessary to allow all the functions of DMSFRE to be used. The second initialization routine performs the following steps:

1. Releases all storage that has been allocated in the high-storage area.
2. Allocates the FREETAB free storage table. This table contains one byte for each 4K page of virtual storage, and so cannot be allocated until the size of virtual storage is known.
3. The FREETAB table is initialized, and all storage protection keys are initialized.
4. All completely unallocated 4K pages on the low-storage nucleus free storage chain are removed to the user chain. Any other necessary operations are performed.

CHECK invokes a routine which checks all free storage chains for consistency and correctness. Thus, it checks to see whether any free storage pointers have been destroyed. This option can be used at any time for system debugging.

CKON turns on a flag which causes the CHECK routine to be invoked each time a call is made to DMSFREE or DMSFRET. This can be useful for debugging purposes (for example, when you wish to identify the routine destroying free storage management pointers). Care should be taken when using this option, since the CHECK routine is coded to be thorough rather than efficient. Thus, after the CKON option has been invoked, each call to DMSFREE or DMSFRET takes much longer to be completed than before.

CKOFF turns off the flag that was turned on by the CKON option.

UREC is used by DMSABN during the ABEND recovery process to release all user storage.

CALOC is used by DMSABN after the ABEND recovery process has been completed. It invokes a routine that returns, in register 0, the number of doublewords of free storage that have been allocated. DMSABN uses this number to determine whether ABEND recovery has been successful.

Error Codes from DMSFRES, DMSFREE, and DMSFRET

A nonzero return code upon return from DMSFRES, DMSFREE, or DMSFRET indicates that the request could not be satisfied. Register 15 contains this return code, indicating which error has occurred. The following codes apply to the DMSFRES, DMSFREE, and DMSFRET macros.

<u>Code</u> <u>Error</u>

1 (DMSFREE) Insufficient storage space is available to satisfy the request for free storage. In the case of a variable request, even the minimum request could not be satisfied.

2 (DMSFREE or DMSFRET) User storage pointers destroyed.

3 (DMSFREE, DMSFRET, or DMSFRES) Nucleus storage pointers destroyed.

4 (DMSFREE) An invalid size was requested. This error exit is taken if the requested size is not greater than zero. In the case of variable requests, this error exit is taken if the minimum request is greater than the maximum request. (However, the latter error is not detected if DMSFREE is able to satisfy the maximum request.)

5 (DMSFRET) An invalid size was passed to the DMSFRET macro. This error exit is taken if the specified length is not positive.

6 (DMSFRET) The block of storage which is being released was never allocated by DMSFREE. Such an error is detected if one of the following errors is found:

a. The block does not lie entirely inside either the low-storage free storage area or the user program area between FREELOWE and FREEUPPR.

b. The block crosses a page boundary that separates a page allocated for USER storage from a page allocated for NUCLEUS type storage.

<u>Code</u> <u>Error</u>

c. The block overlaps another block already on the free storage chain.

7 (DMSFRET) The address given for the block being released is not doubleword aligned.

8 (DMSFRES) An invalid request code was passed to the DMSFRES routine. Because all request codes are generated by the DMSFRES macro, this error code should never appear.

9 (DMSFREE, DMSFRET, or DMSFRES) Unexpected and unexplained error in the free storage management routine.

CMS HANDLING OF PSW KEYS

The the CMS nucleus protection scheme protects the CMS nucleus from inadvertent destruction by a user program. Without it, it would be possible, for example, for a FORTRAN user who accidentally assigns an incorrectly subscripted array element to destroy nucleus code, wipe out a crucial table or constant area, or even destroy an entire disk by destroying the contents of the master file directory.

In general, user programs and disk-resident CMS commands run with a PSW key of X'E', while nucleus code runs with PSW key of X'0'.

There are, however, some exceptions to this rule. Certain disk-resident CMS commands run with a PSW key of X'0', because they have a constant need to modify nucleus pointers and storage. The nucleus routines called by the GET, PUT, READ, and WRITE macros run with a user PSW key of X'E', to increase efficiency.

Two macros are available to any routine that wishes to change its PSW key for some special purpose. These are the DMSKEY macro and the DMSEXS macro.

The DMSKEY macro may be used to change the PSW key to the user value or the nucleus value. The DMSKEY NUCLEUS option causes the current PSW key to be placed in a stack, and a value of 0 to be placed in the PSW key. The DMSKEY USER option causes the current PSW key to be placed in a stack, and a value of X'E' to be placed in the PSW key. The DMSKEY RESET option causes the top value in the DMSKEY stack to be removed and re-inserted into the PSW.

It is a requirement of the CMS system that when a routine terminates, the DMSKEY stack must be empty. This means that a routine should execute a DMSKEY RESET option for each DMSKEY NUCLEUS option and each DMSKEY USER option executed by the routine.

The DMSKEY key stack has a current maximum depth of seven for each routine. In this context, a "routine" is anything invoked by an SVC call.

The DMSKEY LASTUSER option causes the current PSW key to be placed in the stack, and a new key inserted into the PSW, determined as follows: the SVC system save area stack is searched in reverse order (top to bottom) for the first save area corresponding to a user routine. The PSW key which was in effect in that routine is then taken for the new PSW key. (If no user routine is found in the search, then LASTUSER has the same effect as USER.) This option is used by OS macro simulation routines when they must enter a user-supplied exit routine; the exit routine is entered with the PSW key of the last user routine on the SVC system save area stack.

The NOSTACK option of DMSKEY may be used with NUCLEUS, USER, or LASTUSER (as in, for example, DMSKEY NUCLEUS,NOSTACK) if the current key is not to be placed on the DMSKEY stack. If this option is used, then no corresponding DMSKEY RESET should be issued.

The DMSEXS ("execute in system mode") macro instruction is useful in situations where a routine is running with a user protect key, but wishes to execute a single instruction which, for example, sets a bit in the NUCON area. The single instruction may be specified as the argument to the DMSEXS macro, and that instruction will be executed with a system PSW key.

Whenever possible, CMS commands run with a user protect key. This protects the CMS nucleus in cases where there is an error in the system command which would otherwise destroy the nucleus. If the command must execute a single instruction or small group of instructions that modify nucleus storage, then the DMSKEY or DMSEXS macros are used, so that the system PSW key will be used for as short a period of time as possible.

CMS SVC HANDLING

DMSITS (INTSVC) is the CMS system SVC handling routine. The general operation of DMSITS is as follows:

1. The SVC new PSW (low-storage location X'60') contains, in the address field, the address of DMSITS1. The DMSITS module will be entered whenever a supervisor call is executed.
2. DMSITS allocates a system and user save area. The user save area is used as a register save area (or work area) by the called routine.
3. The called routine is called (via a LPSW or BALR).
4. Upon return from the called routine, the save areas are released.
5. Control is returned to the caller (the routine which originally made the SVC call).

SVC TYPES AND LINKAGE CONVENTIONS

SVC conventions are important to any discussion of CMS because the system is driven by SVCs (supervisor calls). SVCs 202 and 203 are the most common CMS SVCs.

SVC 202

SVC 202 is used both for calling nucleus resident routines, and for calling routines written as commands (for example, disk resident modules).

A typical coding sequence for an SVC 202 call is the following:

```
LA   R1,PLIST
SVC  202
DC   AL4(ERRADD)
```

Whenever SVC 202 is called, register 1 must point to a parameter list (PLIST). The format of this parameter list depends upon the actual routine or command being called, but the SVC handler will examine the first eight bytes of this parameter list to find the name of the routine or command being called.

The "DC AL4(address)" instruction following the SVC 202 is optional, and may be omitted if the programmer does not expect any errors to occur in the routine or command being called. If included, an error return is made to the address specified in the DC. DMSITS determines whether this DC was inserted by examining the byte following the SVC call inline. A nonzero byte indicates an instruction, a zero value indicates that "DC AL4(address)" follows.

SVC 203

SVC 203 is called by CMS macros to perform various internal system functions. It is used to define SVC calls for which no parameter list is provided. For example, DMSFREE parameters are passed in registers 0 and 1.

A typical calling sequence for an SVC 203 call is as follows:

```
SVC   203
DC    H'code'
```

The halfword decimal code following the SVC 203 indicates the specific routine being called. DMSITS examines this halfword code, taking the absolute value of the code by an LPR instruction. The first byte of the result is ignored, and the second byte of the resulting halfword is used as an index to a branch table. The address of the correct routine is loaded, and control is transferred to it.

It is possible for the address in the SVC 203 index table to be zero. In this case, the index entry will contain an 8-byte routine or command name, which will be handled in the same way as the 8-byte name passed in the parameter list to an SVC 202.

The programmer indicates an error return by the sign of the halfword code. If an error return is desired, then the code is negative. If the code is positive, then no error return is made. The sign of the halfword code has no effect on determining the routine which is to be called, since DMSITS takes the absolute value of the code to determine the routine called.

Since only the second byte of the absolute value of the code is examined by DMSITS, seven bits (bits 1-7) are available as flags or for other uses. Thus, for example, DMSFREE uses these seven bits to indicate such things as conditional requests and variable requests.

When an SVC 203 is invoked, DMSITS stores the halfword code into the NUCON location CODE203, so that the called routine can examine the seven bits made available to it.

All calls made by means of SVC 203 should be made by macros, with the macro expansion computing and specifying the correct halfword code.

User Handled SVCs

The programmer may use the HNDSVC macro to specify the address of a routine which will handle any SVC call other than for SVC 202 and SVC 203.

In this case, the linkage conventions are as required by the user-specified SVC-handling routine.

OS and DOS/VS Macro Simulation SVC Calls

CMS supports selected SVC calls generated by OS and DOS/VS macros, by simulating the effect of these macro calls. DMSITS is the initial SVC interrupt handler. If the SET DOS command has been issued, a flag in NUCON will indicate that DOS/VS macro simulation is to be used. Control is then passed to DMSDOS. Otherwise, OS macro simulation is assumed and DMSITS passes control to the appropriate OS simulation routine.

Invalid SVC Calls

There are several types of invalid SVC calls recognized by DMSITS.

1. Invalid SVC number. If the SVC number does not fit into any of the four classes described above, then it is not handled by DMSITS. An appropriate error message is displayed at the terminal, and control is returned directly to the caller.
2. Invalid routine name in SVC 202 parameter list. If the routine named in the SVC 202 parameter list is invalid or cannot be found, DMSITS handles the situation in the same way it handles an error return from a legitimate SVC routine. The error code is -3.
3. Invalid SVC 203 code. If an invalid code follows SVC 203 inline, then an error message is displayed, and the ABEND routine is called to terminate execution.

Search Hierarchy for SVC 202

When a program issues SVC 202, passing a routine or command name in the parameter list, then DMSITS must be searched for the specified routine or command. (In the case of SVC 203 with a zero in the table entry for the specified index, the same logic must be applied.)

The search algorithm is as follows:

1. First, a check is made to see if there is a routine with the specified name currently occupying the system Transient Area. If this is the case, then control is transferred there.
2. Second, the system function name table is searched, to see if a command by this name is nucleus-resident. If successful, control goes to the specified nucleus routine.
3. Next, a search is made for a disk file with the specified name as the filename, and MODULE as the filetype. The search is made in the standard disk search order. If this search is successful, then the specified module is loaded (via the LOADMOD command), and control passes to the storage location now occupied by the command.
4. If all searches so far have failed, then DMSINA (ABBREV) is called, to see if the specified routine name is a valid system abbreviation for a system command or function. User-defined abbreviations and synonyms are also checked. If this search is successful, then steps 2 through 4 are repeated with the full function name.
5. If all searches fail, then an error code of -3 is issued.

Commands Entered from the Terminal

When a command is entered from the terminal, DMSINT processes the command line, and calls the scan routine to convert it into a parameter list consisting of eight-byte entries. The following search is performed:

1. DMSINT searches for a disk file whose filename is the command name, and whose filetype is EXEC. If this search is successful, EXEC is invoked to process the EXEC file.

If not found, the command name is considered to be an abbreviation and the appropriate tables are examined. If found, the abbreviation is replaced by its full equivalent and the search for an EXEC file is repeated.

2. If there is no EXEC file, DMSINT executes SVC 202, passing the scanned parameter list, with the command name in the first eight bytes. DMSITS will perform the search described for SVC 202 in an effort to execute the command.

3. If DMSITS returns to DMSINT with a return code of -3, indicating that the search was unsuccessful, then DMSINT uses the CP DIAGNOSE facility to attempt to execute the command as a CP command.

4. If all these searches fail, then DMSINT displays the error message UNKNOWN CP/CMS COMMAND.

See Figure 23 for a description of this search for a command name.

USER AND TRANSIENT PROGRAM AREAS

Two areas can hold programs that are loaded from disk. These are called the user program area and the transient program area. (See Figure 22 for a description of CMS storage usage.)

The user program area starts at location X'20000' and extends upward to the loader tables. Generally, all user programs and certain system commands (such as EDIT, and COPYFILE) run in the user program area. Because only one program can be running in the user program area at any one time, it is impossible (without unpredictable results) for one program running in the user program area to invoke, by means of SVC 202, a module that is also intended to be run in the user program area.

The transient program area is two pages long, running from location X'E000' to location X'FFFF'. It provides an area for system commands that may also be invoked from the user program area by means of an SVC 202 call. When a transient module is called by an SVC, it is normally run with the PSW system mask disabled for I/O and external interruptions.

The transient program area also handles certain OS macro simulation SVC calls. OS SVC calls are handled by the OS simulation routines located either in the CMSSEG discontiguous shared segment or in the user program area, as close to the loader tables as possible. If DMSITS cannot find the address of a supported OS SVC handling routine, then it loads the file DMSSVT MODULE into the transient area, and lets that routine handle the SVC.

A program running in the transient program area may not invoke another program intended to run in the transient program area, including OS macro simulation SVC calls that are handled by DMSSVT. For example, a program running in the transient program area may not invoke the RENAME command. In addition, it may not invoke the OS macro WTO, which generates an SVC 35, which is handled by DMSSVT.

DMSITS starts programs running in the user program area enabled for all interruptions but starts programs running in the transient program area disabled for all interruptions. The individual program may have to use the SSM (SET SYSTEM MASK) instruction to change the current status of its system mask.

CALLED ROUTINE START-UP TABLE

Figures 24 and 25 show how the PSW and registers are set up when the called routine is entered.

RETURNING TO THE CALLING ROUTINE

When the called routine finishes processing, control is returned to DMSITS, which in turn returns control to the calling routine.

Return Location

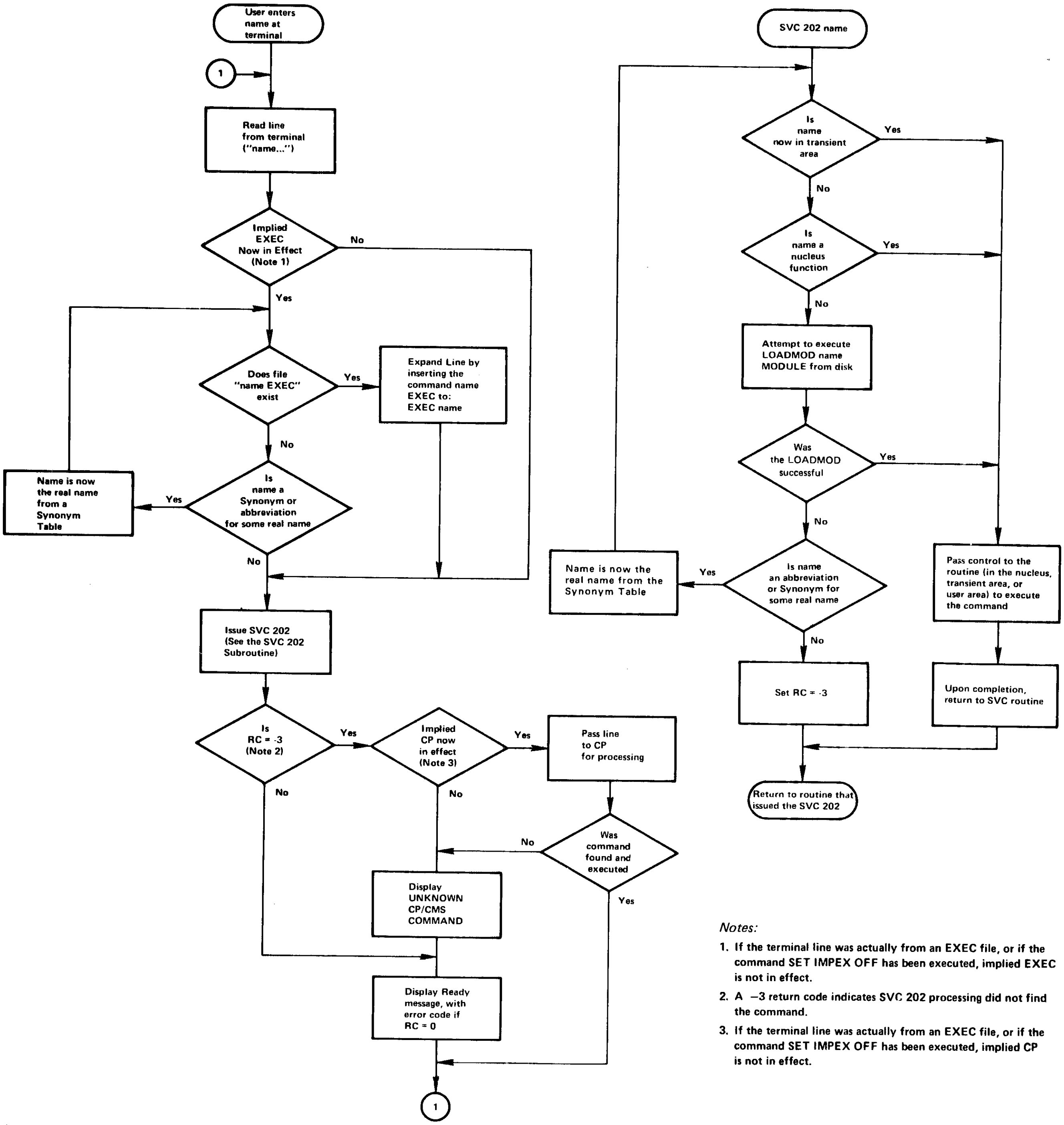
The return is accomplished by loading the original SVC old PSW (which was saved at the time DMSITS was first entered), after possibly modifying the address field. The address field modification depends upon the type of SVC call, and on whether the called routine indicated an error return.

For SVC 202 and 203, the called routine indicates a normal return by placing a zero in register 15, and an error return by placing a nonzero code in register 15. If the called routine indicates a normal return, then DMSITS makes a normal return to the calling routine. If the called routine indicates an error return, DMSITS passes the error return to the calling routine, if one was specified, and abnormally terminates if none was specified.

For an SVC 202 not followed by "DC AL4(address)", a normal return is made to the instruction following the SVC instruction, and an error return causes an ABEND. For an SVC 202 followed by "DC AL4(address)", a normal return is made to the instruction following the DC, and an error return is made to the address specified in the DC. In either case, register 15 contains the return code passed back by the called routine.

For an SVC 203 with a positive halfword code, a normal return is made to the instruction following the halfword code, and an error return causes an ABEND. For an SVC 203 with a negative halfword code, both normal and error returns are made to the instruction following the halfword code. In any case, register 15 contains the return code passed back by the called routine.

For macro simulation SVC calls, and for user-handled SVC calls, no error return is



Notes:

1. If the terminal line was actually from an EXEC file, or if the command SET IMPEX OFF has been executed, implied EXEC is not in effect.
2. A −3 return code indicates SVC 202 processing did not find the command.
3. If the terminal line was actually from an EXEC file, or if the command SET IMPEX OFF has been executed, implied CP is not in effect.

Figure 23. CMS Command (and Request) Processing

Called Type	System Mask	Storage Key	Problem Bit
SVC 202 or 203 - Nucleus resident	Disabled	System	Off
SVC 202 or 203 - Transient area MODULE	Disabled	User	Off
SVC 202 or 203 - User area	Enabled	User	Off
User-handled	Enabled	User	Off
OS - DOS/VS Nucleus resident	Disabled	System	Off
OS - DOS/VS Transient area module	Disabled	System	Off

Figure 24. PSW Fields When Called Routine Starts

Type	Registers 0 - 1	Registers 2 - 11	Register 12	Register 13	Register 14	Register 15
SVC 202 or 203	Same as caller	Unpredic-table	Address of called routine	User save area	Return address to DMSITS	Address of called routine
Other	Same as caller	Same as caller	Address of caller	User save area	Return address to DMSITS	Same as caller

Figure 25. Register Contents When Called Routine Starts

recognized by DMSITS. As a result, DMSITS always returns to the calling routine by loading the SVC old PSW which was saved when DMSITS was first entered.

Register Restoration

Upon entry to DMSITS, all registers are saved as they were when the SVC instruction was first executed. Upon exiting from DMSITS, all registers are restored from the area in which they were saved at entry.

The exception to this is register 15 in the case of SVC 202 and 203. Upon return to the calling routine, register 15 always contains the value which was in register 15 when the called routine returned to DMSITS after it had completed processing.

Called Routine Modifications to System Area

If the called routine has system status, so that it runs with a PSW storage protect key of 0, then it may store new values into the System Save Area.

If the called routine wishes to modify the location to which control is to be returned, it must modify the following fields:

- For SVC 202 and 203, it must modify the NUMRET and ERRET (normal and error return address) fields.
- For other SVCs, it must modify the address field of OLDPSW.

To modify the registers that are to be returned to the calling routine, the fields EGPR1, EGPR2, ..., EGPR15 must be modified.

If this action is taken by the called routine, then the SVCTRACE facility may print misleading information, since SVCTRACE assumes that these fields are exactly as they were when DMSITS was first entered. Whenever an SVC call is made, DMSITS allocates two save areas for that particular SVC call. Save areas are allocated as needed. For each SVC call, a system and user save area are needed.

When the SVC called routine returns, the save areas are not released, but are kept for the next SVC. At the completion of each command, all SVC save areas allocated by that command are released.

The system save area is used by DMSITS to save the value of the SVC old PSW at the time of the SVC call, the calling routine's registers at the time of the call, and any other necessary control information. Because SVC calls can be nested, there can be several of these save areas at one time. The system save area is allocated in protected free storage.

The user save area contains 12 doublewords (24 words), allocated in unprotected free storage. DMSITS does not use this area at all, but simply passes a pointer to this area (via register 13.) The called routine can use this area as a temporary work area, or as a register save area. There is one user save area for each system save area. The field USAVEPTR in the system save area points to the user save area.

The exact format of the system save area can be found in the VM/370: Data Areas and Control Block Logic. The most important fields, and their uses, are as follows:

CALLER (Fullword) The address of the SVC instruction that resulted in this call.

CALLEE (Doubleword) Eight-byte symbolic name of the called routine. For OS and user-handled SVC calls, this field contains a character string of the form SVC nnn, where nnn is the SVC number in decimal.

CODE (Halfword) For SVC 203, this field contains the halfword code following the SVC instruction line.

OLDPSW (Doubleword) The SVC old PSW at the time that DMSITS was entered.

NRMRET (Fullword) The address of the calling routine to which control is to be passed in the case of a normal return from the called routine.

ERRET (Fullword) The address of the calling routine to which control is to be passed in the case of an error return from the called routine.

EGPRS (16 Fullwords, separately labeled EGPR0, EGPR1, EGPR2, EGPR3, ..., EGPR15) The entry registers. The contents of the general registers at entry to DMSITS are stored in these fields.

EFPRS (4 Doublewords, separately labeled EFPR0, EFPR2, EFPR4, EFPR6) The entry floating-point registers. The contents of the floating-point registers at entry to DMSITS are stored in these fields.

SSAVENXT (Fullword) The address of the next system save area in the chain. This points to the system save area which is being used, or will be used, for any SVC call nested in relation to the current one.

SSAVEPRV (Fullword) The address of the previous system save area in the chain. This points to the system save area for the SVC call in relation to which the current call is nested.

USAVEPTR (Fullword) Pointer to the user save area for this SVC call.

CMS INTERFACE FOR DISPLAY TERMINALS

CMS has an interface that allows it to display large amounts of data in a very rapid fashion. This interface for display terminals is much faster and has less overhead than the normal write because it displays up to 1760 characters in one operation, instead of issuing 22 individual writes of 80 characters each (that is one write per line on a display terminal). Data that is displayed in the screen output area with this interface is not placed in the console spool file.

The DISPW macro allows you to use this display terminal interface. It generates a calling sequence for the CMS display terminal interface module, DMSGIO. DMSGIO creates a channel program and issues a DIAGNOSE instruction (Code 58) to display the data. DMSGIO is a TEXT file which must be loaded in order to use DISPW. The format of the CMS DISPW macro is:

```
[label]  | DISPW | bufad  ┌,LINE=n┐  ┌,BYTES=bbbb┐
         |       |        │,LINE=0│  │,BYTES=1760│
         |       |        └       ┘  └           ┘
         |       |        [ERASE=YES]  [CANCEL=YES]
```

where:

label — is an optional macro statement label.

bufad — is the address of a buffer containing the data to be written to the display terminal.

```
┌      ┐
|LINE=n|
|LINE=0|
└      ┘
```

is the number of the line, 0 to 23, on the display terminal that is to be written. Line number 0 is the default.

```
┌         ┐
|BYTES=bbbb|
|BYTES=1760|
└         ┘
```

is the number of bytes (0 to 1760) to be written on the display terminal. 1760 bytes is the default.

[ERASE=YES]
specifies that the display screen is to be erased before the current data is written. The screen is erased regardless of the line or number of bytes to be displayed. Specifying ERASE=YES causes the screen to go into "MORE" status.

[CANCEL=YES]
causes the CANCEL operation to be performed: the output area is erased.

OS MACRO SIMULATION UNDER CMS

When a language processor or a user-written program is executing in the CMS environment and using OS-type functions, it is not executing OS code. Instead, CMS provides routines that simulate the OS functions required to support OS language processors and their generated object code.

CMS functionally simulates the OS macros in a way that presents equivalent results to programs executing under CMS. The OS macros are supported only to the extent stated in the publications for the supported language processors, and then only to the extent necessary to successfully satisfy the specific requirement of the supervisory function.

The restrictions for COBOL and PL/I program execution listed in "Executing a Program that Uses OS Macros" in the VM/370: Planning and System Generation Guide exist because of the limited simulation by CMS of the OS macros.

Figure 26 shows the OS macro functions that are partially or completely simulated, as defined by SVC number.

OS DATA MANAGEMENT SIMULATION

The disk format and data base organization of CMS are different from those of OS. A CMS file produced by an OS program running under CMS and written on a CMS disk, has a different format than that of an OS data set produced by the same OS program running under OS and written on an OS disk. The data is exactly the same, but its format is different. (An OS disk is one that has been formatted by an OS program, such as IBCDASDI.)

HANDLING FILES THAT RESIDE ON CMS DISKS

CMS can read, write, or update any OS data that resides on a CMS disk. By simulating OS macros, CMS simulates the following access methods so that OS data organized by these access methods can reside on CMS disks:

direct — identifying a record by a key or by its relative position within the data set.

partitioned — seeking a named member within the data set.

sequential — accessing a record in a sequence relative to preceding or following items in the data set.

Refer to Figure 26 and the "Simulation Notes", then read "Access Method Support" to see how CMS handles these access methods.

Because CMS does not simulate the indexed sequential access method (ISAM), no OS program which uses ISAM can execute under CMS. Therefore, no program can write an indexed sequential data set on a CMS disk.

HANDLING FILES THAT RESIDE ON OS OR DOS DISKS

By simulating OS macros, CMS can read, but not write or update, OS sequential and partitioned data sets that reside on OS disks. Using the same simulated OS macros, CMS can read DOS sequential files that reside on DOS disks. The OS macros handle the DOS data as if it were OS data. Thus a DOS sequential file can be an input to an OS program running under CMS.

However, an OS sequential or partitioned data set that resides on an OS disk can be written or updated only by an OS program running in a real OS machine.

CMS can execute programs that read and write VSAM files from OS programs written in the VS BASIC, COBOL, or PL/I programming languages. This CMS support is based on the DOS/VS Access Method Services and Virtual Storage Access Method (VSAM) and therefore the OS user is limited to those VSAM functions that are available under DOS/VS.

Macro Title	SVC Number	Function
XDAP[1]	00	Read or write direct access volumes
WAIT	01	Wait for an I/O completion
POST	02	Post the I/O completion
EXIT/RETURN	03	Return from a called phase
GETMAIN	04	Conditionally acquire user storage
FREEMAIN	05	Release user-acquired storage
GETPOOL	-	Simulate as SVC 10
FREEPOOL	-	Simulate as SVC 10
LINK	06	Link control to another phase
XCTL	07	Delete, then link control to another load phase
LOAD	08	Read a phase into storage
DELETE	09	Delete a loaded phase
GETMAIN/ FREEMAIN	10	Manipulate user free storage
TIME[1]	11	Get the time of day
ABEND	13	Terminate processing
SPIE[1]	14	Allow processing program to handle program interrupts
RESTORE[1]	17	Effective NOP
BLDL/FIND[1]	18	Manipulate simulated partitioned data files
OPEN	19	Activate a data file
CLOSE	20	Deactivate a data file
STOW[1]	21	Manipulate partitioned directories
OPENJ	22	Activate a data file
TCLOSE	23	Temporarily deactivate a data file
DEVTYPE[1]	24	Obtain device-type physical characteristics
TRKBAL	25	NOP
WTO/WTOR[1]	35	Communicate with the terminal
EXTRACT[1]	40	Effective NOP
IDENTIFY[1]	41	Add entry to loader table
ATTACH[1]	42	Effective LINK
CHAP[1]	44	Effective NOP
TTIMER[1]	46	Access or cancel timer
STIMER[1]	47	Set timer
DEQ[1]	48	Effective NOP
SNAP[1]	51	Dump specified areas of storage
ENQ[1]	56	Effective NOP
FREEDBUF	57	Release a free storage buffer
STAE	60	Allow processing program to decipher ABEND conditions
DETACH[1]	62	Effective NOP
CHKPT[1]	63	Effective NOP
RDJFCB[1]	64	Obtain information from FILEDEF command
SYNAD[1]	68	Handle data set error conditions
BSP[1]	69	Backup a record on a tape or disk
GET/PUT	-	Access system-blocked data
READ/WRITE	-	Access system-record data
NOTE/POINT	-	Manage data set positioning
CHECK	-	Verify READ/WRITE completion
TGET/TPUT	93	Read or write a terminal line
TCLEARQ	94	Clear terminal input queue
STAX	96	Create an attention exit block

[1]Simulated in the transient routine "DMSSVT". Other simulation routines reside in the nucleus.

Figure 26. Simulated OS Supervisor Calls

SIMULATION NOTES

Because CMS has its own file system and is a single-user system operating in a virtual machine with virtual storage, there are certain restrictions for the simulated OS function in CMS. For example, HIARCHY options and options that are used only by OS multitasking systems are ignored by CMS.

Listed below are descriptions of all the OS macro functions that are simulated by CMS as seen by the programmer. Implementation and program results that differ from those given in *OS/VS Data Management Macro Instructions* and *OS/VS Supervisor Services and Macro Instructions* are stated. HIARCHY options and those used only by OS multitasking systems are ignored by CMS. Validity checking is not performed within the simulation routines. The entry point name in LINK, XCTL, and LOAD (SVC 6, 7, 8) must be a member name or alias in a TXTLIB directory unless the COMPSWT is set to on. If the COMPSWT is on, SVC 6, 7, and 8 must specify a MODULE name. This switch is turned on and off by using the COMPSWT macro. See the *VM/370: CMS Command and Macro Reference* for descriptions of all CMS user macros.

Macro-SVC No.	Differences in Implementation
XDAP-SVC0	The TYPE option must be R or W; the V, I, and K options are not supported. The BLKREF-ADDR must point to an item number acquired by a NOTE macro. Other options associated with V, I, or K are not supported.
WAIT-SVC1	All options of WAIT are supported. The WAIT routine waits for the completion bit to be set in the specified ECBs.
POST-SVC2	All options of POST are supported. POST sets a completion code and a completion bit in the specified ECB.
EXIT/RETURN -SVC3	Post ECB, execute end of task routine, release phase storage, unchain and free latest request block, and restore registers depending on whether this is an exit or return from a linked or an attached routine.
GETMAIN-SVC4	All the options of GETMAIN are supported. GETMAIN gets blocks of free storage.
FREEMAIN-SVC5	All the options of FREEMAIN are supported. FREEMAIN frees blocks of storage acquired by GETMAIN.
LINK-SVC6	The DCB and HIARCHY options are ignored by CMS. All other options of LINK are supported. LINK loads the specified program into storage (if necessary) and passes control to the specified entry point.
XCTL-SVC7	The DCB and HIARCHY options are ignored by CMS. All other options of XCTL are supported. XCTL loads the specified program into storage (if necessary) and passes control to the specified entry point.
LOAD-SVC8	The DCB and HIARCHY options are ignored by CMS. All other options of LOAD are supported. LOAD loads the specified program into storage (if necessary) and returns the address of the specified entry point in register zero. However, if the specified entry point is not in core when SVC 8 is issued, and the subroutine contains VCONs which cannot be resolved within that TXTLIB member, CMS attempts to resolve these references, and may return another entry point address. To insure a correct address in register zero, the user should bring such subroutines into core either by the CMS LOAD/INCLUDE commands or by a VCON in the user program.
GETPOOL/ FREEPOOL	All the options of GETPOOL and FREEPOOL are supported. GETPOOL constructs a buffer pool and stores the address of a buffer pool control block in the DCB. FREEPOOL frees a buffer pool constructed by GETPOOL.
DELETE-SVC9	All the options of DELETE are supported. DELETE decreases the use count by one and if the result is zero frees the corresponding virtual storage. Code 4 is returned in register 15 if the phase is not found.
GETMAIN/ FREEMAIN- SVC10	All the options of GETMAIN and FREEMAIN are supported. Subpool specifications are ignored.
TIME-SVC11	All the options of TIME except MIC are supported. TIME returns the time of day to the calling program.
ABEND-SVC13	The completion code parameter is supported. The DUMP parameter is not. If a STAE request is outstanding, control is given to the proper STAE routine. If a STAE routine is not outstanding, a message indicating an ABEND has occurred is printed on the terminal along with the completion code.
SPIE-SVC14	All the options of SPIE are supported. The SPIE routine specifies interruption exit routines and program interruption types that will cause the exit routine to receive control.
RESTORE-SVC17	The RESTORE routine in CMS is a NOP. It returns control to the user.

BLDL-SVC18 BLDL is an effective NOP for LINKLIBs and JOBLIBs. For MACLIBs, item numbers are filled in the TTR field of the BLDL list; the K, Z, and user data fields, as described in _OS/VS Data Management Macro Instructions_, are set to zeros. The 'alias' bit of the C field is supported, and the remaining bits in the C field are set to zero.

FIND-SVC18 All the options of FIND are supported. FIND sets the read/write pointer to the item number of the specified member.

STOW-SVC21 All the options of STOW are supported. The 'alias' bit is supported, but the user data field is not stored in the MACLIB directory since CMS MACLIBs do not contain user data fields.

OPEN/OPENJ-SVC19/22 All the options of OPEN and OPENJ are supported except for the DISP and RDBACK options which are ignored. OPEN creates a CMSCB (if necessary), completes the DCB, and merges necessary fields of the DCB and CMSCB.

CLOSE/TCLOSE-SVC20/23 All the options of CLOSE and TCLOSE are supported except for the DISP option, which is ignored. The DCB is restored to its condition before OPEN. If the device type is disk, the file is closed. If the device type is tape, the REREAD option is treated as a REWIND.

DEVTYPE-SVC24 All the options of DEVTYPE are supported. DEVTYPE moves device characteristic information for a specified data set into a specified user area.

WTO/WTOR-SVC35 All options of WTO and WTOR are supported except those options concerned with multiple console support. WTO displays a message at the operator's console. WTOR displays a message at the operator's console, waits for a reply, moves the reply to the specified area, sets a completion bit in the specified ECB, and returns.

EXTRACT-SVC40 The EXTRACT routine in CMS is essentially a NOP. The user provided answer area is set to zeros and control is returned to the user with a return code of 4 in register 15.

IDENTIFY-SVC41 The IDENTIFY routine in CMS adds a RPQUEST block to the load request chain for the requested name and address.

ATTACH-SVC42 All the options of ATTACH are supported in CMS as in OS PCP. The following options are ignored by CMS: DCB, LPMOD, DPMOD, HIARCHY, GSPV, GSPL, SHSPV, SHSPL, SZERO, PURGE, ASYNCH, and TASKLIB. ATTACH passes control to the routine specified, fills in an ECB completion bit if an ECB is specified, passes control to an exit routine if one is specified, and returns control to the instruction following the ATTACH.

Because CMS is not a multitasking system, a phase requested by the ATTACH macro must return to CMS.

CHAP-SVC44 The CHAP routine in CMS is a NOP. It returns control to the user.

TTIMER-SVC46 All the options of TTIMER are supported.

STIMER-SVC47 All options of STIMER are supported except for TASK and WAIT. The TASK option is treated as if the REAL option had been specified, and the WAIT option is treated as a NOP; it returns control to the user.

DEQ-SVC48 The DEQ routine in CMS is a NOP. It returns control to the user.

SNAP-SVC51 All the options of SNAP are supported except for the DCB, SDATA, and PDATA options, which are ignored. SNAP always dumps output to the printer. The dump contains the PSW, the registers, and the storage specified.

ENQ-SVC56 The ENQ routine in CMS is a NOP. It returns control to the user.

FREEDBUF-SVC57 All the options of FREEDBUF are supported. FREEDBUF returns a buffer to the buffer pool assigned to the specified DCB.

STAE-SVC60 All the options of STAE are supported except for the XCTL option, which is set to XCTL=YES; the PURGE option, which is set to HALT; and the ASYNCH option, which is set to NO. STAE creates, overlays, or cancels a STAE control block as requested. STAE retry is not supported.

DETACH-SVC62 — The DETACH routine in CMS is a NOP. It returns control to the user.

CHKPT-SVC63 — The CHKPT routine is a NOP. It returns control to the user.

RDJFCB-SVC64 — All the options of RDJFCB are supported. RDJFCB causes a Job File Control Block (JFCB) to be read from a CMS Control Block (CMSCB) into real storage for each data control block specified. CMSCBs are created by FILEDEF commands.

SYNADAF-SVC68 — All the options of SYNADAF are supported. SYNADAF analyzes an I/O error and creates an error message in a work buffer.

SYNADRLS-SVC68 — All the options of SYNADRLS are supported. SYNADRLS frees the work area acquired by SYNAD and deletes the work area from the save area chain.

BSP-SVC69 — All the options of BSP are supported. BSP decrements the item pointer by one block.

TGET/TPUT-SVC93 — TGET and TPUT operate as if EDIT and WAIT were coded. TGET reads a terminal line. TPUT writes a terminal line.

TCLEARQ-SVC94 — TCLEARQ in CMS clears the input terminal queue and returns control to the user.

STAX-SVC96 — Updates a queue of CMTAXEs each of which defines an attention exit level.

NOTE — All the options of NOTE are supported. NOTE returns the item number of the last block read or written.

POINT — All the options of POINT are supported. POINT causes the control program to start processing the next read or write operation at the specified item number. The TTR field in the block address is used as an item number.

CHECK — All the options of CHECK are supported. CHECK tests the I/O operation for errors and exceptional conditions.

DCB — The following fields of a DCB may be specified, relative to the particular access method indicated:

ACCESS METHOD SUPPORT

The manipulation of data is governed by an access method. To facilitate the execution of OS Code under CMS, the processing program must see data as OS would present it. For instance, when the processors expect an access method to acquire input source cards sequentially, CMS invokes specially written routines that simulate the OS sequential access method and pass data to the processors in the format that the OS access methods would have produced. Therefore, data appears in storage as if it had been manipulated using an OS access method. For example, block descriptor words (BDW), buffer pool management, and variable records are updated in storage as if an OS access method had processed the data. The actual writing to and reading from the I/O device is handled by CMS file management.

The essential work of the Volume Table of Contents (VTOC) and the Data Set Control Block (DSCB) is done in CMS by a Master File Directory (MFD) which updates the disk contents, and a File Status Table (FST) (one for each data file). All disks are formatted in physical blocks of 800 bytes.

CMS continues to update the OS format, within its own format, on the auxiliary device, for files whose filemode number is 4. That is, the block and record descriptor words (BDW and RDW) are written along with the data. If a data set consists of blocked records, the data is written to, and read from, the I/O device in physical blocks, rather than logical records. CMS also simulates the specific methods of manipulating data sets.

To accomplish this simulation, CMS supports certain essential macros for the following access methods:

- BDAM — (direct) -- identifying a record by a key or by its relative position within the data set.
- BPAM — (partitioned) -- seeking a named member within data set.
- BSAM/QSAM — (sequential) -- accessing a record in a sequence relative to preceding or following records.
- VSAM — (direct or sequential) -- accessing a record sequentially or directly by key or address. Note: CMS support of OS VSAM files is based on DOS/VS Access Method Services and Virtual Storage Access Method (VSAM). Therefore, the OS user is restricted to those functions available under DOS/VS Access Method Services. See the section "CMS Support for OS and DOS VSAM Functions" for details.

CMS also updates portions of the OS control blocks that are needed by the OS simulation routines to support a program during execution (see Figure 27). Most of the simulated supervisory OS control blocks are contained in the following two CMS control blocks:

Operand	BDAM	BPAM	BSAM	QSAM
BFALN	F,D	F,D	F,D	F,D
BLKSIZE	n(number)	n	n	n
BUFCB	a(address)	a	a	a
BUFL	n	n	n	n
BUFNO	n	n	n	n
DDNAME	s(symbol)	s	s	s
DSORG	DA	PO	PS	PS
EODAD	–	a	a	a
EXLST	a	a	a	a
KEYLEN	n	–	n	–
LIMCT	n	–	–	–
LRECL	–	n	n	n
MACRF	R,W	R,W	R,W, P	G,P,L,M
OPTCD	A,E,F,R	–	–	–
RECFM	F,V,U	F,V,U	F,V,B,S,A,M,U	F,V,B,U,A,M,S
SYNAD	a	a	a	a
NCP	–	n	n	–

Figure 27. DCB Fields That Can be Specified for Each Access Method

CMSCVT
simulates the Communication Vector Table. Location 16 contains the address of the CVT control section.

CMSCB
is allocated from system free storage whenever a FILEDEF command or an OPEN (SVC19) is issued for a data set. The CMS Control Block (CMSCB) consists of a File Control Block (FCB) for the data file, and partial simulation of the Job File Control Block (JFCB), Input/Output Block (IOB), and Data Extent Block (DEB).

The Data Control Block (DCB) and the Data Event Control Block (DECB) are used by the access method simulation routines of CMS.

The GET and PUT macros are not supported for use with spanned records. READ and WRITE are supported for spanned records, provided the filemode number is 4, and the data set is Physical Sequential (BSAM) format.

GET (QSAM)
All the QSAM options of GET are supported. Substitute mode is handled the same as move mode. If the DCBRECFM is FB, the filemode number is 4, and the last block is a short block, an EOF indicator (X'61FFFF61') must be present in the last block after the last record.

GET (QISAM)
QISAM is not supported in CMS.

PUT (QSAM)
All the QSAM options of PUT are supported. Substitute mode is handled the same as move mode. If the DCBRECFM is FB, the filemode number is 4, and the last block is a short block, an EOF indicator is written in the last block after the last record.

PUT (QISAM)
QISAM is not supported in CMS.

PUTX
PUTX support is provided only for data sets opened for QSAM-UPDATE with simple buffering.

READ/WRITE (BISAM)
BISAM is not supported in CMS.

READ/WRITE (BSAM and BPAM)
All the BSAM and BPAM options of READ and WRITE are supported except for the SE option (read backwards).

READ (Offset Read of Keyed BDAM data set)
This type of READ is not supported because it is only used for spanned records.

READ/WRITE (BDAM)
All the BDAM and BSAM (create) options of READ and WRITE are supported except for the R and RU options.

BDAM Restrictions

The four methods of accessing BDAM records are:

1. Relative Block RRR
2. Relative Track TTR
3. Relative Track and Key TTKey
4. Actual Address MBBCCHHR

The restrictions on those methods are as follows:

- Only the BDAM identifiers underlined above can be used to refer to records, since CMS files have a two-byte record identifier.
- CMS BDAM files are always created with 255 records on the first logical track, and 256 records on all other logical tracks, regardless of the block size. If BDAM methods 2, 3, or 4 are used and the RECFM is U or V, the BDAM user must either write 255 records on the first track and 256 records on every track thereafter, or he must not update the track indicator until a NO SPACE FOUND message is returned on a write. For method 3 (WRITE ADD), this message occurs when no more dummy records can be found on a WRITE

request. For methods 2 and 4, this will not occur, and the track indicator will be updated only when the record indicator reaches 256 and overflows into the track indicator.

- Two files of the same filetype, which both use keys, cannot be open at the same time. If a program that is updating keys does not close the file it is updating for some reason, such as a system failure or another IPL operation, the original keys for files that are not fixed format are saved in a temporary file with the same filetype and a filename of ¬KEYSAVE. To finish the update, run the program again.
- Once a file is created using keys, additions to the file must not be made without using keys and specifying the original length.
- The number of records in the data set extent must be specified using the FILEDEF command. The default size is 50 records.
- The minimum LRECL for a CMS BDAM file with keys is eight bytes.

READING OS DATA SETS AND DOS FILES USING OS MACROS

CMS users can read OS sequential and partitioned data sets that reside on OS disks. The CMS MOVEFILE command can be used to manipulate those data sets, and the OS QSAM, BPAM, and BSAM macros can be executed under CMS to read them.

The CMS MOVEFILE command and the same OS macros can also be used to manipulate and read DOS sequential files that reside on DOS disks. The OS macros handle the DOS data as if it were OS data.

The following OS Release 20.0 BSAM, BPAM, and QSAM macros can be used with CMS to read OS data sets and DOS files:

BLDL	ENQ	RDJFCB
BSP	FIND	READ
CHECK	GET	SYNADAF
CLOSE	NOTE	SYNADRLS
DEQ	POINT	WAIT
DEVTYPE	POST	

CMS supports the following disk formats for the OS and OS/VS sequential and partitioned access methods:

- split cylinders
- user labels
- track overflow
- alternate tracks

As in OS, the CMS support of the BSP macro produces a return code of 4 when attempting to backspace over a tape mark or when a beginning of an extent is found on an OS data set or a DOS file. If the data set or file contains split cylinders, an attempt to backspace within an extent resulting in a cylinder switch, also produces a return code of 4.

The ACCESS Command

Before CMS can read an OS data set or DOS file that resides on a non-CMS disk, you must issue the CMS ACCESS command to make the disk on which it resides available to CMS.

The format of the ACCESS command is:

```
ACCESS cuu mode[/ext]
```

You must not specify options or file identification when accessing an OS or DOS disk.

The FILEDEF Command

You issue the FILEDEF command to assign a CMS file identification to the OS data set or DOS file so that CMS can read it. The format of the FILEDEF command used for this purpose is: If you are issuing a FILEDEF for a DOS file, note that the OS program that will use the DOS file must have a DCB for it. For "ddname" in the FILEDEF command line, use the ddname in that DCB. With the DSN operand, enter the file-id of the DOS file.

Sometimes, CMS issues the FILEDEF command for you. Although the CMS MOVEFILE command, the supported CMS program product interfaces, and the CMS OPEN routine each issue a default FILEDEF, you should issue the FILEDEF command yourself to be sure the appropriate file is defined.

After you have issued the ACCESS and FILEDEF commands for an OS sequential or partitioned data set or DOS sequential file, CMS commands (such as ASSEMBLE and STATE) can refer to the OS data set or DOS file just as if it were a CMS file.

Several other CMS commands can be used with OS data sets and DOS files that do not reside on CMS disks. See the <u>VM/370: CMS Command and Macro Reference</u> for a complete description of the CMS ACCESS, FILEDEF, LISTDS, MOVEFILE, QUERY, RELEASE, and STATE commands.

For restrictions on reading OS data sets and DOS files under CMS, see the "VM/370 Restrictions" in "Part 1. Debugging with VM/370".

The CMS FILEDEF command allows you to specify the I/O device and the file characteristics to be used by a program at execution time. In conjunction with the OS simulation scheme, FILEDEF simulates the functions of the Data Definition JCL statement.

FILEDEF may be used only with programs using OS macros and functions. For example:

```
 FIledef | (ddname)( [DISK fn ft [fm]] [DSN ?         ] )
         | { nn   }(                 [A1]] [DSN q1 [q2...]] )
         | ( *    )(                                        )
         |         ( DISK [fn    ft    ] [fm]               )
         |         (      [FILE ddname ] [A1]               )
         |         (                                        )
         |         ( DUMMY                                  )
         |
         | Related Option: [MEMBER membername]
         |                 [CONCAT           ]
```

filedef file1 disk proga data a1

After issuing this command, your program referring to FILE1 would access PROGA DATA on your A-disk.

If you wished to supply data from your terminal for FILE1, you could issue the command:

filedef file1 terminal

and enter the data for your program without recompiling.

fi tapein tap2 (recfm fb lrecl 50 block 100 9track den 800)

After issuing this command, programs referring to TAPEIN will access a tape at virtual address 182. (Each tape unit in the CMS environment has a symbolic name associated with it.) The tape must have been previously attached to the virtual machine by the VM/370 operator.

The AUXPROC Option of the FILEDEF Command

The AUXPROC option can only be used by a program call to FILEDEF and not from the terminal. The CMS language interface programs use this feature for special I/O handling of certain (utility) data sets.

The AUXPROC option, followed by a fullword address of an auxiliary processing routine, allows that routine to receive control from DMSSEB before any device I/O is performed. At the completion of its processing, the auxiliary routine returns control to DMSSEB signalling whether I/O has been performed or not. If not, DMSSEB performs the appropriate device I/O.

GPR15 is used by the auxiliary processing routine to inform to DMSSEB of the action that has been or should be taken with the data block as follows:

GPR15=0 No I/O performed by AUXPROC routine; DMSSEB will perform I/O.

GPR15<0 I/O performed by AUXPROC routine and error was encountered. DMSSEB will take error action.

GPR15>0 I/O performed by AUXPROC routine with residual count in GPR15; DMSSEB returns normally.

DOS/VS SUPPORT UNDER CMS

CMS supports interactive program development for DOS/VS. This includes creating, compiling, testing, debugging, and executing commercial application programs. The DOS/VS programs can be executed in a CMS virtual machine or in a CMS Batch Facility virtual machine.

DOS/VS files and libraries can be read under CMS. VSAM data sets can be read and written under CMS.

The CMS/DOS environment (called CMS/DOS) provides many of the same facilities that are available in DOS/VS. However, CMS/DOS supports only those facilities that are supported by a single (background) partition. The DOS/VS facilities supported by CMS/DOS are:

- DOS/VS linkage editor
- Fetch support
- DOS/VS Supervisor and I/O macros
- DOS/VS Supervisor control block support
- Transient area support
- DOS/VS VSAM macros

The CMS/DOS environment is entered each time the CMS SET DOS ON command is issued. In the CMS/DOS environment, CMS supports many DOS/VS facilities. When you no longer need DOS/VS support under CMS, you issue the SET DOS OFF command and DOS/VS facilities are no longer available.

CMS/DOS can execute programs that use the sequential (SAM) and virtual storage (VSAM) access methods, and can access DOS/VS libraries.

CMS/DOS cannot execute programs that have execution-time restrictions, such as programs that use sort exits, teleprocessing access methods or multitasking. DOS/VS COBOL, DOS PL/I, and assembler language programs are executable under CMS/DOS.

All of the CP and CMS online debugging and testing facilities (such as the CP ADSTOP and STORE commands and the CMS DEBUG environment) are supported in the CMS/DOS environment. Also, CP disk error recording and recovery is supported in CMS/DOS.

With its support of a CMS/DOS environment, CMS becomes an important tool for DOS/VS application program development. Because CMS/DOS was designed as a DOS/VS program development tool, it assumes that a DOS/VS system exists, and uses it. The following sections describe what is supported, and what is not.

CMS SUPPORT FOR OS AND DOS VSAM FUNCTIONS

CMS supports interactive program development for OS and DOS programs using VSAM. CMS supports VSAM for OS programs written in VS BASIC, OS/VS COBOL, or OS PL/I programming languages; or DOS programs written in DOS/VS COBOL or DOS PL/I programming languages. CMS does not support VSAM for OS or DOS assembler language programs.

CMS also supports Access Method Services to manipulate OS and DOS VSAM and SAM data sets.

Under CMS, VSAM data sets can span up to nine DASD volumes. CMS does not support VSAM data set sharing; however, CMS already supports the sharing of minidisks or full pack minidisks.

VSAM data sets created in CMS are not in the CMS file format. Therefore, CMS commands currently used to manipulate CMS files cannot be used for VSAM data sets which are read or written in CMS. A VSAM data set created in CMS has a file format that is compatible with OS and DOS VSAM data sets. Thus a VSAM data set created in CMS can later be read or updated by OS or DOS.

Because VSAM data sets in CMS are not a part of the CMS file system, CMS file size, record length, and minidisk size restrictions do not apply. The VSAM data sets are manipulated with Access Method Services programs executed under CMS, instead of with the CMS file system commands. Also, all VSAM minidisks and full packs used in CMS must be initialized with the IBCDASDI program; the CMS FORMAT command must not be used.

CMS supports VSAM control blocks with the GENCB, MODCB, TESTCB, and SHOWCB macros.

In its support of VSAM data sets, CMS uses RPS (rotational position sensing) wherever possible. CMS does not use RPS for 2314/2319 devices, or for 3340 devices that do not have the feature.

Hardware devices Supported

Because CMS support of VSAM data sets is based on DOS/VS VSAM and DOS/VS Access Method Services, only disks supported by DOS/VS can be used for VSAM data sets in CMS. These disks are:

- IBM 2314 Direct Access Storage Facility
- IBM 2319 Disk Storage
- IBM 3330 Disk Storage, Models 1 and 2
- IBM 3330 Disk Storage, Model 11 only as a Model 1 or 2
- IBM 3340 Direct Access Storage Facility
- IBM 3344 Direct Access Storage
- IBM 3350 Direct Access Storage, only in 3330 Model 1 compatibility mode

RSCS INTRODUCTION

The introduction provides the following information:

- A brief description of the Remote Spooling Communications Subsystem (RSCS) external structure and the commands used to control the system.
- An overview of the RSCS control program, that is, of the RSCS supervisor and RSCS tasks.
- Descriptions of the nonprogrammable terminal (NPT) and spool MULTI-LEAVING[1] (SML) line drivers.
- Brief descriptions of major RSCS data areas and storage requirements.
- Detailed information about RSCS supervisor functions, such as synchronizing and dispatching tasks, task-to-task communications, I/O methods, and how RSCS network links are manipulated.

REMOTE SPOOLING COMMUNICATIONS SUBSYSTEM: OVERVIEW

The VM/370 Remote Spooling Communications Subsystem (RSCS) is the VM/370 component that provides for the transmission of files across a teleprocessing network controlled by the VM/370 computer. Using RSCS, virtual machine users can transmit files to remote stations. (Remote stations are I/O configurations attached to the VM/370 computer by communications lines.) Also, users at remote stations can transmit files to VM/370 virtual machines and to other remote stations using RSCS.

RSCS resides in a virtual machine dedicated to remote spooling. Using the RSCS command language, the RSCS operator manages the telecommunications facilities for the installation.

Operators at remote stations can manage their own configurations using a subset of the command language. Commands issued from remote stations can be entered either at a terminal or from a card reader.

You can find detailed descriptions of RSCS functions in the publication _VM/370_: _Remote_

----

[1]Trademark of IBM

Spooling Communications Subsystem (RSCS) User's Guide.

THE RSCS VIRTUAL MACHINE AND THE VM/370 CONTROL PROGRAM (CP)

Like the other VM/370 virtual machines, the RSCS virtual machine runs under the control of CP. In extending the VM/370 spooling system capability to include spooling to remote stations, RSCS interacts with the CP spooling system. Therefore, some of the information in this publication requires a knowledge of that area of CP.

The RSCS virtual machine consists of the virtual machine operator console, an RSCS system disk, and virtual telecommunications lines. During system generation, a virtual card reader is defined for the RSCS virtual machine, but this reader does not exist in the CP directory entry for the RSCS virtual machine.

Virtual printers, card punches, and readers are defined dynamically as they are needed. For example, when a file from a remote station is transmitted to RSCS, a virtual punch is defined to accept the file. Similarly, virtual readers are defined when RSCS receives a file to transmit. RSCS virtual storage also dumps onto a virtual printer when abnormal termination of the system occurs. Figure 28 shows the configuration of an RSCS virtual machine.

The minimum virtual storage required to run RSCS is 512K.

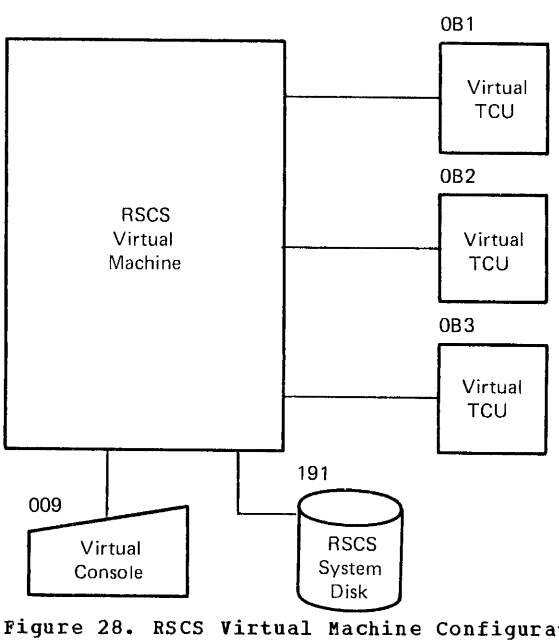


Figure 28. RSCS Virtual Machine Configuration

LOCATIONS AND LINKS

At a local installation there are a number of transmission paths to remote stations. A unique location identifier (locid) is assigned to each of these remote stations.

For each transmission path (nonswitched line) or potential transmission path (switched line), a link must be defined at the local VM/370 installation. Each such link is given a name (linkid) that defines the location identifier of the remote station to which the transmission path leads. This link can be defined either at system generation or by means of the DEFINE command.

REMOTE STATIONS

Remote stations are configurations of I/O devices attached to the VM/370 computer by binary synchronous (BSC) switched or nonswitched lines. Two types of remote stations are supported by RSCS: programmable remote stations and nonprogrammable remote stations.

Programmable Remote Stations

Programmable remote stations, such as the IBM System/3 and System/370, are IBM processing systems with attached binary synchronous communications adapters. These systems must be programmed to provide the MULTI-LEAVING line protocol necessary for their devices to function as remote stations. This programming support is provided by a remote terminal processor (RTP) program generated according to HASP workstation protocol and tailored to the system's hardware configuration. Certain programmable remote stations like the System/3 can only be programmed to function as remote terminals. Others, like the System/360 and System/370, can function either as remote terminals or as host batch systems using RSCS as a remote job entry workstation. Both of these types of remote stations are managed by the spool MULTI-LEAVING (SML) line driver of RSCS.

Nonprogrammable Remote Stations

Nonprogrammable remote stations are I/O configurations that cannot be programmed, but are hard-wired to provide the line protocol necessary for them to function as remote stations. They can receive, read, print, punch, and send files. An example of a nonprogrammable remote station is a 2780 Data Transmission Terminal. Nonprogrammable remote stations are managed by the NPT (Nonprogrammable Terminal) RSCS line driver.

The types of devices supported for all types of remote stations, programmable and nonprogrammable, are listed in the *VM/370: Remote Spooling Communications Subsystem (RSCS) User's Guide*.

NETWORK CONTROL: RSCS AND VM/370 COMMANDS

Both RSCS and VM/370 commands are used to control RSCS. The RSCS commands are used to control the RSCS network; VM/370 CP and CMS commands are used by virtual machine users who use the RSCS network.

RSCS COMMANDS

To manipulate the file being transmitted across the network and to communicate with the various network users, the RSCS control program provides a command language. Figure 29 is a list of RSCS commands and the functions they perform. You can find detailed descriptions of these commands in the publication *VM/370: Remote Spooling Communications Subsystem (RSCS) User's Guide*.

The operator may enter RSCS commands described in Figure 29 at the RSCS virtual machine console. A subset of the RSCS command language may be entered by operators of remote stations.

VM/370 CP AND CMS COMMANDS FOR RSCS

The VM/370 CP TAG and SPOOL commands specify a device to be spooled and to associate a destination location identifier (locid) with that device. SPOOL directs the file to the RSCS virtual machine. The CP CLOSE command or the CMS PRINT or PUNCH commands close the file and transfer it to the RSCS virtual machine.

Data specified by the CP TAG command controls processing of files transmitted across the RSCS network. When a VM/370 user creates a file to be transmitted to a remote station via RSCS, the TAG command text operand takes the following format:

linkid [userid] [priority]

where:

linkid — is the location identifier of the link on which the file is to be transmitted.

userid — is the remote virtual machine that is to receive the file.

priority — is the requested transmission priority (a decimal number 0-99, default 99). The lower numbers have higher priorities.

Also, the CP SPOOL command directs files to the RSCS virtual machine. See the publication For details on how to use the CP TAG and SPOOL commands to control RSCS network functions, see the *VM/370: Remote Spooling Communications Subsystem (RSCS) User's Guide*.

Command Name	Function
BACKSPAC	Restarts or repositions in a backward direction the file currently being transmitted.
CHANGE	Alters one or more attributes of a file owned by RSCS.
CMD	Controls certain functions performed by a remote system, or controls the logging of I/O activity on a specified link.
DEFINE	Temporarily adds a new link definition to the RSCS link table or temporarily redefines an existing link.
DELETE	Temporarily deletes a link definition from the RSCS link table.
DISCONN	Places RSCS in disconnect mode and optionally directs output to another virtual machine.
DRAIN	Deactivates an active communication link.
FLUSH	Discontinues processing the current file on the specified link.
FREE	Resumes transmission on a communication link previously in HOLD status.
FWDSPACE	Repositions the file currently being transmitted in a forward direction.
HOLD	Suspends file transmission on an active link without deactivating the line.
MSG	Sends a message to a local or remote station.
ORDER	Reorders files enqueued on a specific link.
PURGE	Removes all or specified files from a link.
QUERY	Requests system information for a link, a file, or for the system in general.
START	Activates a specified communication link.
TRACE	Monitors line activity on a specified link.

Figure 29. RSCS Commands and Functions

CP Instructions Used by the RSCS Control Program

When RSCS handles files being transmitted across the network, the RSCS control program (line driver tasks) issues CP DIAGNOSE instructions.

The DIAGNOSE instruction is the method of communication between a virtual machine and CP. In VM/370, the machine-coded format for the DIAGNOSE instruction is:

0–7	8–11	12–15	16–31
83	rx	ry	Code

Content	Explanation
83	DIAGNOSE operation code
rx	User-specified register number
ry	User-specified register number
Code	Hexadecimal value that selects a particular CP function.

Figure 30 lists the DIAGNOSE function codes used by RSCS, the functions of those codes, and the RSCS modules from which they are issued.

DIAGNOSE Code	Function	Issued by Module(s)
0008	Executes a CP command.	DMTAXS DMTREX DMTCMX DMTMGX DMTSML DMTNPT
000C	Gets the current time and date.	DMTSML DMTNPT
0014	Manipulates input spool files.	DMTAXS DMTSML DMTNPT
0020	Performs general I/O without interrupt.	DMTINI
0024	Determines virtual device type information.	DMTREX DMTLAX DMTSML
005C	Edits error messages.	DMTREX

Figure 30. VM/370 DIAGNOSE Instructions Issued by the RSCS Program

THE RSCS CONTROL PROGRAM

RSCS is a control program composed of a multitasking supervisor and multiple tasks, which are controlled by the supervisor.

The supervisor provides only those functions that cannot be consistently provided by the tasks themselves; that is, the supervisor provides only the support necessary to control and coordinate the execution of the tasks.

In RSCS, a task is a single program or set of subprograms that can run concurrently and autonomously with other such programs and subprograms, and which uses control functions provided by the Supervisor.

There are two types of tasks: system service tasks and line driver tasks. The system service tasks are those that provide the system support functions for the supervisor and for other tasks. The line driver tasks are those that manage the transmission paths to remote stations and that interact between the remote stations and the system service tasks and the Supervisor. Each line driver task manages the transmission of files to and from a single remote station.

Figure 56 in section 2 shows the communications paths between the supervisor, system service tasks, line driver tasks, remote stations, and VM/370 virtual machines.

THE RSCS SUPERVISOR

The RSCS supervisor is composed of a set of service routines that provide functions for the tasks that run under them. These service routines may be called by any task. In general, they provide four kinds of services:

- Task management
- I/O management
- Interrupt handling
- Virtual storage management

TASK MANAGEMENT

The task management service routines provide three kinds of services: task execution control, task synchronization, and task-to-task communication.

Task execution control includes initiating and terminating tasks. In general, the only task to request these services is the REX system control task, which is described below. Task execution control also includes the dispatcher, DMTDSP, which activates task execution as soon as that task is initiated and while the task is active.

Task synchronization comprises a mechanism by which tasks are made ready or not ready for execution. When a task requests the services of another task, the requestor task may suspend its execution while the request is being processed. The synchronization mechanism that accomplishes this consists of two routines, DMTWAT and DMTPST. DMTWAT causes the requestor task to temporarily halt execution. DMTPST causes a temporarily-halted task to resume execution. For more information on task synchronization refer to the section "Task Synchronization".

There are two types of task-to-task communications: (1) the DMTSIG routine (ALERT) and (2) the DMTGIV and DMTAKE routines (GIVE/TAKE).

The DMTSIG routine allows a task to immediately interrupt another task to pass it information. The interrupted task must have an asynchronous exit routine defined to handle the interruption. Functionally, DMTSIG performs a function analagous to an SVC instruction.

The DMTGIV and DMTAKE routines allow tasks to exchange information buffers with other tasks. The GIVE/TAKE function provides the means for organized enqueuing and delivery of requests for services or information from one task to another.

For more information on task-to-task communications, refer to the section "Task-to-Task Communications" in this section.

I/O MANAGEMENT

I/O management for tasks consists of the following functions:

- Handling requests for I/O operations
- Handling I/O interrupts
- Starting an I/O operation
- Completing an I/O request

Whenever a task requests the services of the I/O manager, that task builds an I/O request table to be passed to the I/O manager. This table consists of the following information:

- A synchronization lock for signalling I/O completion
- The address of the device on which the I/O operation is to take place
- The number of SENSE bytes to be returned, when applicable
- The address of the channel program to be executed

The following information is returned to the task by the I/O manager, in the I/O request table:

- The condition code for the SIO issued for the I/O operation
- The composite CSW
- The SENSE bytes returned by the operation (if any)

Using the information in this table, the I/O manager enqueues the request on the specified subchannel, starts the I/O operation, assembles the return information in the requestor's I/O request table, and posts the synchronization lock in the I/O request table signalling that the I/O operation is complete.

INTERRUPTION HANDLING

Supervisor service routines handle three kinds of interruptions: external interruptions, SVC interruptions, and I/O interruptions.

In RSCS, supervisor routines use the SVC (SUPERVISOR CALL) to suspend the execution or dispatching of a task when that supervisor routine received control. On an SVC interruption in RSCS, DMTSVC is entered. DMTSVC saves the status of the executing task and passes control to the calling supervisor routine in supervisor execution mode.

RSCS handles external interruptions from tasks by searching for asynchronous exit requests supplied by tasks. When a request with a code matching the external interruption code is found, its asynchronous exit is taken; otherwise, the external interruption is ignored.

I/O interruptions are handled by the RSCS I/O manager. When an active I/O request causes an I/O interruption, the status of the I/O request is updated to reflect the new information. Otherwise, a search is made for an asynchronous exit request for the interrupting device. When one is found, the asynchronous exit is taken. Otherwise, the interruption is ignored.

VIRTUAL STORAGE MANAGEMENT

The supervisor virtual storage service routine DMTSTO handles requests by tasks for main storage. When a task requests main storage, DMTSTO reserves page(s) of storage for it. Main storage is freed directly by task programs.

DMTQRQ manages requests for free elements of the supervisor status queue. Supervisor routines call DMTQRQ to reserve and release supervisor status queue elements.

RSCS TASK STRUCTURE

As described in the previous section, the RSCS supervisor comprises a set of routines that function together to manage RSCS system processing. The supervisor provides a base for many system programs called tasks. (These tasks are not to be confused with user-application programs.)

The RSCS system service tasks perform less generalized functions for the system than those functions performed by the supervisor. For example, the AXS system service task is designed specifically to access the VM/370 spool file system.

The supervisor identically manages all tasks in RSCS; the supervisor makes no distinction between system service tasks and line driver tasks. Figure 31 is a list of the RSCS tasks and a brief statement of the service each performs.

CREATE SYSTEM TASKS: DMTCRE

Task Name	Module Name	Function
REX	DMTREX	Handles console I/O; accepts requests for services passed by other system service tasks or line driver tasks; terminates a task; handles program check interruptions.
	DMTCRE	Creates a system service or line driver task.
	DMTCMX	Monitors processing of commands in RSCS; executes the DEFINE, DELETE, DISCONN, QUERY, and START commands.
	DMTMGX	Builds a message element and passes the element to the appropriate tasks for transmission or printing.
	DMTCOM	Performs common task functions.
AXS	DMTAXS	Communicates with the spool file system.
LAX	DMTLAX	Manages telecommunications line allocation.
Line Driver	DMTSML	Manages a telecommunications line for a programmable remote station using RTAM.
	DMTNPT	Manages a telecommunications line for a nonpro- programmable remote station terminal.

Figure 31. RSCS Tasks

The main system service task, REX, is loaded with the supervisor during RSCS initialization. The REX task, in turn, creates other tasks required by the system. DMTCRE reads these other tasks from a CMS disk by means of a CMS read access method. The task is then started as a new active task under RSCS.

PROCESS COMMANDS: DMTCMX

DMTCMX receives commands by means of either GIVE request elements passed by line driver tasks or in the form of a console input line resulting from a console read by DMTREX.

The commands DEFINE, DELETE, DISCONN, QUERY, and START (for inactive links) are executed by DMTCMX. Execution of these commands generally involves referencing and modification of system status tables (SVECTORS, TTAGQ, TLINKS, etc.).

If the command is not one that DMTCMX executes within its own code, the command line is examined for syntax errors and then passed to the appropriate task for execution. To do this, DMTCMX generates a formatted table called a command element to be passed to another active task for execution via an ALERT asynchronous exit.

The commands CHANGE, ORDER, and PURGE are executed by DMTAXS; the commands BACKSPAC, CMD, DRAIN, FLUSH, FREE, FWDSPACE, HOLD, MSG, TRACE and START (for active links) are executed by the line driver task for the specified link.

PROCESS MESSAGES: DMTMGX

DMTMGX manages distribution of all RSCS messages, which may be generated by REX or by any other RSCS task. Each message to be issued is presented to DMTMGX (via GIVE/TAKE for tasks other than REX) along with an internal routing code and an internal severity code.

Messages may be addressed to the local RSCS operator console, to the local VM/370 operator, to a local VM/370 user console, to a remote station operator, or to any combination of these destinations, by means of the routing code. The severity code is defined for each message, and is an indication of the importance of the message.

Messages for the RSCS local operator console are enqueued for output on the RSCS virtual machine console. Messages for the local VM/370 system operator and for local virtual machine consoles are issued by means of execution of a VM/370 MESSAGE command (through the DIAGNOSE interface). Messages for remote RSCS operators are presented to the line drivers for the associated links by means of the RSCS MSG command element interface. This method of message handling simplifies RSCS message routing, tracing, and recording.

TERMINATE SYSTEM TASKS AND HANDLE PROGRAM CHECKS: DMTREX

When a line driver task requests termination, a TAKE request is passed to DMTREX specifying that function. DMTREX marks the task as terminated, then searches for active I/O associated with the task. If active I/O is found, it is terminated. To ensure that system integrity is maintained during the termination of the I/O, a mechanism (at label QUIESE) is set up to handle situations in which an HIO (Halt I/O instruction) does not take effect immediately.

All RSCS program checks are handled by a routine in DMTREX. Program check diagnostic information is dumped, a message to the operator is issued, and the RSCS system status is modified, depending on the nature of the program check.

COMMUNICATE WITH THE VM/370 SPOOL FILE SYSTEM: DMTAXS

DMTAXS is responsible for the maintenance of the total RSCS interface to the VM/370 spool system. When a spool file arrives at the RSCS virtual machine, AXS receives the associated asynchronous interrupt, reads and interprets the file's VM/370 spool file block (SFBLOK) and TAG, enqueues the file for transmission as appropriate, and notifies the appropriate line driver of the new file's availability. AXS provides a GIVE/TAKE request interface to line driver tasks for spool file data input and output, and defines and detaches virtual spool I/O devices as necessary. Also, AXS provides an interface to DMTCMX for second-level command execution support.

AXS maintains a queue of a fixed number of virtual storage elements (called tag slots) that describe files currently owned by the RSCS virtual machine. To maintain RSCS integrity in a simple way when a very large number of files is enqueued on the RSCS virtual machine, the virtual storage tag queue is not extended during execution.

When a new file arrives at the RSCS virtual machine, its destination locid is examined, and it is accepted only if there is a matching linkid for which there is a free tag slot available. If the file's destination locid is not defined as a linkid, the file is purged and the originating user is notified of the action. If there is no free tag slot available for a defined linkid, the file is left "pending", and is accepted when a TAG slot becomes free. While a file is pending, it is not recognized by the RSCS command processors, and cannot be referenced through RSCS functions.

To prevent links from being totally locked out by an exhausted (and stagnant) virtual storage tag queue, a minimum number of tag slots is reserved for each link. This guarantees that a minimum number of files is accepted for each associated link. The number of reserved slots is defined during system generation or in the DEFINE command for each link to be defined in RSCS. The appropriate number of slots to be reserved for each link may depend on the expected file traffic, the link's line speed, the expected time the link is to be active, and the desired level of service to be provided to the link. This number for each link may be arrived at through actual operational experience in each location.

MANAGE TELECOMMUNICATION LINE ALLOCATION: DMTLAX

DMTLAX is responsible for line port resource allocation to line driver tasks. DMTLAX allocates available switched ports (when a link is activated without a specified line address) through an ALERT request interface. When a line port is specifically requested (by virtual address), DMTLAX checks the device for validity as a line port.

LINE DRIVER TASKS: DMTNPT AND DMTSML

As part of the link activation process, REX (module DMTCRE) loads and starts a line driver task to service the remote location.

The general functions of line driver tasks are:

- Manage I/O on the BSC line
- Manage transmission of spool file data via a GIVE/TAKE request to the AXS task
- Provide GIVE/TAKE requests to the REX task command module (DMTCMX)

The precise functional requirements vary from line driver to line driver, depending on the type of remote station the line driver supports.

Each line driver is responsible for maintenance of its link status and line activity (TRACE) records in the RSCS system status tables.

Two line drivers are provided, one to support remote 2770, 2780, 3770 (in 2770 mode), and 3780 terminals, and another to interface to remote HASP- and ASP-type systems or work stations.

THE SML LINE DRIVER PROGRAM

The SML line driver program is composed of four general types of routines:

- Processors, which are routines that execute the functions required by the HOST and RJE processing modes.
- An input/output routine that accepts and transmits data on the BSC line.
- A function selector routine that dispatches one of the processors when a request for services is received.
- Buffer blocking and deblocking routines.

The SML line driver supports programmable remote stations (in both HOST and RJE modes) for HASP- and ASP-type systems. HOST mode is that processing mode in which a remote station may submit jobs to VM/370 and receive print and punch output from VM/370. RJE mode is that processing mode in which VM/370 may send jobs to a remote batch system for processing and receive print and punch output from the remote batch system.

Figure 32 shows the types of data flowing to and from RSCS via the SML line driver program.

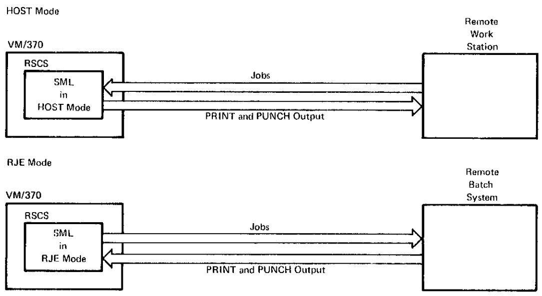


Figure 32. Data Flow Between RSCS and Remote Stations via the SML Line Driver

SML PROCESSORS

To support the HOST and RJE processing modes, the SML program provides seven "processors," or routines, that handle the seven functions required to support the two processing modes. Figure 33 is a list of the SML processors, the processing modes they support, and a brief statement of their function.

Command Processing

When a command is transmitted from a remote station to RSCS, SML receives the command and coordinates processing of the command with supervisor routines and the REX task command module DMTCMX.

The SML processor, $WRTN1, processes a command request from a remote station by passing a command request element to the REX task (module DMTCMX) via a GIVE request. DMTCMX then determines whether the command should be executed by DMTCMX, DMTAXS, or by the line driver. If the command is to be executed by the line driver, it is passed back to SML via an ALERT request. The SML routine CMDPROC then executes the command.

THE SML LINE I/O HANDLER ROUTINE: COMSUP

The SML line I/O handler routine, COMSUP, controls communications on the BSC line for SML. This routine receives data from the BSC line and passes the data to the deblocker routine ($TPGET). COMSUP also sends data (which has been blocked by the blocker routine, $TPPUT) to a remote station. COMSUP is also responsible for acknowledging receipt of data over the line using the standard BSC line control characters.

THE SML FUNCTION SELECTOR ROUTINE: $START

The $START routine is entered when SML is required (by either a remote station or a virtual machine) to perform a function. The purpose of this routine is to select a function to execute. The routine performs this function by using a commutator table, a list of synch locks, and task control tables.

The SML commutator table is a branch table consisting of branch (B) and no-operation (NOP) instructions. The targets of the branch instructions are the seven processor routines, each of which performs a specific function. When the service of a processor is not required, the Commutator Table entry for that processor is a NOP instruction. When the function of the processor is required, the NOP instruction in the commutator table entry for that processor is replaced with a B instruction, thereby opening a gate in the commutator table.

The $START routine cycles through the commutator table, falling through any NOP instructions and taking any branches. Control is passed in this way to any processor whose gate in the commutator table is open.

When the processor completes the function requested, it closes its gate in the commutator table by replacing the B instructions with a NOP instruction. $START continues cycling through the commutator table taking any open branches.

When the bottom of the commutator table is reached, $START tests a series of synch locks to see if any have been posted, signifying a request for an SML function. If any synch locks are posted, $START opens the commutator table gate for the requested processor and goes to the top of the commutator table to start cycling through it again.

If the bottom of the commutator table is reached and there are no posted synch locks, SML discontinues processing by issuing a wait request via a call to the supervisor module DMTWAT, waiting on a list of the synch locks. When any of the synch locks is posted, $START receives control, opens the appropriate gate, and starts cycling through the commutator table.

The task control table (TCT) is a DSECT defining data required by each of the processors. There is a TCT for each of the processors. Also, contained within the TCT is a branch instruction to the appropriate processor.

BLOCK AND DEBLOCK SML TELEPROCESSING BUFFERS: $TPPUT AND $TPGET

Data received over the BSC line is placed in a teleprocessing (TP) buffer. The size of TP buffers is specified by a START command parameter and can be up to 1024 bytes.

Data contained in TP buffers is deblocked into tanks, which are unit buffers of a specific

Processor	Mode	Function
$CRTN1	HOST/RJE	Processes the following MULTI-LEAVING control records: permission to transmit, request to transmit, and SIGNON control records.
$PRTN1	RJE	Processes print file records received from remote stations and passes them to the VM/370 spool system.
$URTN1	RJE	Processes punch file records received from remote stations and passes them to the VM/370 spool system.
$JRTN1	HOST	Processes job file records received from the remote station and passes them to the VM/370 spool system.
$WRTN1	HOST/RJE	In HOST mode, passes command request elements, via DMTMGX, to DMTCMX for processing. In RJE mode, passes message request elements to the RSCS operator's console.
$RRTN1	HOST/RJE	Receives records from the VM/370 spool system for transmission to remote stations.
CMDPROC		Executes local commands passed by DMTCMX, and passes messages and commands to remote stations.

Figure 33. SML Function Processors

size used to deblock the larger TP buffers. There are 15 tanks; these are allocated as they are needed by processors. The size of tanks is determined by MULTI-LEAVING control bytes.

When an SML function has been requested, the data must be either blocked for transmission (if it is data for a remote station) or deblocked for processing (if it has been received from a remote station).

$TPGET receives data from a BSC line (via the COMSUP routine) and allocates tanks to output processors as they are needed.

$TPPUT receives tanks from input processors, blocks the data in these tanks into TP buffers, and gives control to COMSUP to transmit the buffers over the line.

THE NPT LINE DRIVER PROGRAM

The NPT line driver program processes only one file at a time; it can either receive a file as input from the remote station or transmit an output file to a remote station. These two processes execute under control of a line monitor that reads and writes data over the BSC line and a function selector routine that determines whether an input or output function has been requested.

THE NPT LINE MONITOR ROUTINE: LINEIO

The NPT line monitor routine, LINEIO, controls communications on the BSC line. This routine sends and receives data over the BSC line.

When the data is received from remote stations, that data is received in the LINEINB buffer. When data is transmitted to a remote station, it is transited using the LINEBUFF buffer. The NPT buffers are a fixed size, defined by terminal type and buffer size specified on the SIGNON card.

THE NPT FUNCTION SELECTOR ROUTINE: NPTGET

When the NPT line driver program has been loaded and initialized, the NPTGET program begins a cycle in which it checks every three seconds for one of three functions to perform:

- Process a command
- Read a file from a remote station
- Write a file to a remote station

When a function is requested, a branch is taken to the appropriate routine.

NPT INPUT FILE PROCESSING

For files being received from remote stations, two processing routines are executed: PUTVRFY and PUTBLOCK. PUTVRFY reads the data contained in the input buffer (LINEINB) and verifies the BSC control characters for that data. PUTBLOCK deblocks the data in LINEINB, formats it for use by VM/370, and then writes the data to the VM/370 spool system.

NPT OUTPUT PROCESSING ROUTINES

For files being transmitted to a remote station, three processing routines are executed: MAKEBLOC, GETBLOCK, and GETVRFY.

MAKEBLOC accepts a block of data from the VM/370 spool system and passes control to GETBLOCK. GETBLOCK then builds a buffer with which to transmit the data and transmits the data to the remote station. The response received from that transmission is analyzed by GETVRFY.

MAJOR DATA AREAS

The major data areas used by RSCS are:

- SVECTORS
- RSCS supervisor queue elements
- MAINMAP
- TAREA
- LINKTABL
- TAG
- RSCS request elements
- VM/370 data areas referenced by RSCS

The data areas discussed below give a brief functional overview of each data area and its relationship to other data areas in the system. These are not meant to be a comprehensive description of the RSCS data areas. Rather, it is meant as an introduction to the types of data used by RSCS in performing its various functions.

SVECTORS: SUPERVISOR CONTROL QUEUES AND SUPERVISOR ROUTINE ADDRESSES

The SVECTORS DSECT contains:

- The PSW for the last task dispatched
- The RSCS System Save area
- The task ID and address of the task element for the last task dispatched
- Pointers to the RSCS supervisor subqueues
- Entry addresses for all supervisor service routines

This data area is updated dynamically as tasks execute and is used by RSCS to monitor the execution status of the system.

RSCS SUPERVISOR QUEUE ELEMENTS

All supervisor status information pertaining to tasks and task requests is maintained in Supervisor storage defined by the SVECTORS DSECT. There are various queues defined in this DSECT, each pertaining to a particular Supervisor function, and composed of elements of similar format. The heads of these queues are defined in a portion of SVECTORS from FREEQ through GIVEQ. The DSECTS defining the elements chained on these queues are: FREEE, TASKE, IOE, ASYNE, and GIVEE.

MAINMAP: STORAGE AVAILABLE TO RSCS PROGRAMS AND TASKS

The MAINMAP DSECT is a grid of a fixed number of bytes, each of which represents a page of virtual storage. When a task (or the Supervisor) requests storage, the byte is filled with the TASKID (generated by the Supervisor) of the requestor, thus marking the storage page as taken by that task. When a page is free, its map entry is cleared to zero by the task owning the storage.

TAREA: THE SAVE AREA FOR AN INTERRUPTED TASK

The TAREA DSECT contains the PSW at which a task is to resume execution, the contents of the task general registers when it was interrupted, and the task's request synchronization lock. This area is used to maintain the status of a task when it is interrupted by another task.

LINKTABL: LINK DESCRIPTION DATA

The LINKTABL DSECT describes control data associated with each link in the system. The control data includes such information as the linkid of the link, the task name for the link's line driver (that is, the name by which RSCS knows the task), the address of the line which is used by the link, and so on. The link table (a chain of LINKTABL DSECTs) is built during system generation and may be updated by the DEFINE, DELETE, START, and DRAIN commands.

TAG: THE RSCS FILE DESCRIPTOR

The TAG DSECT defines the attributes and status of a file being processed by RSCS. The TAG is built from information passed via the CP TAG command (or its counterpart for remote stations) and from the CP Spool File Block (SFBLOK) that describes the file.

RSCS REQUEST ELEMENTS

Request elements are data tables built by task programs when a service is to be requested by the task.

For example, when a command is processed by DMTCMX, the command line may be formatted into a command element, which gives the following types of information:

- Length of the command element
- The unique code identifying the command element

- The linkid to which command response is to be returned
- Modifiers that specify options for a given command
- A variable length buffer field containing the command line

This command element is then passed (via DMTSIG) to another task for processing.

Other types of request elements are built to process individual commands and messages, to create and terminate tasks, to process console I/O, and so on.

In many cases, elements are contained in a generalized control area used when processing a system function, for example, monitoring requests for DMTAXS module to open or close a VM/370 spool file.

VM/370 DATA AREAS REFERENCED BY RSCS

There are two VM/370 CP data areas referenced by RSCS when VM/370 spool files are processed:

- SFBLOK The VM/370 spool file block that contains control information and describes attributes of a VM/370 spool file.
- SPLINK The data block that links pages of a VM/370 spool file buffer.

RSCS STORAGE REQUIREMENTS

Figure 34 shows the storage used by the RSCS control program and how the parts of the system (the Supervisor, the tasks, and the data areas) fit together in storage.

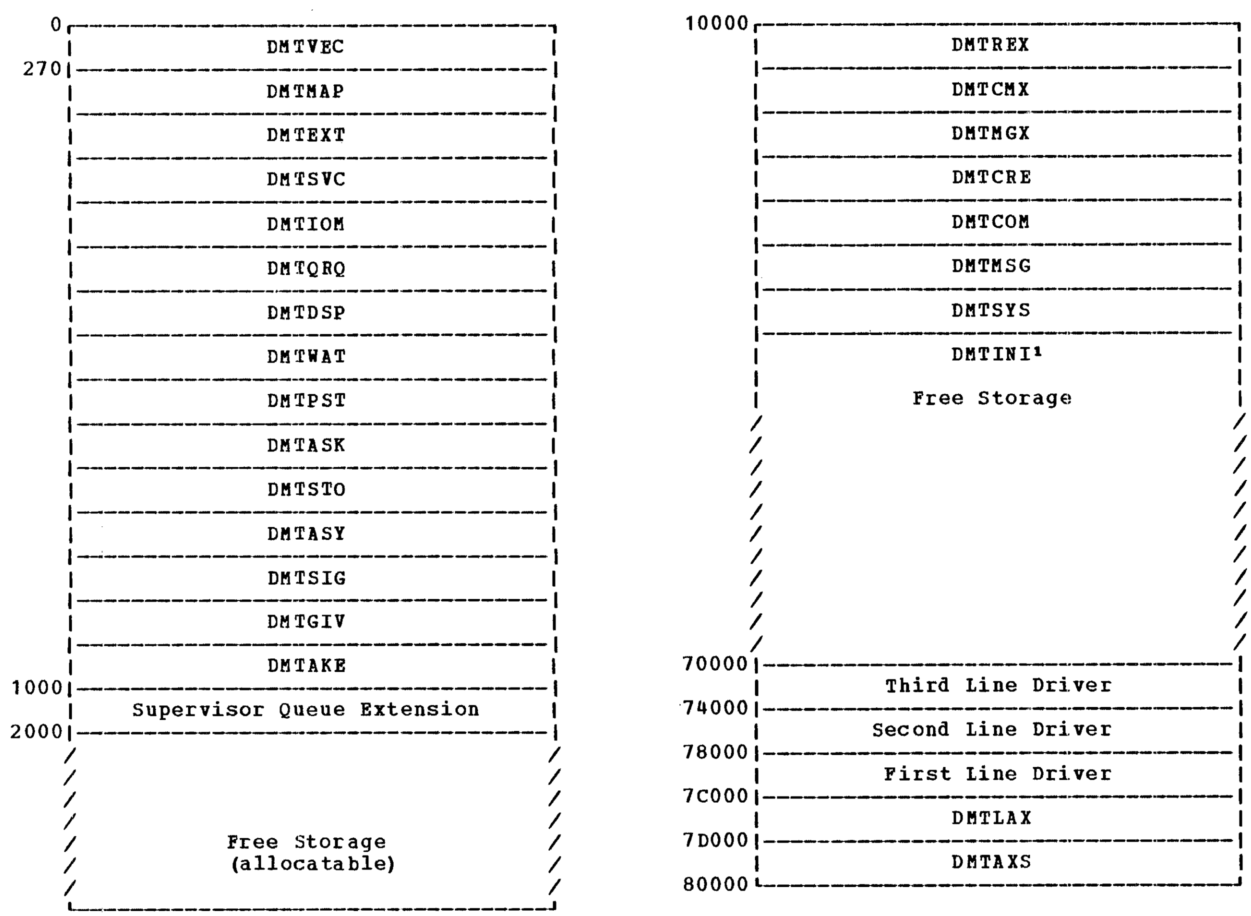


Figure 34. RSCS Storage Allocation

[1]DMTINI begins at the first page boundary following DMTSYS. After initialization its storage becomes part of free storage.

SYNCHRONIZING AND DISPATCHING TASKS

The means by which RSCS synchronizes and dispatches tasks are the WAIT/POST routines (DMTWAT and DMTPST), synchronization locks, asynchronous requests and exits, and the dispatcher routine (DMTDSP).

The WAIT/POST method of task synchronization (Supervisor modules DMTWAT and DMTPST) is used when an executing task requires the services of another task. When this situation occurs, the requesting task must suspend its execution while it waits for the requested service to be performed. In conjunction with the dispatcher, WAIT/POST allows tasks to temporarily suspend execution until they receive a signal (via the synch lock) that they can resume execution.

THE WAIT/POST ROUTINES

To suspend its execution, the requesting task calls DMTWAT, which inspects the synchronization locks RSCS uses to synchronize task execution. Completion of a service is signalled by means of a synch lock, which is set (or "posted") by DMTPST.

SYNCHRONIZATION LOCKS

Synchronization locks (or "synch locks") are fullwords contained in task save areas or control tables (such as TAREA or IOTABLE). Synch locks are also found in control areas in function selector routines such as REXCYCLE in module DMTREX.

The synch lock must be set to zero before the request for services is made. Setting the synch lock to zero prepares it for processing by the WAIT routine.

The first byte of the fullword may contain either a zero or a "post code." If the first byte is zero, the task is nondispatchable, because the requested service has not yet been performed. A post code is a code which sets to one any bit in the first byte of the synch lock. DMTPST sets such a bit to specify that a requested service has been completed.

The requesting task, that is, the caller of DMTWAT, may specify the address of a single synch lock (as in the case of a GIVE Table or an IOTABLE) or the address of a list of synch locks (as in the case of REXCYCLE), one of which must be posted by DMTPST before dispatching of the requesting task can resume. Figure 35 shows the contents of Register 1 on a call to DMTWAT.

```
 R1                                  Synch Lock
┌─────────────────┐               ┌───────────────┐
│A(Synch Lock)    │──────────────>│   │ 000000    │
└─────────────────┘               └───────────────┘

                    -- OR --

                                                        Synch Lock
 R1                                              ┌──────>│   │  0    │
┌──────────────────┐        ┌──────────────┐     │       └───────────┘
│A(List Address)   │───────>│A(Synch Lock) │─────┘        Synch Lock
└──────────────────┘        │──────────────│             ┌───────────┐
                            │A(Synch Lock) │────────────>│   │  0    │
                            │──────────────│             └───────────┘
                            /              /
                            /              /
                            │──────────────│
                            │A(Synch Lock) │─────┐        Synch Lock
                            └──────────────┘     │       ┌───────────┐
                                                 └──────>│   │  0    │
                                                         └───────────┘
```

Figure 35. Input to the DMTWAT Routine

ASYNCHRONOUS INTERRUPTIONS AND EXITS

Asynchronous interruptions result from processes external to RSCS. For example, during REX task execution, the RSCS operator may press the ATTN key on the RSCS console, thereby asynchronously interrupting execution of the REX task.

To handle asychronous interruptions, RSCS tasks contain asynchronous exit routines. These asychronous exit routines are set up during initialization without dispatching the task being requested to perform the requested service. Asynchronous exits are provided for external interruptions, for certain I/O interruptions, and for ALERT requests that occur during execution of another task.

Asynchronous exits are taken after a task calls DMTASY specifying the requested exit conditions and the entry address of the asynchronous exit routine.

DMTASY also handles external interruptions reqeusted for the clock comparator. The request element is queued on the asynchronous exit queue and processed by DMTEXT. The DMTASY clock comparator provides a time delay mechanism by using the CPU hardware clock comparator.

Asynchronous exit routines perform limited function, often enqueueing requests for further processing at a later time by dispatched tasks. When the asynchronous exit routine completes processing, it returns control to the supervisor, which then resumes dispatching tasks via a call to the dispatcher (DMTDSP).

USING ASYNCHRONOUSLY REQUESTED SERVICES: DMTWAT

Before a task can use the results of an asynchronously requested service, it must ensure that the service has been performed. To ensure that the service has been performed, the calling task signals that it is waiting for completion of a service via a call to the supervisor routine DMTWAT, specifying the synch lock associated with the requested service.

If the high-order byte of the task's synch lock is nonzero when DMTWAT inspects it, control is returned directly to the calling task. If the high-order byte of the synch lock is zero, DMTWAT marks the calling task nondispatchable (via the task's request element), stores the address of the task's request element in the low-order bytes of the synch lock, and resumes dispatching for other tasks.

POSTING A SYNCH LOCK

When the requested service is complete the REX Task signals completion by calling the POST routine (DMTPST), specifying the requesting task's associated synchronization lock. The POST routine sets the high-order byte of the synch lock to nonzero. This is referrred to as "posting" that synch lock, and indicates that the requested service is complete.

DISPATCHING IN RSCS

The supervisor functions return control to the tasks by means of the dispatcher (DMTDSP). The dispatcher scans the queue of tasks to be executed (TASKE in SVECTORS), selects the first dispatchable task element (that is, one that is not marked nondispatchable by DMTWAT), moves this task element to the end of the task queue, and restarts its execution. If no task element is marked "nondispatchable," a masked-on wait state PSW is loaded by the dispatcher.

In addition to posting a synch lock, DMTPST inspects the synch lock to determine whether DMTWAT has stored the address of a task element in that synch lock, implying that the task is nondispatchable. If this is the case, DMTPST marks the task's task element dispatchable and clears the last three bytes of the synch lock to zero.

Tasks may call DMTWAT specifying multiple synch locks. When this is the case, each synch lock is inspected and, if any synch lock is posted, task execution resumes immediately. If no synch locks are posted, the task element for the calling task is marked nondispatchable, its address is stored in each of the synchronization locks, and dispatching is resumed for other tasks.

When any synch lock in the list is posted, the task element is marked dispatachable. The dispatcher clears the low-order three bytes of each of the task's synchronization locks (pointed to in the task element before task execution is resumed).

Refer to Diagrams 1-9 and 1-10 in the Method of Operation section for details on processing by DMTWAT and DMTPST.

TASK-TO-TASK COMMUNICATIONS

There are situations when a task requires the services of another task in order to complete a function. For example, SML may require that AXS open a file for input before processing of that file can continue. RSCS task communicate with each other to request these kinds of services using two methods: ALERT task-to-task communication and GIVE/TAKE communication.

Both methods use an element, which is a table of information that describes the nature of the request. In general, these elements are referred to as request elements and ALERT elements.

ALERT TASK-TO-TASK COMMUNICATION

The ALERT method of task-to-task communication allows a task to interrupt another task to request an immediate service. The type of request is described by an ALERT element, the address of which is specified by the requesting task in a call to DMTASY.

The supervisor responds by giving control to the asynchronous exit routine defined by the request task and by passing to that task the address of the ALERT element that describes the requested service.

The requested task's (i.e., the task receiving the request) asynchronous exit routine responds immediately and may copy the ALERT element into its own storage for further processing. The receiving task's asynchronous exit routine then returns control to the supervisor, which allows the dispatched task to resume execution.

The ALERT routine (DMTSIG) also notifies another task that an asynchronous event has taken place. In this case, DMTSIG is not used with an ALERT request element.

GIVE/TAKE TASK-TO-TASK COMMUNICATION

While the ALERT method of task-to-task communication demands immediate response from the alerted task, the GIVE/TAKE method provides a means for ordered enqueueing of requests for services. These requests are handled when the servicing task is free to handle it, rather than upon immediate demand.

Request and Response Elements

Generally, request and response elements are formatted tables of information that reside in the storage of both the requesting task and the task providing the service. During task-to-task communication, these elements are passed from one task to another, containing either requests for services or responses to requests.

GIVE Tables

When a task requests services of another task via GIVE/TAKE, it builds a GIVE table in its storage. The GIVE request buffer and a GIVE response buffer. (The request and response buffers may be at the same location in storage.)

The GIVE request buffer contains a GIVE request element, which is a table of information describing the service being requested. Once the GIVE request element is built, the requesting task clears the synch lock in its address of the GIVE table to zero (in preparation for a call to DMTWAT) and specifies the address of the GIVE table in a call to DMTGIV.

Supervisor Handling of GIVE Requests

The supervisor then enqueues a supervisor GIVE element containing a pointer to the GIVE table, so that the request can be forwarded to the receiving task when that task is ready to accept the request.

Taking a GIVE Request

When the receiving task signals that it can process a GIVE request, the receiving task builds a TAKE table in its own storage. The TAKE table consists of a field to receive the task name of the requesting task and the addresses and the lengths of a TAKE request buffer and a TAKE response buffer. Functionally, these buffers complement the GIVE request and response buffers and, like the GIVE buffers, may be at the same location in storage.

Once the TAKE table is built, the receiving task specifies the address of the TAKE table in a call to DMTAKE. The supervisor then moves the GIVE request buffer (containing the GIVE request element) to the receiving task's TAKE request buffer.

Responding to a GIVE Request: DMTAKE Processing

The receiving task performs the requested service and updates the GIVE request element and places it in its TAKE response buffer. This modified GIVE request element contains information on results of request processing to be returned to the requesting task.

When all request processing is complete, the receiving task again calls DMTAKE, specifying the address of the TAKE table. The supervisor responds by immediately moving the contents of the receiving task's TAKE reponse buffer to the requesting task's GIVE response buffer, and posting the synch lock in the requesting task's GIVE table.

Multiple GIVE Requests for the Same Task

If another GIVE request addressed to the receiving task has been enqueued, it is given to the receiving task as described above, and dispatched task execution is resumed. On each call to it, DMTAKE first responds to a previously accepted GIVE request (if one exists) and then gives another modified GIVE request element back to the calling task (if one exists).

Waiting for Request Completion

The requesting task waits for request completion by specifying the address of the synch lock in its GIVE table in a call to the WAIT routine (DMTWAT).

The receiving task waits for request availability by calling DMTWAT and specifying the address of its "task request synch lock," which is located in its Task Save Area. The task request synch lock is cleared to zero by DMTAKE when no GIVE request address to the calling task remains enqueued. It is posted by

DMTGIV when such a request is enqueued as a result of DMTGIV processing for another task.

Figure 36 shows the movement of data during a GIVE/TAKE transaction.

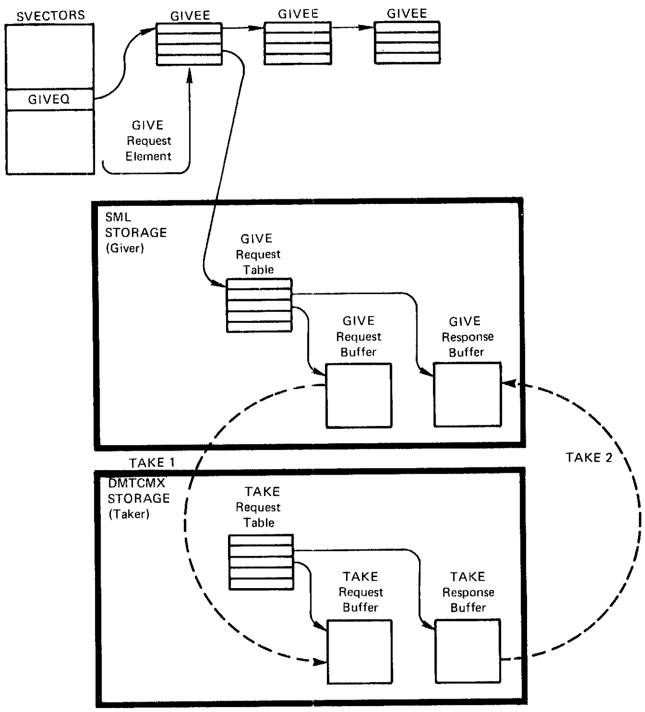


Figure 36. Movement of Data During a Typical GIVE/TAKE Transaction

INPUT/OUTPUT METHODS AND TECHNIQUES

Two data structures are created when RSCS performs an I/O operation: an I/O element and an I/O table.

The I/O table (defined by DSECT IOTABLE) is built by the requesting task and describes specific information required to perform the requested I/O operation.

The I/O element (defined by DSECT IOE) is built by the I/O request manager (DMTIOM) and consists of items of system information describing a request for an I/O operation.

I/O elements are placed on queues pointed to in SVECTORS: MPXIOQ (for multiplexer I/O requests) and SELIOQ (for Selector I/O requests). The elements in these two queues are in ascending subchannel order. Queue elements may also contain pointers to subqueues, which represent requests for use of the same nonshared subchannel. Each I/O element points to an I/O table.

Also, there is a queue of I/O asynchronous exit request elements pointed to in the SVECTORS data area. Figure 37 shows the relationships between these various data areas.

ACTIVE AND PENDING I/O QUEUES

The supervisor I/O queues (MPXIOQ and SELIOQ) include an active queue and a number of inactive or "pending" subqueues. Each element in the active I/O queue represents an I/O operation which is active on a particular nonshared I/O subchannel. The active I/O queue is ordered according to ascending numerical I/O subchannel address.

When an I/O operation is requested on an idle I/O subchannel, an I/O element representing the request is built and enqueued on the active I/O queue in its I/O subchannel's numerical address position. The I/O operation is then started.

When an I/O operation is requested on an I/O subchannel for which an I/O element is enqueued on the active I/O queue, the nonshared subchannel is busy and, therefore cannot be started immediately. In this case, an I/O element representing the request is built and enqueued on the subchannel's inactive I/O subqueue. The head of this subqueue is contained in the active I/O element enqueued on the active I/O queue.

When the nonshared subchannel's active I/O completes and the subchannel becomes available, the first element on the inactive I/O subqueue is enqueued on the active I/O queue and its I/O operation is started.

HANDLING LINK ACTIVITY: LINKTABLS AND TAGS

When the RSCS system is generated, a number of TAG slots are generated and enqueued on the free TAG queue. TAG slots are storage areas defined by the TAG DSECT; TAG slots describe the files being transmitted via RSCS; the free TAG queue comprises those TAG slots available for a given RSCS system.

The Free TAG Queue is defined in the DSECT TAGAREA, which also defines the status of TAG slots in the RSCS system. TAGAREA is pointed to by TTAGQ in SVECTORS.

HOW LINKS HANDLE FILES

Each link in RSCS is defined by a LINKTABL DSECT. The LPOINTER field of the LINKTABL DSECT points to the link's inactive TAG queue. This queue comprises those TAGs describing files that RSCS has not yet transmitted. Only one TAG per link can be active at a time.

The queue of LINKTABLs (called the link table) is pointed to by the TLINKS field in SVECTORS.

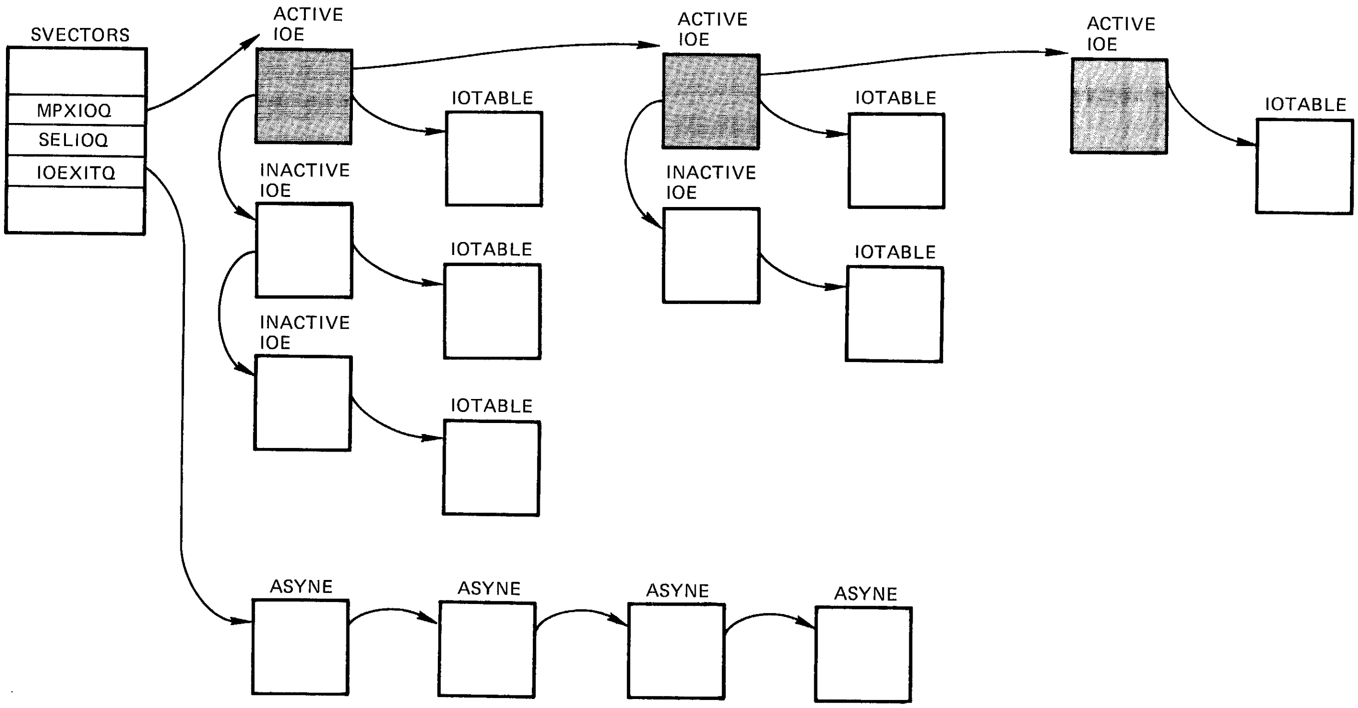


Figure 37. I/O Queues and Subqueues

TRANSMITTING VM/370 FILES TO AN RSCS LINK

When a VM/370 file is spooled to RSCS for a specific link, RSCS accepts the file and:

- Obtains a free TAG slot for the file.
- Builds a description of the file in the TAG slot.
- Enqueues the new TAG on the link's inactive TAG queue.

When transmission to the remote station begins, the file's TAG is dequeued from the inactive TAG queue and enqueued on the active input file queue (TAGACIN in TAGAREA). When transmission of the file is complete, the TAG is dequeued from the active input queue and its slot is returned to the Free TAG Queue.

PROCESSING FILES FROM REMOTE STATIONS

As in the case of VM/370 spool files, when files are received from remote stations, RSCS obtains a TAG slot and builds a description of the file in that slot. However, files from remote stations are enqueued on the active output queue (TAGACOUT in TAGAREA).

When the file is completely transmitted, its TAG is dequeued from the active output queue, closed to the VM/370 spool system, and its freed slot returned to the free TAG queue.

Figure 38 shows the relationships between the DSECTs described above.

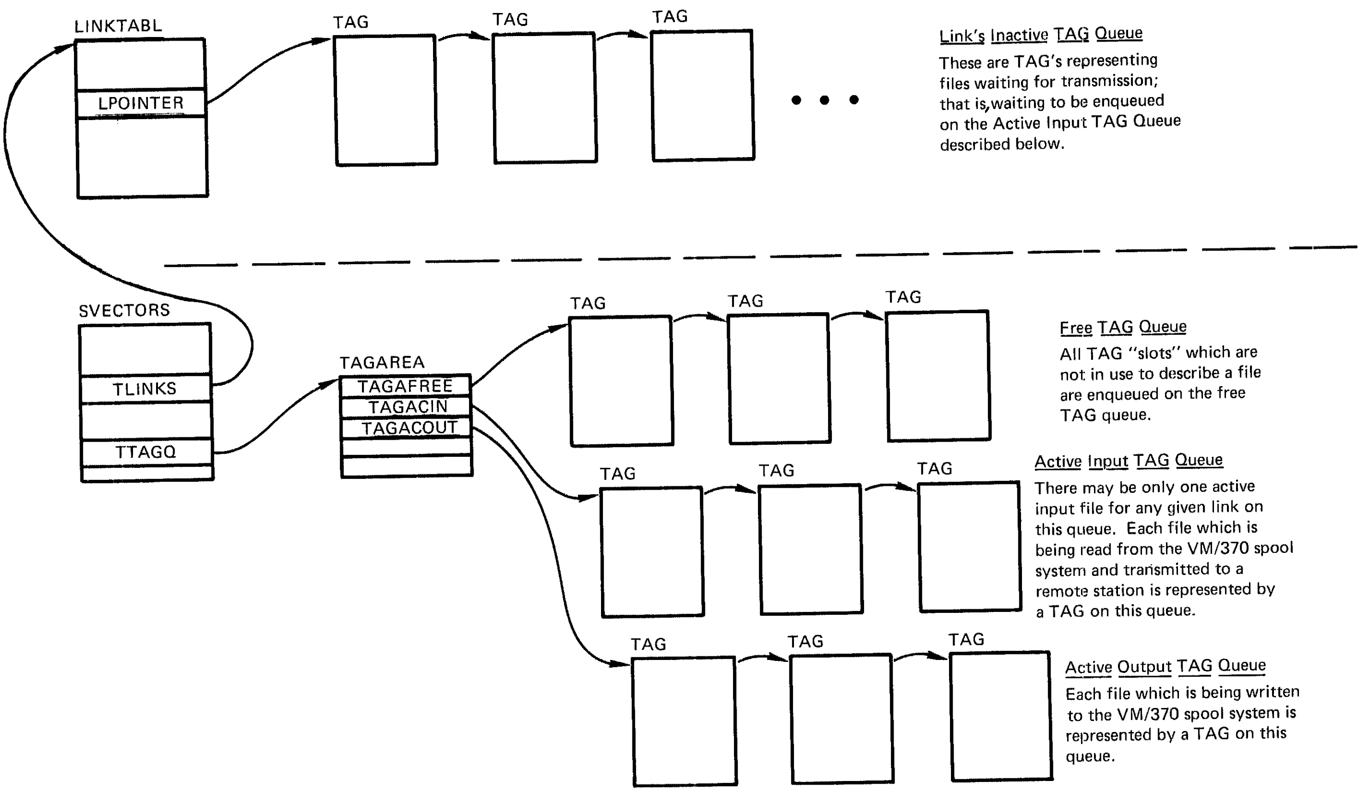


Figure 38. Chaining of Data Areas Required for File TAG Manipulation

Section 2 describes the program organization for CMS, CP, and RSCS.

CMS PROGRAM ORGANIZATION

The CMS description is in two parts. The first part contains figures showing the functional organization of CMS. The second part contains general information about internal structure of CMS programs and their interaction with one another.

CMS program organization is in two figures. Figure 39 is an overview of the functional areas of CMS. Each block is numbered and corresponds to a more detailed outline of the function found in Figure 40.

Figure 40 shows how CMS routines relate to these functional areas. The numbers on top of each detailed figure correspond to the numbers on Figure 39.

In most cases, the detailed figures contain three levels of information: the functional topic, a breakdown of logical areas within that topic, and the CMS routines that perform those logical functions.

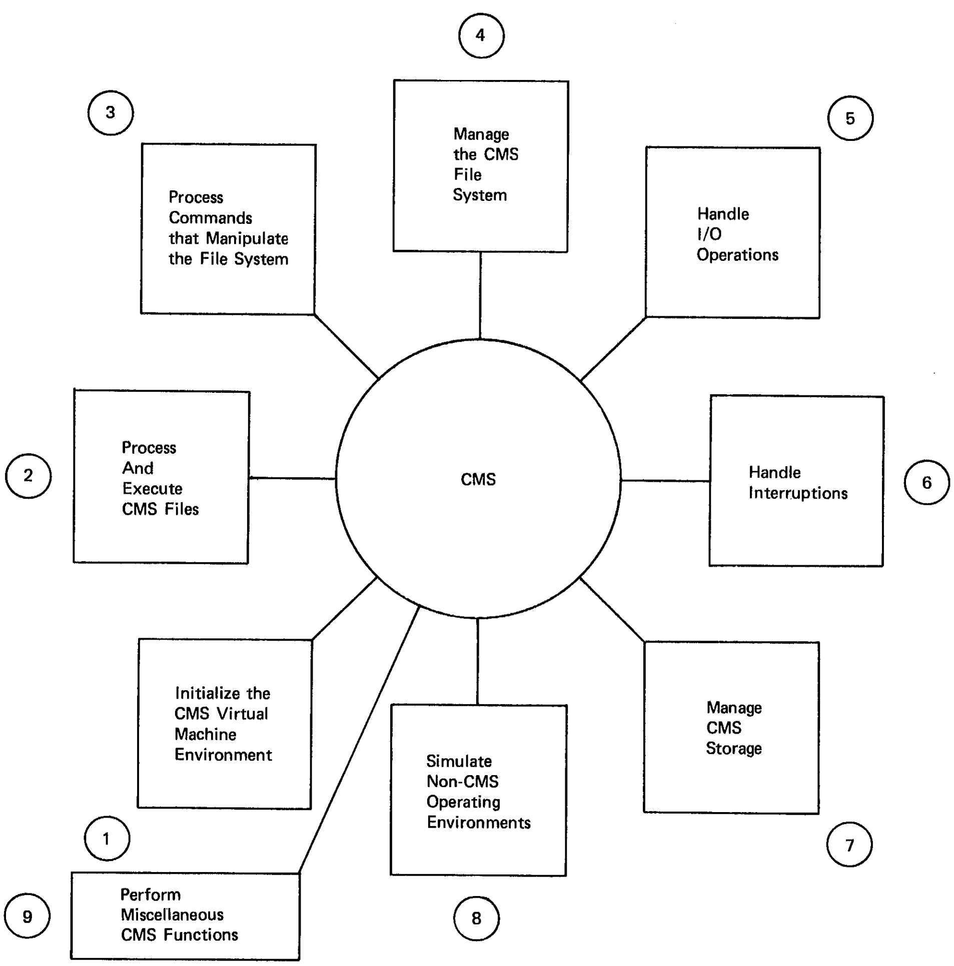


Figure 39. An Overview of the Functional Areas of CMS

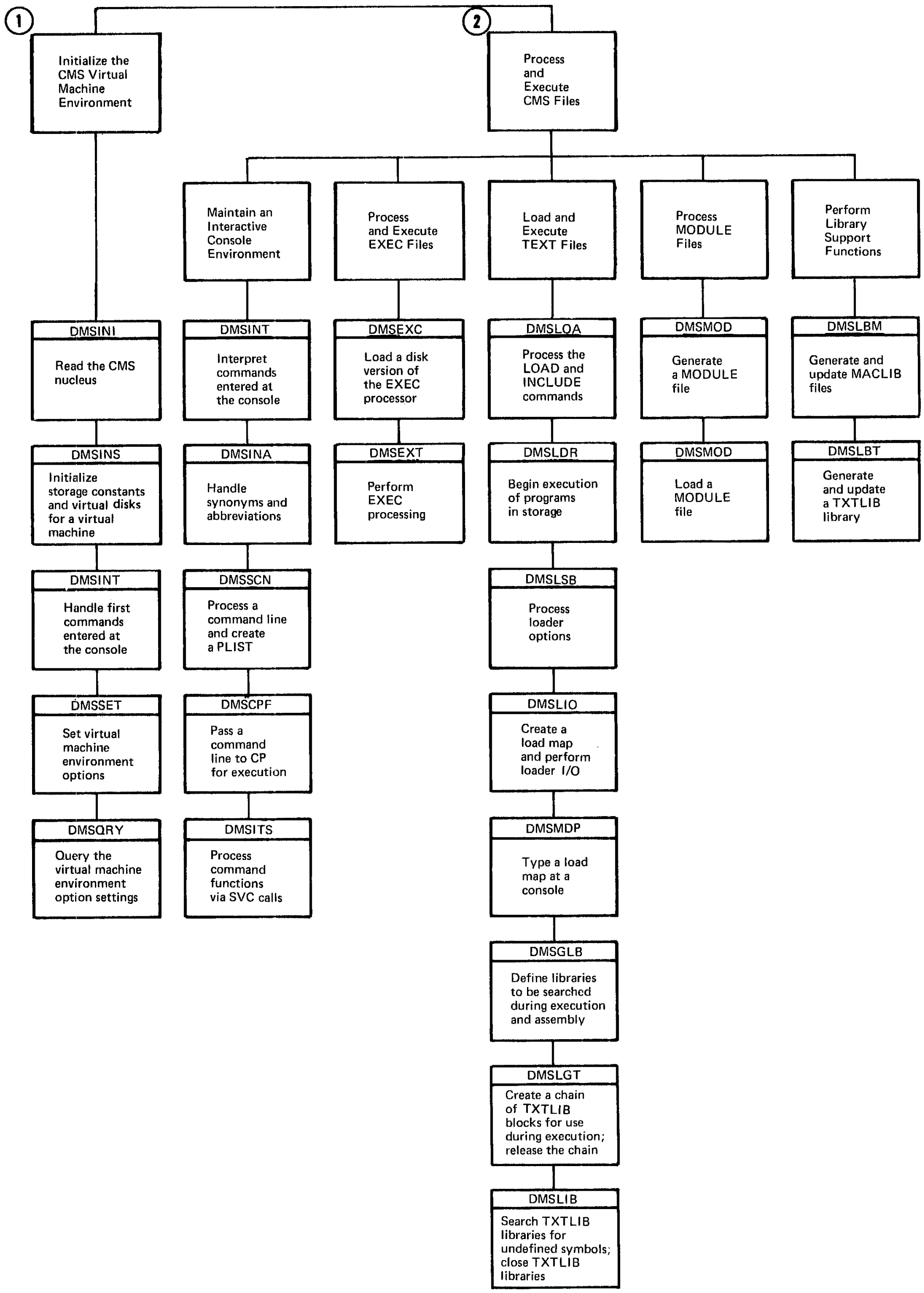


Figure 40. Details of CMS System Functions and the Routines That Perform Them (Part 1 of 4)

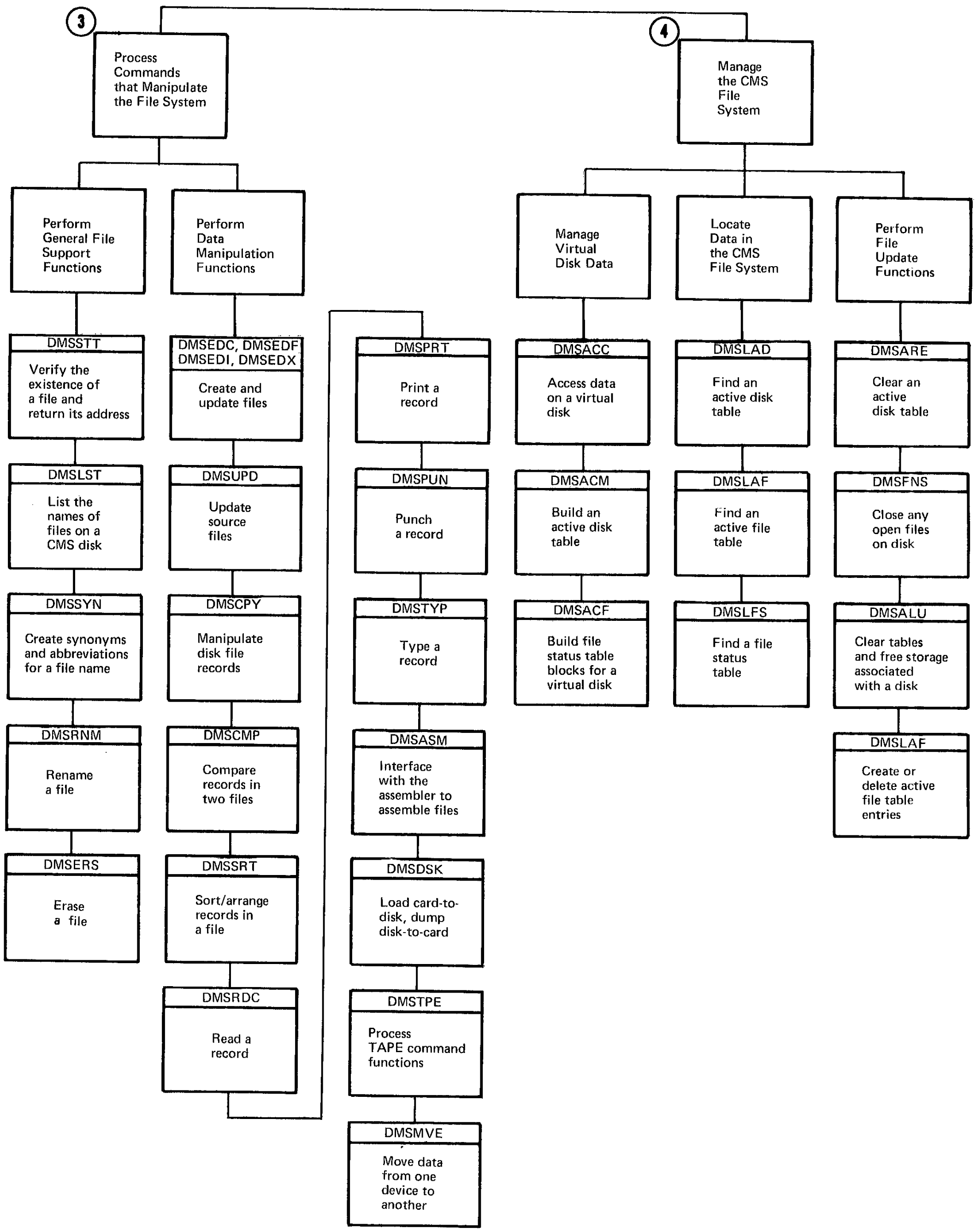


Figure 40. Details of CMS System Functions and the Routines That Perform Them (Part 2 of 4)

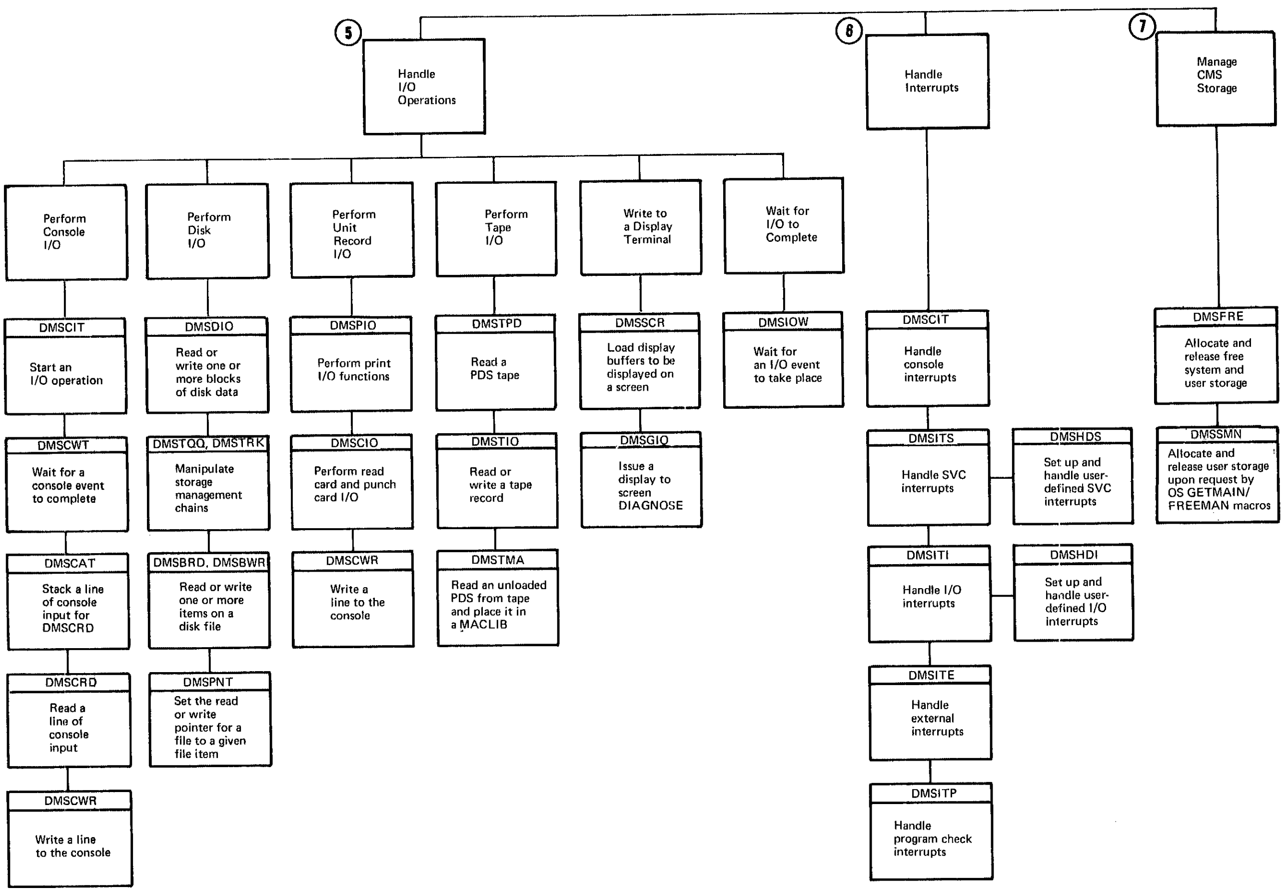


Figure 40. Details of CMS System Functions and the Routines That Perform Them (Part 3 of 4)

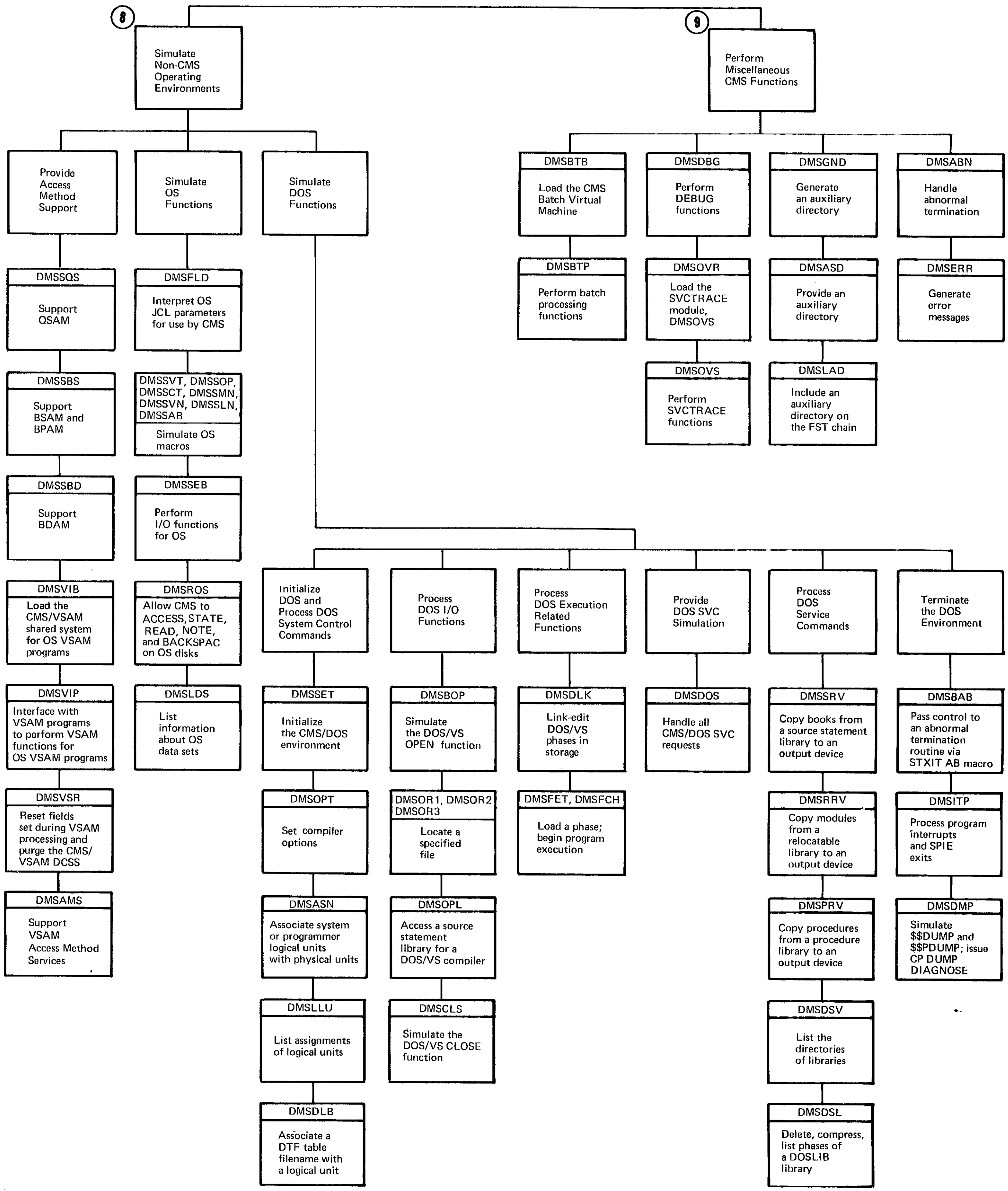


Figure 40. Details of CMS System Functions and the Routines That Perform Them (Part 4 of 4)

INTRODUCTION TO CMS

This introduction contains brief descriptions of the concepts on which CMS operates, for example, how the CMS file system is organized. This introduction also contains descriptions of logic flow for significant routines, for example, the logic executed when the CMS command handler, DMSITS, is invoked.

The following CMS information is organized into topics that correspond to the functional areas described in the previous charts.

INITIALIZE THE CMS VIRTUAL MACHINE ENVIRONMENT

There are four steps involved in initializing a CMS virtual machine:

- Processing the IPL command for a virtual card reader.
- Processing the IPL command for a disk device or a named or saved system.
- Processing the first command line entered at the CMS virtual console.
- Setting up the options for the virtual machine operating environment.

DMSINI and DMSINS are the two routines that are mainly responsible for the one-time initialization process in which the virtual card reader is initial program loaded. DMSINI also handles the IPL process when a named or saved system is loaded. The CMS command interpreter, DMSINT, processes the first line entered from the console as a special case; the processing performed by this code is a part of the initialization process. DMSSET sets up the user-specified virtual machine environment features; DMSQRY allows the user to query the status of these settings.

INITIALIZATION: LOADING A CMS VIRTUAL MACHINE FROM CARD READER

When a virtual card reader is specified by the IPL command, for example 00C, initialization processing begins. Initialization refers to the process of loading from a card reader as opposed to reading a nucleus from a cylinder of a CMS minidisk or reading a named or shared system (description follows).

IPL 00C invokes the CMS module DMSINI, which requests that the operator enter information such as the address of the DASD where the nucleus is to be written, the cylinder address where the write operation is to begin, and which version of CMS is to be written (if there is more than one to choose from).

When all questions are answered, the requested nucleus is written to the DASD. Figure 41 shows the structure of the CMS nucleus.

Once written on the DASD, a copy of the nucleus is read into virtual machine storage. One track at a time is read from the disk-resident nucleus into virtual storage. DMSINS is then invoked to initialize storage constants and to set up the disks and storage space required by this virtual machine.

DMSINS performs three general functions:

- Initializes storage constants and system tables.
- Processes IPL command line parameters (SEG= and BATCH).
- Initializes for OS SVC processing, in the case where a saved segment is not available for use in processing OS simulation requests.

Initializes Storage Contents and System Tables

DMSINS
: Saves the address of this virtual machine in NUCON.

DMSLAD
: Locates and returns the address of the ADT for this virtual machine.

DMSFRE
: Allocates free storage to be used during initialization.

DMSFRE
: Allocates all low free storage so that the system status table (SSTAT) will be built in high free storage.

DMSACM
: Reads the S-disk ADT entry and builds the SSTAT.

DMSFRE
: Releases the low free storage allocated above (to force SSTAT into high storage) so that it can be used again.

DMSINS
: Stores the address of SSTAT into ASSTAT and ADTFDA in NUCON.

DMSALU
: Sorts the entries in the SSTAT.

Processes IPL Command Line Parameters

DMSINS
: Checks for parameters BATCH and SEG=. If BATCH is specified, DMSINS sets the flag BATFLAGS. If SEG= is specified, DMSINS loops through again to read the segment name. At this point, all the parameters on the command line have been scanned.

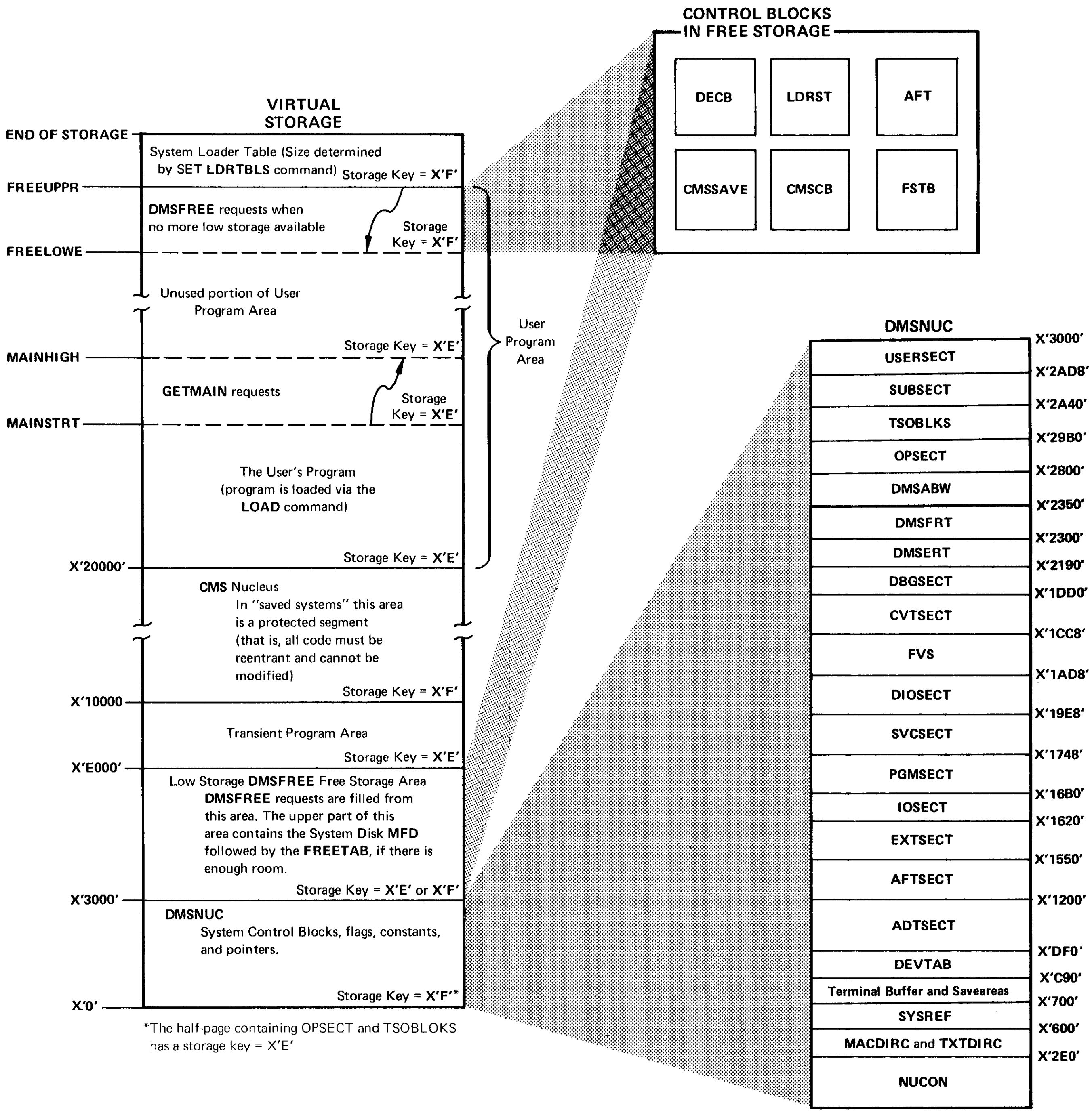


Figure 41. CMS Storage Map

If SEG= is specified, the DIAGNOSE 64 FINDSYS function is issued to determine whether the segment specified on the command line exists. If it does, the DCSSAVAL flag is temporarily set.

DMSINS
Issues DIAGNOSE 24 to obtain the device type of the console.

DMSCWR
Writes the system id message to the console.

DMSCRD
Reads the IPL command line from the console.

DMSSCN
Puts 3 the IPL command line in PLIST format.

DMSINS
If the FINDSYS DIAGNOSE validated the segment name specified on the IPL command line, DMSINS issues a DIAGNOSE 64 SAVESYS function for that segment.

DMSINS
Clears DCSSAVAL and ensures that all the parameters on the command line are valid; branches back to label INITLOOP to reprocess for the segment just saved.

DMSINS
If BATCH is specified, sets BATFLAGS and BATFLAG2 in NUCON. Saves the name of the BATCH saved system in SYSNAME in NUCON.

DMSACC
Issues ACCESS 195 A to access the batch virtual machine A-disk.

DMSINS
Issues DIAGNOSE 60 to get the size of the virtual machine; sets up enough storage for this virtual machine.

DMSINS
If the DCSSAVAL flag is set, sees if the size of the CMSSEG segment overlaps the size of the virtual machine. If this is the case, DMSINS sets the flag DCSSOVLP and continues the initialization procedure for a CMS virtual machine running without the use of the CMSSEG segment, that is, performs time-of-day processing and OS initialization.

If the CMSSEG segment can be used, DMSINS issues the DIAGNOSE 64 LOADSYS function as the final check to see if the segment is usable. If the segment is loaded successfully, it can be used whenever one of the functions contained in it is requested. Because it is not required immediately, DMSINS issues the DIAGNOSE 64 PURGESYS function to purge the segment.

If the segment cannot be successfully loaded, DMSINS turns off the DCSSAVAL flag.

Initialize OS SVC-Handling Without the Use of the CMSSEG Segment

DMSINS
Checks for the availability of CMSSEG.

DMSSTT
Finds and returns the address of DMSSVT, the CMS OS SVC-handler.

DMSFRE
Acquires enough free storage to contain DMSSVT.

DMSLOA
Loads DMSSVT.

DMSINS
Sets the flag DCSSVTLD.

DMSINS
If the BATCH virtual machine is not being IPLed, determines whether there is a PROFILE EXEC or a first command line to be handled. If so, issues SVC 202's to process these commands and passes control to DMSINT, the CMS console manager.

DMSACC
If the BATCH virtual machine is being initial program loaded, accesses the D-disk and passes control to DMSINT, the console manager.

INITIALIZING A NAMED OR SAVED SYSTEM

A named system is a copy of the nucleus that has been saved and named with the CP SAVESYS command. It is faster to IPL a named system than to IPL by disk address because CP maintains the named system in page format instead of CMS disk format. That is, the saved system is on disk in 4096-byte blocks instead of 800-byte blocks. The initialization of a saved system is also faster because the SSTAT is already built.

The shared system is a variant of the saved system. In the shared system, reentrant portions of the nucleus are placed in storage pages that are available to all users of the shared system. Each user has his own copy of nonreentrant portions of the nucleus. The shared pages are protected by CP, and may not be altered by any virtual machine.

During DMSINI processing, the virtual machine operator is asked if the nucleus must be written (via message DMSINI607R). If the operator answers no, control passes directly to DMSINS to initialize the named or saved system specified by the operator in his answer to message DMSINI606R.

HANDLE THE FIRST COMMAND LINE PASSED TO CMS

DMSINT, the CMS console manager, contains the code to handle commands stacked by module DMSINS during initialization processing. DMSINT checks for the presence of a stacked command line, and if there is one to process, processes it just as it would a command entered during a terminal session. That is, DMSINT calls the WAITREAD subroutine and issues an SVC 202 to execute the command. When first command processing completes, DMSINT receives control to handle commands entered at the console for the duration of the session.

SETTING AND QUERYING VIRTUAL MACHINE ENVIRONMENT OPTIONS

DMSSET sets up the virtual machine environment options, as outlined in the publication *VM/370: CMS Command and Macro Reference*. DMSQRY displays these settings at the user console. Both of these modules are structured and relatively easy to follow, except for some sections of DMSSET.

DMSSET: SET DOS ON (VSAM) Processing

DMSSET
: (label DOS) If a disk mode is specified on the command line, ensure that it is valid.

DMSLAD
: If the disk mode specified is valid, locates and returns the address of the disk.

DMSSET
: Issues DIAGNOSE 64 FINDSYS to locate the CMSDOS segment. If the segment is not already loaded, issues DIAGNOSE 64 LOADSYS to load it.

DMSSET
: Sets up the $$B-transient area for use by DOS routines.

DMSSET
: If SET DOS OFF has been specified, issues the DIAGNOSE 64 PURGESYS function for the CMSDOS segment and, if VSAM has been loaded, for the CMSVSAM segment.

DMSSET: SET SYSNAME Processing

DMSSET
: Determines whether the name of the CMSSEG segment is being changed.

DMSSET
: Determines whether NONSHARE is specified. If so, the segment may be loaded and kept. If NONSHARE is not specified, the segment is purged, because it is needed only on demand.

DMSSET
: Once a new name is placed in the SYSNAMES table replacing CMSSEG, the DIAGNOSE 64 FINDSYS function is issued to determine whether the new name has been entered correctly. If the FINDSYS is successful, the size of the virtual machine is compared to beginning address of the segment to determine whether the segment overlays virtual machine storage.

DMSSET
: If the segment can be used (i.e. does not overlay the virtual machine storage) the DIAGNOSE 64 LOADSYS function is performed. If the LOADSYS executes successfully, control passes to DMSINT, where the segment is purged (because it is only needed on demand).

PROCESS AND EXECUTE CMS FILES

As shown in Part 1 of Figure 40, the five general topics form the category "Process and Execute CMS Files." Two of these topics are discussed in this section: "Maintaining an Interactive Console Environment" and "Loading and Executing TEXT files."

MAINTAINING AN INTERACTIVE CONSOLE ENVIRONMENT

Two levels of information are discussed in the following section. The first level is a general discussion of how CMS maintains an interactive console environment. The second level is a more detailed discussion of the methods of operation mainly responsible for this function.

CONSOLE MANAGEMENT AND COMMAND HANDLING IN CMS

There are two major functions concerned with maintaining an interactive terminal environment for CMS: console management and command processing. The CMS module that manages the virtual machine console is DMSINT. The module responsible for command processing is DMSITS. Many CMS modules are called in support of these two functions but the modules in the following list are primarily responsible for supporting the functions:

DMSCRD
: Reads a line from the console.

DMSCWR
: Writes a line to the console.

DMSSCN
: Converts a command line to PLIST format.

DMSINA
: Converts abbreviated commands to their full names.

DMSCPF
: Passes a command line to CP for execution.

MAINTAINING AN INTERACTIVE COMMAND/RESPONSE SESSION

Three main lines of control maintain the continuity for an interactive CMS session: (1) handling of commands passed to DMSINT by the initialization module, DMSINS (2) handling of commands entered at the console during a session, and (3) handling of commands entered as subset commands. The following lists show the main logic paths for first two functions.

Execute Commands Passed via DMSINS

DMSINT
: On entry from DMSINA, processes any commands passed via the console read put on the user's console by that routine; that is processes any commands the user stacks on the line as the first read that DMSINT processes. In handling the first read, if that read is null, control passes to the main loop of the program, which is described in the following section.

DMSINM
: Get the current time.

DMSCRD
: Branch to the waitread subroutine to read a command line at the console.

DMSSCN
: Waitread then calls DMSSCN to convert the line just read into plist format. Once converted to plist format, an SVC 202 is issued (at label INIT1A) to execute the function. This cycle is repeated until all stacked commands are executed.

DMSFNS
: When command execution completes, calls DMSFNS (at label UPDAT) to close any files that may have remained open during the command processing.

DMSVSR
: Ensures that any fields set by VSAM processing are reset for CMS. Also ensures that the VSAM discontiguous shared segment is purged.

DMSINT
: Sets up an appropriate status message (CMS, CMS SUBSET, CMS/DOS, etc.).

DMSCWR
: Writes the status message to the console.

Handle Commands Entered During a CMS Terminal Session

DMSINT
: Branches (from label INLOOP2) to the waitread subroutine to read a line entered at the console.

DMSCRD
: Reads a line entered at the console (subroutine waitread).

DMSSCN
: Converts the command line to PLIST format (subroutine waitread).

DMSINT
: Determines whether the command line is a null line or a comment.

DMSLFS
: If the command line is neither a command line nor a comment, determines whether the command is an EXEC file.

DMSINA (ABBREV)
: Determines whether the command is an abbreviation and, if it is, returns its full name.

DMSITS
: Passes the command line to DMSITS via an SVC 202. DMSITS is the CMS SVC handler. For a detailed description of the SVC handler, see "Method of Operation for DMSITS."

DMSCPF
: If the command could not be executed by the SVC handler, passes the command to CP to see if CP can execute it.

DMSFNS
: On return from processing the command line (label UPDAT), closes any files that may have been opened during processing.

DMSSMN
: Resets any flags or fields that may have been set during OS processing.

DMSVSR
: Ensures that any fields set for VSAM processing are reset for CMS. Also ensures that the VSAM discontiguous shared segment is purged.

DMSINT
: When the command line has been successfully executed, builds a CMS ready message for the user (label PRNREADY).

DMSCWR
: Writes the ready message to the console.

DMSINT
: Returns control to DMSINT at label INLOOP2 to continue monitoring the CMS terminal session.

METHOD OF OPERATION FOR DMSINT

DMSINT, the console manager, maintains the continuity of operation of the CMS command environment. The main control loop of DMSINT is initiated by a call to DMSCRD to get the next command. When the command is entered, DMSINT calls DMSINM to initialize the CPU time for the new command and then puts it in standard parameter list form by calling the scan function program DMSSCN. After calling DMSSCN, DMSINT checks to see if an EXEC filetype exists with a filename of the typed-in command. (For example, if ABC was typed in, it checks to see if ABC EXEC exists.) If the EXEC file does exist, DMSINT adjusts register 1 to point to the same command as set up by DMSSCN, but preceded by CL8'EXEC', and then issues an SVC 202 to call the corresponding EXEC procedure ('ABC EXEC' in the example).

If no such EXEC file exists for the first word typed in, DMSINT makes a further check using the CMS abbreviation-check routine, DMSINA. If, for example, the first word typed in had been 'E', DMSINT looks up 'E' via the DMSINA routine. If an equivalent is found for 'E', DMSINT looks for an EXEC file with the name of the equivalent word (for example, EDIT EXEC); if such a file is found, DMSINT adjusts register 1 as described above to call EXEC and substitutes the equivalent word, EDIT, for the first word typed in. Thus, if 'E' is a valid abbreviation for 'EDIT' and the user has an EXEC file called EDIT EXEC, he invokes this when he merely types in 'E' from the terminal.

If no EXEC file is found either for the entered command name or for any equivalent found by DMSINA, DMSINT leaves the terminal command as processed by DMSSCN and then issues an SVC 202 to pass control to DMSITS which, in turn, passes control to the appropriate command program. When the command terminates execution, or if DMSITS cannot execute it, the return code is passed in register 15.

A 0 return code indicates successful completion of the command.

A positive return code indicates that the command was completed, but with an apparent error; and a negative code returned by DMSITS indicates that the typed in command could not be found or executed at all.

In the last case, DMSINT assumes that the command is a CP command and issues a DIAGNOSE instruction to pass the command line to the CP environment. If the command is not a CP command, DMSINT calls DMSCWR to type a message indicating that the command is unknown and the main control loop of DMSINT is entered at the beginning.

If the return code from DMSITS is positive or zero, DMSINT saves the return code briefly and calls module DMSAUD to update the Master File Directory (MFD) on the user's appropriate user's disk. DMSINT also frees the TXTLIB chain and releases pages of storage if required.

After updating the master file directory, DMSINT checks the return code that was passed back. If the code is zero, DMSINT types a ready message and the CPU time used by the given command. Control is passed to the beginning of the main control loop of DMSINT. If the return code is positive, an error message is typed, along with the CPU time used. The command caused the typing of an error message of the format: DMSxxxnnnt 'text' where DMSxxx is the module name, nnn is the message identification number, t is the message type, and 'text' is the message explaining the error. Control is then passed to the beginning of the main control loop.

METHOD OF OPERATION FOR DMSITS

DMSITS (INTSVC) is the CMS system SVC handling routine. Since CMS is SVC driven, the SVC interruption processor is more complex than the other interruption processors.

The general operation of DMSITS is as follows:

1. The SVC new PSW (low-storage location X'60') contains, in the address field, the address of DMSITS1. Thus, the DMSITS routine is entered whenever a supervisor call is executed.
2. DMSITS allocates a system and user save area, as described below. The user save area is a register save area used by the routine, which is invoked later as a result of the SVC call.
3. The called routine is invoked.
4. Upon return from the called routine, the save areas are deallocated.
5. Control is returned to the caller (the routine which originally made the SVC call).

The following expands upon various features of the general operation that has just been described.

Types of SVCs and Linkage Conventions

The types of SVC calls recognized by DMSITS, and the linkage conventions for each are as follows:

SVC 201: When a called routine returns control to DMSITS, the user storage key may be in the PSW. Because the called routine may also have turned on the problem bit in the PSW, the most convenient way for DMSITS to restore the system PSW is to cause another interruption, rather than to attempt the privileged Load PSW instruction. DMSITS does this by issuing SVC 201, which causes a recursive entry into DMSITS. DMSITS determines if the interruption was caused by SVC 201, and if so, determines if the SVC 201 was from within DMSITS. If both conditions are met, control returns to the instruction following the SVC 201 with a PSW that has the problem bit off and the system key restored.

SVC 202: SVC 202 is the most commonly used SVC in the CMS system. It is used for calling nucleus resident routines and for calling routines written as commands.

A typical coding sequence for an SVC 202 call is the following:

```
LA   R1,PLIST
SVC  202
DC   AL4(ERRADD)
```

Whenever SVC 202 is called, register 1 must point to a parameter list (PLIST). The format of this parameter list depends upon the actual routine or command being called, but the SVC handler examines the first 8 bytes of the list to find the name of the routine or command being called. It searches for the routine or module as described for SVC 201.

The DC AL4(address) following the SVC 202 is optional, and may be omitted if the programmer does not expect any errors to occur in the routine or command being called. DMSITS can determine whether this DC was inserted by examining the byte following the SVC call. If it is nonzero, then it is an instruction; if it is zero, then it is a "DC AL4(address)".

SVC 203: SVC 203 is used by CMS macros to perform various internal system functions. SVC 203 is an SVC call for which no parameter list is provided. An example is DMSFREE, for which the parameters are passed in registers 0 and 1.

A typical sequence for an SVC 203 call follows:

```
SVC   203
DC    H'code'
```

The halfword decimal code following the SVC 203 indicates the specific routine being called. DMSITS examines this halfword code as follows: (1) the absolute value of the code is taken, using an LPR instruction, (2) the first byte of the result is ignored, and the second byte of the resulting halfword is an index into a branch table, (3) the address of the correct routine is loaded, and control is transferred there, as the called routine.

It is possible for the address in the SVC 203 index table to be zero. In this case, the index entry contains an 8-byte routine or command name, which is processed in the same way as the 8-byte name passed in the parameter list passed to SVC 202.

The sign of the halfword code indicates whether the programmer expects an error return; if so, the code is negative: if not, the code is positive. Note that the sign of the halfword code has no effect on determining the routine which is to be called, because DMSITS takes the absolute value of the code to determine the called routine.

Because only the second byte of the absolute value of the code is examined by DMSITS, seven bits (bits 1-7) are available as flags or for other uses. For example, DMSFREE uses these seven bits to indicate such things as conditional requests and variable requests. Therefore, DMSITS considers the codes H'3' and H'259' to be identical, and handles them the same as H'-3' and H'-259', except for error returns.

When an SVC 203 is invoked, DMSITS stores the halfword code into the NUCON location CODE203, so that the called routine can interrogate the seven bits made available to it.

USER-HANDLED SVCs: The programmer may use the HNDSVC macro to specify the address of a routine that processes any SVC call for SVC numbers 0 through 200 and 206 through 255.

If the HNDSVC macro is used, the linkage conventions are as required by the user specified SVC-handling routine.

There is no way to specify a normal or error return from a user-handled SVC routine.

OS MACRO SIMULATION SVC CALLS: CMS supports certain of the SVC calls generated by OS macros, by simulating the effect of these macro calls.

The proper linkages are set up by the OS macro generations. DMSITS does not recognize any way to specify a normal or error return from an OS macro simulation SVC call.

DOS SVC CALLS: All SVC functions supported for CMS/DOS are handled by the CMS module DMSDOS. DMSDOS receives control from DMSITS (the CMS SVC handler) when that routine intercepts a DOS SVC code and finds that the DOSSVC flag in DOSFLAGS is set in NUCON.

DMSDOS acquires the specified SVC code from the OLDPSW field of the current SVC save area. Using this code, DMSDOS computes the address of the routine where the SVC is to be handled.

Many CMS/DOS routines (including DMSDOS) are contained in a discontiguous shared segment (DCSS). Most SVC codes are executed within DMSDOS, but some are in separate modules external to DMSDOS. If the SVC code requested is external to DMSDOS, its address is computed using a table called DCSSTAB; if the code requested is executed within DMSDOS, the table SVCTAB is used to compute the address of the code to handle the SVC.

DOS SVC calls are discussed in more detail in "Simulating a DOS Environment Under CMS" in this section.

INVALID SVC CALLS: There are several types of invalid SVC calls recognized by DMSITS. These are:

- Invalid SVC number. If the SVC number does not fit into any of the classes described above, it is not handled by DMSITS. An error message is displayed at the terminal, and control is returned directly to the caller.
- Invalid routine name in SVC 202 parameter list. If the routine named in the SVC 202 parameter list is invalid or cannot be found, then DMSITS handles the situation in the same way it handles an error return from a legitimate SVC routine. The error code is -3.
- Invalid SVC 203 code. If an illegal code follows SVC 203, an error message is displayed, and the ABEND routine is called to terminate execution.

Search Hierarchy for SVC 202

When a program issues SVC 202, and passes a routine or command name in the parameter list, DMSITS must search for the specified routine or command. (In the case of SVC 203 with a zero in the table entry for the specified index, the same logic must be applied.)

The search order is as follows:

1. A check is made to see if there is a routine with the specified name currently in the system transient area. If so, then control is transferred there.

2. The system function name table is searched to see if a command by this name is nucleus resident. If successful, control goes to the specified nucleus routine.

3. A search is made for a disk file with the specified name as the filename, and module as the filetype. The search is made in the standard disk search order. If this search is successful, then the specified module is loaded by LOADMOD and control passes to the storage location now occupied by the command.

4. If all searches so far have failed, then DMSINA (ABBREV) is called to see if the specified routine name is a valid system abbreviation for a system command or function. User-defined abbreviations and synonyms are checked at the same time. If this search is successful, then steps 2 through 4 are repeated with the full nonabbreviated name.

5. If all searches fail, then an error code of -3 is forced.

User and Transient Program Areas

There are two areas which can hold program modules which are loaded by LOADMOD from the disk. These are called the user program area and the transient program area.

The user program area starts at location X'20000' and extends upward to the loader tables. However, the high-address end of that area can be allocated as free storage by DMSFREE. Generally, all user programs and certain system commands, such as EDIT and COPYFILE, execute in the user program area. Because only one program can be executing in the user program area at one time, unless it is an overlay structure, it is impossible for one program in the user program area to invoke, by means of SVC 202, a module which is also intended to execute the user program area.

The transient program area is two pages, running from location X'E000' to location X'10000'. It provides an area for system commands that may also be invoked from the user program area by means of an SVC 202 call. For example, a program in the user program area may invoke the RENAME command, because this command is loaded into the transient program area.

The transient program area also handles certain OS macro simulation SVC calls. If DMSITS cannot find the address of a supported OS macro simulation SVC handling routine, it calls LOADMOD to load the file DMSSVT module into the transient area, and lets that routine handle the SVC.

A program in the transient program area may not invoke another program intended to execute in the transient program area, including OS macro simulation SVC calls that are handled by DMSSVT. Thus, for example, a program in the transient program area may not invoke the RENAME command. In addition, it may not invoke the OS macro WTO, which generates an SVC 35, which is handled by DMSSVT.

There is one further functional difference between the use of the two program areas. DMSITS starts a program in the user program area so that it is enabled for all interruptions. It starts a program in the transient program area so that it is disabled for all interruptions. Thus, the individual program may have to use the SSM (Set System Mask) instruction to change the current status of its system mask.

Called Routine Start-Up Table

Figures 42 and 43 show how the PSW and registers are set up when the called routine is entered.

Called Type	System Mask	Storage Key	Problem Bit
SVC 202 or 203 – Nuc resident	Disabled	System	Off
SVC 202 or 203 – Transient area MODULE	Disabled	User	Off
SVC 202 or 203 – User Area	Enabled	User	Off
User-handled	Enabled	User	Off
OS – Nuc res	Disabled	System	Off
OS – in DMSSVT	Disabled	System	Off

Figure 42. PSW Fields when Called Routine is Started

Returning to The Caller

When the called routine is finished processing it returns control to DMSITS, which then must return control to the caller.

RETURN LOCATION: The return is effected by loading the original SVC old PSW (which was saved at the time DMSITS was first entered), after possibly modifying the address field. How the address field is modified depends upon the type of SVC call, and on whether the called routine indicated an error return address.

Type	0 - 1	2 - 11	12	13	14	15
SVC 202 or 203	Same as caller	Unpredictable	Address of called routine	User save area	Return address to DMSITS	Address of called routine
Other	Same as caller	Same as caller	Address of called routine	User save area	Return address to DMSITS	Same as caller

Figure 43. Register Contents when Called Routine is Started

For SVC 202 and 203, the called routine indicates a normal return by means of a zero returned in register 15, and an error return by means of a nonzero in register 15. If the called routine indicates a normal return, then DMSITS makes a normal return to the caller. If the called routine indicates an error return, then DMSITS returns to the caller's error return address, if one was specified, and abnormally terminates if none was specified.

For SVC 202 not followed by "DC AL4(address)", a normal return is made to the instruction following the SVC instruction, and an error return causes an abnormal termination. For SVC 202 followed by "DC AL4(address)", a normal return is made to the instruction following the DC, and an error return is made to the address specified in the DC. In either case, register 15 contains the return code passed by the called routine.

For SVC 203 with a positive halfword code, a normal return is made to the instruction following the halfword code, and an error return causes an abnormally terminates. For SVC 203 with a negative halfword code, both normal and error returns are made to the instruction following the halfword code. In any case, register 15 contains the return code passed back by the called routine.

For OS macro simulation SVC calls, and for user-handled SVC calls, no error return is recognized by DMSITS. As a result, DMSITS always returns to the caller by loading the SVC old PSW that was saved when DMSITS was first entered.

REGISTER RESTORATION: Upon entry to DMSITS, all registers are saved as they were when the SVC instruction was first executed. Upon exiting from DMSITS, all registers are restored to the values that were saved at entry.

The exception to this is register 15 for SVC 202 and 203. Upon return to the caller, register 15 contains the value that was in register 15 when the called routine returned to DMSITS after it had completed processing.

System and User Save Area Formats

Whenever an SVC call is made, DMSITS allocates two save areas for that particular SVC call.

DMSITS uses the system save area (DSECT SSAVE) to save the value of the SVC old PSW at the time of the SVC call, the caller's registers at the time of the call, and any other necessary control information. Since SVC calls can be nested, there can be several of these save areas at one time. The system save area is allocated in protected free storage.

The user save area contains (DSECT EXTUAREA) 12 doublewords (24 fullwords), allocated in unprotected free storage. DMSITS does not use this area at all, but simply passes to the called routine a pointer to this area in register 13. Thus, the called routine can use this area as a temporary work area, or as a register save area. There is one user save area for each system save area, and the latter contains a pointer to the former in the USAVEPTR field.

LOAD AND EXECUTE TEXT FILES

The CMS loader consists of a nucleus resident loader (DMSLDR), a file and message handler program (DMSLIO), a library search program (DMSLIB), and other subroutine programs. DMSLDR starts loading at the user first location (AUSRAREA) specified in NUCON or at a user specified location. When performing an INCLUDE function, loading resumes at the next available location after the previous LOAD, INCLUDE, or LOADMOD.

The loader reads in the entire user's program, which consists of one or more control sections, each defined by a type 0 ESD record ("card"). Each control section contains a type 1 ESD card for each entry point and may contain other control cards.

Once the user's program is in storage, the loader begins to search his files for library subprograms called by the program. The loader reads the library subprograms into storage, relocating and linking them as required. To relocate programs, the loader analyzes information on the SLC, ICS, ESD, TXT, and REP cards. To establish linkages, it operates on ESD, and RLD cards. Information for end-of-load transfer of control is provided by the END and LDT cards, the ENTRY control card, START command, or RESET option.

The loader also analyzes the options specified on the LOAD and INCLUDE commands. In response to specified options, the loader can:

- Set the load area to zeros before loading (CLEAR option).
- Load the program at a specified location (ORIGIN option).
- Suppress creation of the load-map file on disk (NOMAP option).
- Suppress the printing of invalid card images in the load map (NOINV option).
- Suppress the printing of REP card images in the load map (NOREP option).
- Load program into "transient area" (ORIGIN TRANS option).
- Suppress TXTLIB search (NOLIBE option).
- Suppress text file search (NOAUTO option).
- Execute the loaded program (START option).
- Type the load map, if the TYPE option was specified.
- Set the program entry point (RESET option).

During its operation, the loader uses a loader table (REFTBL), and external symbol identification table (ESIDTB), and a location counter (LOCCNT). The loader table contains the names of control sections and entry points, their current location, and the relocation factor. (The relocation factor is the difference between the compiler-assigned address of a control section and the address of the storage location where it is actually loaded.) The ESIDTB contains pointers to the entries in REFTBL for the control section currently being processed by the loader. The loader uses the location counter to determine where the control section is to be loaded. Initially, the loader obtains from the nucleus constant area the address (LOCCNT) of the next location at which to start loading. This value is subsequently incremented by the length indicated on an ESD (type0), END, or ICS card, or it may be reset by an SLC card.

The loader contains a distinct routine for each type of input card. These routines perform calculations using information contained in the nucleus constant area, the location counter, the ESIDTB, the loader table, and the input cards. Other loader routines perform initialization, read cards into storage, handle error conditions, provide disk and typewritten output, search libraries, convert hexadecimal characters to binary, process end-of-file conditions, and begin execution of programs in core.

Following are descriptions of the individual subprocessors with LDR.

SLC Card Routine

Function
This routine sets the location counter (LOCCT) to the address specified on an SLC card, or to the address assigned (in the REFTBL) to a specified symbolic name.

Entry
The routine is entered at the first instruction when it receives control from the initial and resume loading routine. It is entered at ORG2 whenever a loader routine requires the current address of a symbolic location specified on an SLC card.

Operation
This routine determines which of the following situations exists, and takes the indicated action:

1. The SLC card does not contain an address or a symbolic name. The SLC card routine branches, via BADCRD in the reference table search routine, to the disk and type output routine (DMSLIO), which generates an error message.

2. The SLC card contains an address only. The SLC card routine sets the location counter (LOCCT) to that address and returns to RD, in the initial and resume loading routine, to read another card.

3. The SLC card contains a name only, and there is a reference table entry for that name. The SLC card routine sets LOCCT to the current address of that name (at ORG2) and returns to the initial and resume loading routine to get another card.

4. The SLC card contains a name only, and there is no reference table entry for that name. The SLC card routine branches via ERRSLC to the Disk and Type Output routine (DMSLIO), which generates an error message for that name.

5. The SLC card contains both an address and a name. If there is a REFTBL entry for the name, the sum of the current address of the name and the address specified on the SLC card is placed in LOCCT; control returns to the initial and resume loading routine to get another card. If there is no REFTBL entry for the name, the SLC card routine branches via ERRSLC to the Disk and Type Output routine, which generates an error message for the name.

ICS Card Routine - C2AE1

Function
This routine establishes a reference table entry for the control-segment name on the ICS card if no entry for that name exists, adjusts the location counter to a fullword boundary, if necessary, and adds the card-specified control-segment length to the location counter if necessary.

Entry
This routine has one entry point, named C2AE1. The routine is entered from the

initial and resume loading routine when it finds an ICS card.

Operation

1. The routine begins its operation with a test of card type. If the card being processed is not an ICS card, the routine branches to the ESD card analysis routine; otherwise, processing continues in this routine.

2. The routine tests for a hexadecimal address on the ICS card. If an address is present, the routine links to the DMSLSBA subroutine to convert the address to binary, otherwise the routine branches via BADCRD to the disk and type output routine (DMSLIO).

3. The routine next links to the REFTBL search routine, which determines whether there is a reference table entry for the card-specified control-segment name. If such an entry is found, the REFTBL search routine branches to the initial and resume loading routine; otherwise, the REFTBL search routine places the control-segment name in the reference table, and processing continues.

4. The routine determines whether the card-specified control-segment length is zero or greater than zero. If the length is zero, the routine places the current location counter value in the reference table entry as the control segment's starting address (ORG2), and branches to the initial and resume loading routine. If the length is greater than zero, the routine sets the current location counter value at a fullword boundary address. The routine then places this adjusted current location counter value in the reference table entry, adjusts the location counter by adding the specified control-segment length to it, and branches to RD in the initial and resume loading routine to get another card.

ESD Type 0 Card Routine - C3AA3

Function

This routine creates loader table and ESID table entries for the card-specified control section.

Entry

This routine has one entry point, location C3AA3. The routine is entered from the ESD card analysis routine.

Operation

1. If this is the first section definition, its ESDID is proved.

2. This routine first determines whether a loader table (REFTBL) entry has already been established for the card-specified control section. To do this, the routine links to the REFTBL search routine. The ESD type 0 card routine's subsequent operation depends on whether there already is a REFTBL entry for this control section. If there is such an entry, processing continues with operation 5, below; if there is not, the REFTBL search routine places the name of this control section in REFTBL, and processing continues with operation 3.

3. The routine obtains the card-specified control section length and performs operation 4.

4. The routine links to location C2AJ1 in the ICS card routine and returns to C3AD4 to obtain the current storage address of the control section from the REFTBL entry, inserts the REFTBL entry position (N - where this is the Nth REFTBL entry) in the card-specified ESID table location, and calculates the difference between the current (relocated) address of the control section and its card-specified (assembled) address. This difference is the relocation factor; it is placed in the REFTBL entry for this control section. If previous ESD's have been waiting for this CSECT, a branch is taken to SDDEF, where the waiting elements are processed. A flag is set in the REFTBL entry to indicate a section definition.

5. The entry found in the REFTBL is examined to determine whether it had been defined by a COMMON. If so, it is converted from a COMMON to a CSECT and performs operation 3.

6. If the entry had not been defined previously by an ESD type 0, processing continues at 3.

7. If the entry had been defined previously as other than COMMON, DMSLIO is called via ERRORM to print a warning message, "DUPLICATE IDENTIFIER". The entry in the ESID table is set negative so that the CSECT will be skipped (that is, not loaded) by the TXT and RLD processing routines.

ESD Type 1 Card Routine - ENTESD

Function

This routine establishes a loader table entry for the entry point specified on the ESD card, unless such an entry already exists.

Entry

This routine is entered from the ESD card analysis routine.

Operation

1. Branches and links to REFADR to find loader table entry for first section definition of the text deck saved by the ESD 0 routine.

2. The routine then adds the relocation factor and the address of the ESD found in operation 1 or the address in LOCCNT if an ESD has not yet been encountered. The sum is the current storage address of the entry point.

3. The routine links to the REFTBL search routine to find whether there is already a REFTBL entry for the card-specified entry point name. If such an entry exists, the routine performs operation 4. If there is no entry, the routine performs operation 5.

4. Upon finding a REFTBL entry that has been previously defined for the card-specified name, the routine then compares the REFTBL-specified current storage address with the address computed in operation 2. If the addresses are different, the routine branches and links to the DMSLIO routine (duplicate symbol warning); if the addresses are the same, the routine branches to location RD in the initial and resume loading routine to read another card. Otherwise, it is assumed that the REFTBL entry was created as a result of previously encountered external references to the entry. The DMSLSBC routine is called to resolve the previous external references and adjust the REFTBL entry. The entry point name and address are printed by calling DMSLIO.

5. If there is no REFTBL entry for the card-specified entry point name, the routine makes such an entry and branches to the DMSLIO routine.

ESD Type 2 Card Routine - C3AH1

Function
This routine creates the proper ESID table entry for the card-specified external name and places the name's assigned address (ORG2) in the reference table relocation factor for that name.

Entry
This routine has two entry points: location C3AH1 and location ESD00. Location C3AH1 is entered from the ESD card analysis routine; this occurs when an ESD type 2 card is being processed. Location ESD00 is entered from:

- The ESD card analysis routine, when the card being processed is an ESD type 2, and an absolute loading process is indicated.
- The ESD type 0 card routine and ESD type 1 card routine, as the last operation in each of these routines.

Operation

1. When this routine is entered at location C3AH1, it first links to the REFTBL search routine to determine whether there is a REFTBL entry for the card-specified external name. If none is found, the REFTBL search routine sets the undefined flag for the new loader table entry.

2. The routine resets a possible WEAK EXTRN flag. The routine next places the REFTBL entry's position-key in the ESID table. If the entry has already been defined by means of an ESD type 0, 1, 5, or 6, processing continues at operation 4. Otherwise, it continues at operation 3.

3. The relocated address is placed in the RELFAC entry in the external name's REFTBL entry.

4. The ESD type 2 card routine then determines (at location ESD00) whether there is another entry on the ESD card. If there is another entry, the routine branches to location CA3A1 in the ESD card analysis routine for further processing of this card; otherwise, the routine branches to location RD in the initial and resume loading routine.

Exits
This routine exits to location CA3A1 in the ESD card analysis routine if there is another entry on the ESD card being processed, and exits to location RD in the initial and resume loading routine if the ESD card requires no further processing.

ESD Type 4 Routine - PC

Function
This routine makes loader table and ESIDTAB entries for private code CSECT.

Operation
The ESD Type 4 Card Routine:

1. The routine LDRSYM is called to generate a unique character string number of the form 00000001, which is left in the external data area NXTSYM; it is greater in value than previously generated symbol.

2. The CSECT is then processed as a normal type 0 ESD with the above assigned name.

ESD Types 5 and 6 Card Routine - PRVESD and COMESD

Function
This routine creates reference table and ESIDTAB entries for common and pseudo-register ESDs.

Operation
The ESD type 5 and 6 card routine:

1. Links to ESIDINC in the ESD type 0 card routine, to update the number of ESIDTB entries.

2. Links to the REFTBL search routine to determine whether a reference table (REFTBL) entry has already been created. If there is no entry, the REFTBL search routine places the name of the item in the REFTBL.

3. If the REFTBL search routine had to create an entry for the item, the ESD type 5 and 6 card routine indexes it in the ESIDTB, enters the length and alignment in the entry, indicates whether it is a PR or common, and branches to ESD00 in the ESD type 2 card routine to determine whether the card contains additional ESD's to be processed. If the entry is a PR, the ESD type 5 and 6 card routine enters its displacement and length in the REFTBL before branching to ESD00.

4. If the REFTBL already contained an entry, the ESD type 5 and 6 card routine indexes it in the ESIDTB, checks alignment and branches to ESD00.

Note: The PR alignment is coded and placed into the REFTBL. It is an error to encounter more restrictive alignment PR than previously defined. A blank alignment factor is translated to fullword alignment.

ESD Type 10 Routine - WEAK EXTRN

The WEAK EXTRN routine calls the search routine to find the EXTRN name in the loader table. If not found, set the WEAK EXTRN flag in the new loader table entry. Exit to ESD00.

TXT Card Routine - C4AA1

Function

This routine has two functions: address inspection and placing text in storage.

Entry

This routine has three entry points: location C4AA1, which is entered from the ESD card analysis routine, and locations REPENT and APR1, which are entered from the REP card routine for address inspection.

Operation

1. This routine begins its operation with a test of card type. If the card being processed is not a TXT card, the routine branches to the REP card routine; otherwise, processing continues in this routine.

2. The routine then determines how many bytes of text are to be placed in storage, and finds whether the loading process is absolute or relocating. If the loading process is absolute, the routine performs operation 4, below; if relocating, the routine performs operation 3.

3. If the ESIDTB entry was negative, this is a duplicate to CSECT and processing branches to RD. Otherwise, the routine links to the REFADR routine to obtain the relocation factor of the current control segment.

4. The routine then adds the relocation factor (0, if the loading process is absolute) and the card-specified storage address. The result is the address at which the text must be stored. This routine also determines whether the address is such that the text, when loaded starting at that address, overlays the loader or the reference table. If a loader overlay or a reference table overlay is found, the routine branches to the LDRIO routine. If neither condition is detected, the routine proceeds with address inspection.

5. The routine then determines whether an address has already been saved for possible use as the end-of-load branch address. If an address has been saved, the routine performs operation 7; if not, the routine performs operation 6.

6. The routine determines whether the text address is below location 128. If the address is below location 128, it should not be saved for use as a possible end-of-load branch address, and the routine performs operation 7; otherwise the routine saves the address and then performs operation 7.

7. The routine then stores the text at the address specified (absolute or relocated) and branches to location RD in the initial and resume loading routine to read another card.

Exits

The routine exits to two locations, as follows:

1. The routine exits to location RD in the initial and resume loading routine if it is being used to process a TXT card.

2. The routine exits to location APRIL in the REP card routine if it is being used for REP card address inspection.

REP Card Routine - C4AA3

Function

This routine places text corrections in storage.

Entry

This routine has one entry point, location C4AA3. The routine is entered from the TXT card routine.

Operation

1. This routine begins its operation with a test of card type. If the card being processed is not a REP card, the routine branches to the RLD card routine;

otherwise, processing continues in this routine.

2. The routine then links to the HEXB conversion routine to convert the REP card-specified correction address from hexadecimal to binary.

3. The routine then links to the HEXB conversion routine again to convert the REP card-specified ESID from hexadecimal to binary.

4. The routine then determines whether the 2-byte correction being processed is the first such correction on the REP card. If it is the first correction, the routine performs operation 5; otherwise, the routine performs operation 6.

5. When the routine is processing the first correction, it links to location REPENT in the TXT card routine, where the REP card-specified correction address is inspected for loader overlay and for end-of-load branch address saving; in addition, if the loading process is relocating, the relocated address is calculated and checked for reference table overlay. The routine then performs operation 7.

6. When the correction being processed is not the first such correction on the REP card, the routine branches to location APR1 in the TXT card routine for address inspection.

7. The routine then links to the HEXB conversion routine to convert the correction from hexadecimal to binary, places the correction in storage at the absolute (card-specified) or relocated address, and determines whether there is another correction entry on the REP card. If there is another entry, the routine repeats its processing from operation 4, above; otherwise, the routine branches to location RD in the initial and resume loading routine.

Exits

When all the REP-card corrections have been processed, this routine exits to location RD in the initial and resume loading routine.

RLD Card Routine - C5AA1

Function

This routine processes RLD cards, which are produced by the assembler when it encounters address constants within the program being assembled. This routine places the current storage address (absolute or relocated) of a given defined symbol or expression into the storage location indicated by the assembler. The routine must calculate the proper value of the defined symbol or expression and the proper address at which to store that value.

Entry

This routine has two entry points, locations C5AA1 and PASSTWO.

Operation

1. Location C5AA1 writes each RLD card into a work file (DMSLDR CMSUT1). Exit to RD to process the next card.

   Location PASSTWO reads an RLD card from the work file. At EOF got to C6AB6 to finish this file.

2. The routine uses the relocation header (RH ESID) on the card to obtain the current address (absolute or relocated) of the symbol referred to by the RLD card. This address is found in the relocation factor section of the proper reference table entry. If the RH ESID is 0, the routine branches to the LDRIO routine (invalid ESD).

3. The routine uses the position header (PH ESID) on the card to obtain the relocation factor of the control segment in which the DEFINE CONSTANT assembler instruction occurred. If the PH ESID is 0, the routine branches to BADCRD in the REFTBL search routine (invalid ESID). If the ESIDTAB entry is negative (duplicate CSECT), the RLD entry is skipped.

4. The routine next decrements the card-specified byte count by 4 and tests it for 0. If the count is now 0, the routine branches to location RD in the initial and resume loading routine; otherwise, processing continues in this routine.

5. The routine determines the length, in bytes, of the address constant referred to in the RLD card. This length is specified on the RLD card.

6. The routine then adds the relocation factor obtained in operation 3 (relocation factor of the control segment in which the current address of the symbol must be stored), and the card-specified address. The sum is the current address of the location at which the symbol address must be stored.

7. The routine then computes the arithmetic value (symbol address or expression value) that must be placed in storage at the address calculated in operation 6, above, and places that value at the indicated address. If the value is undefined, the routine branches to location DMSLSBB, where the constant is added to a string of constants that are to be defined later.

8. The routine again decrements the byte count of information on the RLD card and tests the result for zero. If the result is zero, go to operation 2; otherwise, processing continues in this routine.

9. The routine next checks the continuation flag, a part of the data placed on the RLD card by the assembler. If the flag is on, the routine repeats its processing for a new address only; the processing is repeated from operation 4. If the flag is off, the routine repeats its processing for a new symbol; the processing is repeated from operation 2.

Exits

This routine exits to location RD in the initial and resume loading routine.

END Card Routine - C6AA1

Function

This routine saves the END card address under certain circumstances, and initializes the loader to load another control segment.

Entry

This routine has one entry point, location C6AA1. The routine is entered from the RLD card routine.

Operation

1. This routine begins its operation with a test of card type. If the card being processed is not an END card, the routine branches to the LDT card routine; otherwise, processing continues in this routine.

2. The routine then determines whether the END card contains an address. If the card contains no address, the routine performs operation 7, below; otherwise, the routine performs operation 3.

3. The routine next checks the end-address-saved switch. If this switch is on, an address has already been saved, and the routine performs operation 7. If the switch is off, the routine performs operation 4.

4. The routine determines whether loading is absolute or relocated. If the loading process is absolute, the routine performs operation 6; otherwise, the routine performs operation 5.

5. The routine links to the REFADR routine to obtain the current relocation factor, and adds this factor to the card-specified address.

6. The routine stores the address (absolute or relocated) in area BRAD, for possible use at the end-of-load transfer of control to the problem program.

7. Goes to location PASSTWO (in RLD routine) to process RLD cards.

8. The routine then clears the ESID table, sets the absolute load flag on, and branches to the location specified in a general register (see "Exits").

Exits

This routine exits to the location specified in a general register. This may be either of two locations:

1. Location RD in the initial and resume loading routine. This exit occurs when the END card routine is processing an END card.

2. The location in the LDT card routine that is specified by that routine's linkage to the END card routine. This exit occurs when the LDT card routine entered this routine to clear the ESID table and set the absolute load flag on.

Control Card Routine - CTLCRD1

Function

This routine handles the ENTRY and LIBRARY control cards.

Entry

This routine has one entry point, location CTLCRD1. The routine is entered from the LDT card routine.

Operations

1. The CMS function SCAN is called to parse the card.

2. If the card is not an ENTRY or LIBRARY card, the routine determines whether the NOINV option (no printing of invalid card images) was specified. If printing is suppressed, control passes to RD in the initial and resume loading routine, where another card is read. If printing is not suppressed, control passes to the disk and type output routine (DMSLIO), where the invalid card image is printed in the load map. If the card is a valid control card, processing continues.

ENTRY Card

3. If the ENTRY name is already defined in REFTBL, its REFTBL address is placed in ENTADR. Otherwise, a new entry is made in REFTBL, indicating an undefined external reference (to be resolved by later input or library search), and this REFTBL entry's address is placed in ENTADR.

4. The control card is printed by calling DMSLIO via CTLCRD; it then exits to RD.

LIBRARY Card

5. Only nonobligatory reference LIBRARY cards are handled; any others are considered invalid.

6. Each entry-point name is individually isolated and is searched for in the REFTBL. If it has already been loaded and defined, nothing is done and the next entry-point name is processed.

Otherwise, the nonobligatory bit is set in the flag byte of the REFTBL entry.

7. Processing continues at operation 4.

REFADR Routine (DMSLDRB)

Function

This routine computes the storage address of a given entry in the reference table.

Entry

This routine has one entry point, location REFADR. The routine is entered for several of the routines within the loader.

Operation

1. Checks to see if requested ESDID is zero. If so, uses LOCCNT as requested location; branches to the return location + 44; otherwise continues this routine.

2. The routine first obtains, from the indicated ESID table entry, the position (n) of the given entry within the reference table (where the given entry is the nth REFTBL entry).

3. The routine then multiplies n by 16 (the number of bytes in each REFTBL entry) and subtracts this result from the starting address of the reference table. The starting address of the reference table is held in area TBLREF; this address is the highest address in storage, and the reference table is always built downward from that address.

4. The result of the subtraction in operation 2, above, is the storage address of the given reference table entry. If there is no ESD for the entry, goes to operation 5; otherwise, this routine returns to the location specified by the calling routine.

5. Adds an element to the chain of waiting elements. The element contains the ESD data item information to be resolved when the requested ESDID is encountered.

PRSERCH Routine (DMSLDRD)

Function

This routine compares each reference table entry name with the given name determining (1) whether there is an entry for that name and (2) what the storage address of that entry is.

Entry

This routine is initially entered at PRSERCH, and subsequently at location SERCH. The routine is entered from several routines within the loader.

Operation

1. This routine begins its operation by obtaining the number of entries currently in the reference table (this number is contained in area TBLCT), the size of a reference table entry (16 bytes), and the starting address of the reference table (always the highest address in storage, contained in area TBLREF).

2. The routine then checks the number of entries in the reference table. If the number is 0, the routine performs operation 5; otherwise, the routine performs operation 3.

3. The routine next determines the address of the first (or next) reference table entry to have its name checked, increments by one the count it is keeping of name comparisons, and compares the given name with the name contained in that entry. If the names are identical, PRSERCH branches to the location specified in the routine that linked to it. PRSERCH then returns the address of the REFTBL entry; else PRSERCH performs operation 4.

4. The routine then determines whether there is another reference table entry to be checked. If there is none, the routine performs operation 5; if there is another, the routine decrements by one the number of entries remaining and repeats its operation starting with operation 3.

5. If all the entries have been checked, and none contains the given name for which this routine is searching, the routine increments by one the count it is keeping of name comparisons, places that new value in area TBLCT, moves the given name to form a new reference table entry, and returns to the calling program.

Exits

This routine exits to either of two locations, both of which are specified by the routine that linked to this routine. The first location is that specified in the event that an entry for the given name is found; the second location is that specified in the event that such as entry is not found.

Loader Data Bases

ESD Card Codes (col. 25...)

Code	Meaning
00	SD (CSECT or START)
01	LD (ENTRY)
02	ER (EXTRN)
04	PC (Private code)
05	CM (COMMON)
06	XD (Pseudo-register)
0A	WX (WEAK EXTERN)

ESIDTB Entry

The ESD ID table (ESIDTB) is constructed separately for each text deck processed by the loader. The ESIDTB produces a correspondence between ESD ID numbers (used on RLD cards) and entries in the loader reference table (REFTBL) as specified by the ESD cards. Thus, the ESIDTB is constructed while processing the ESD cards. It is then used to process the TXT and RLD cards in the text deck.

The ESIDTB is treated as an array and is accessed by using the ID number as an index. Each ESIDTB entry is 16 bits long.

Bits	Meaning
0	If 1, this entry corresponds to a CSECT that has been previously defined. All TXT cards and RLD cards referring to this CSECT in this text deck should be ignored.
1	If 1, this entry corresponds to a CSECT definition (SD).
2	Waiting ESD items exist for this ESDID.
3	Unused.
4-15	REFTBL entry number (e.g. 1, 2, 3, etc.)

Bit 1 is very crucial because it is necessary to use the VALUE field of the REFTBL if the ID corresponds to an ER, CM, or PR; but, the INFO field of the REFTBL entry must be used in the ID corresponds to an SD.

REFTBL Entry

```
┌──────────────────────────────────────────────┐
|0(0)                                          |
|- - - - - - - - -  NAME  - - - - - - - - - - -|
|                                              |
|----------------------------------------------|
|8(8)          |9(9)                           |
|    FLAG1     |            INFO               |
|--------------|-------------------------------|
|12(C)         |13(D)                          |
|    NOTE1     |            VALUE              |
|--------------|-------------------------------|
|16(10)        |17(11)                         |
|    FLAG2     |            ADDRESS            |
└──────────────────────────────────────────────┘
```

A REFTBL entry is 20 bytes. The fields have the following uses:

NAME Field: contains the symbolic name from the ESD data item.

FLAG1 BYTE

Loader Code	ESD Code	Routine Label	Meaning
7C	00	XBYTE	PR - byte alignment
7D	01	XHALF	PR - halfword alignment
7E	03	XFULL	PR - fullword alignment
7F	07	XDBL	PR - doubleword alignment
80	05	XUNDEF	Undefined symbol
81	04	XCXD	Resolve CXD
82	02	XCOMSET	Define common area
83	05	WEAKEXT	Weak external reference
90	06	CTLLIB	TXTLIBs not to be used to resolve names

INFO Field: depends upon the type of the ESD item.

ESD Item Type	INFO Field Meaning
SD (CSECT or START)	Relocation factor
LD (ENTRY)	zero
CM (COMMON)	maximum length
PR (Psuedo Register)	-

VALUE Field: depends upon the type of the ESD item, as does the INFO field.

ESD Item Type	VALUE Field Meaning
SD (CSECT or START)	Absolue address
LD (ENTRY)	Absolue address
CM (COMMON)	Absolue address
PR (Psuedo register)	Assigned value (starting from 0)

FLAG2 Byte

Bit	Meaning	Bit	Meaning
0	Unused	4	Unused
1	Unused	5	Name was located in a TXTLIB
2	Unused	6	Section definition entry
3	Unused	7	Name specifically loaded from command line.

ADDRESS Field: Unused

Entries may be created in the loader reference table prior to the actual defining of the symbol. For example, an entry is created for a symbol if it is referenced by means of an EXTRN (ER) even if the symbol has not yet been defined or its type known. Furthermore, common (CM) is not assigned absolute addresses until prior to the start of execution by the START command.

These circumstances are determined by the setting of the flag byte; if the symbol's value has not yet been defined, the value field specifies the address of a patch control block (PCB).

Patch Control Block (PCB)

These are allocated from free storage and pointed at from REFTBL entries or other PCBs.

Byte	Meaning
0-3	Address of next PCB
5-7	Location of ADCON in storage
4	Flag byte

All address constant locations in loaded program for undefined symbols are placed on PCB chains.

Loader Input Restrictions

All restrictions which apply to object files for the OS linkage editor apply to CMS loader input files.

PROCESS COMMANDS THAT MANIPULATE THE FILE SYSTEM

Figure 40 lists the CMS modules that perform either general file system support functions or that perform data manipulation.

MANAGE THE CMS FILE SYSTEM

A description of the structure of the CMS file system and the flow of routines that access and update the file system follows.

HOW CMS FILES ARE ORGANIZED IN STORAGE

CMS files are organized in storage by three types of data blocks: the file status table (FST), chain links, and file records. Figure 44 shows how these types of data blocks relate to each other; the following text and figures describe these relationships and the individual data blocks in more detail.

FILE STATUS TABLES

CMS files consist of 800-byte records whose attributes are described in the file status table (FST). The file status table is defined by DSECT FSTSECT. The FST consists of such information as the filename, filetype, and filemode of the file, the date on which the file was last written, and whether the file is in fixed-length or variable format. Also, the FST contains a pointer to the first chain link. The first chain link is a block that contains addresses of the data blocks that contain the actual data for the file.

The FSTs are grouped into 800-byte blocks called FST Blocks (these are sometimes referred to in listings as hyperblocks). Each FST block contains 20 FST entries, each describing the attributes of a separate file. Figure 45 shows the structure of an FST block and the fields defined in the FST.

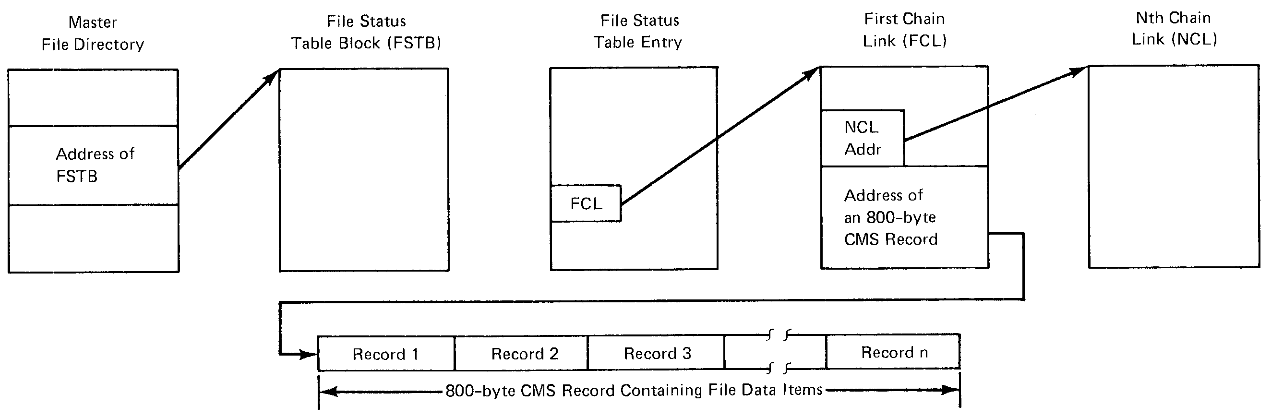


Figure 44. How CMS File Records are Chained Together

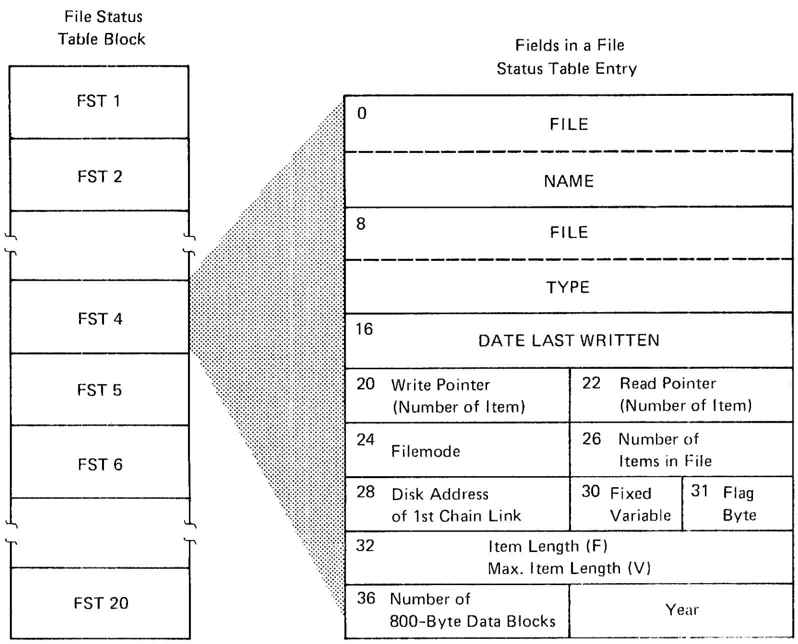


Figure 45. Format of a File Status Block; Format of a File Status Table

CHAIN LINKS

Chain links are 200- or 800-byte blocks of storage that chain the records of a file in storage. There are two types of chain links: first chain links and Nth chain links.

The first chain link points to two kinds of data. The first 80 bytes of the first chain link contain the halfword addresses of the remaining 40 chain links used to chain the records of the file. The next 120 bytes of the file are the halfword addresses of the first 60 records of the file.

The Nth chain links contain only halfword addresses of the records that contained in the file.

Because there are 41 chain links (of which the first contains addresses for only 60 records), the maximum size for any CMS file is 16,060 800-byte records.

CMS RECORD FORMATS

CMS records are 800-byte blocks containing the data that comprises the file. For example, the CMS record may contain several card images or print images, each of which is referred to a record item. Figure 46 shows how chain links are chained together.

CMS records can be stored on disk in either fixed-length or variable-length format. However, the two formats may not be mixed in a single file.

Regardless of their format, the items of a file are stored by CMS in sequential order in as many 800-byte records as are required to accommodate them. Each record (except the last) is completely filled and items that begin in one record can end on the next record. Figure 47 shows the arrangement of records in files for files containing fixed-length records and files containing variable-length records.

The location of any item in a file containing fixed-length records is determined by the formula:

$$\text{locations} = \frac{(\text{Item Number} - 1) \times \text{Record Length}}{800}$$

where the quotient is the number of the item and the remainder is the displacement of the item into the file.

For variable-length records, each record is preceded by a 2-byte field specifying the length of the record.

DISK ORGANIZATION IN CMS

CMS virtual disks (also referred to as minidisks) are blocks of data designed to externally parallel the function of real disks. Several virtual disks may reside on one real disk.

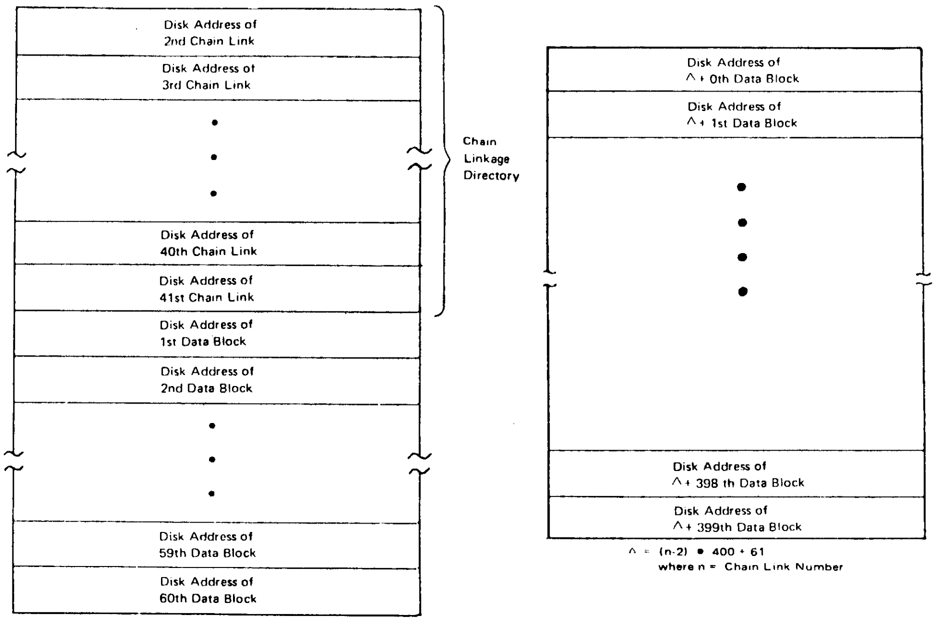


Figure 46. Format of the First Chain Link and Nth Chain Links

Figure 47. Arrangement of Fixed-Length or Variable- Length Records in Files

A CMS virtual machine may have up to 10 virtual disks accessed during a terminal session, depending on user specifications. Some disks, such as the S-disk, are accessed during CMS initialization; however, most are accessed dynamically as they are needed during a terminal session.

PHYSICAL ORGANIZATION OF VIRTUAL DISKS

Virtual disks are physically organized in 800-byte records. Records 1 and 2 of each user disk are reserved for IPL. Record 3 contains the disk label. Record 4 contains the master file directory. The remaining records on the disk contain user file-related information such as the FSTs, chain links, and the individual file records discussed above.

THE MASTER FILE DIRECTORY

The master file directory (MFD) is the major file management table for a virtual disk. As mentioned earlier, it resides on cylinder 0, track 0, record 4 of each virtual disk. Six types of information contained in the master file directory:

- The disk addresses of the FST entries describing user files on that disk.
- A 4-byte "sentinel," which can be either FFFD or FFFF. FFFD specifies that extensions of the QMSK (described below) follow. FFFF specifies that no QMSK extensions follow.
- Extensions to the QMSK, if any.
- General information describing the status of the disk:
  - ADTNUM - The total number of 800-byte blocks on the user's disk.
  - ADTUSED - The number of blocks currently in use on the disk.
  - ADTLEFT - Number of blocks remaining for use (ADTNUM - ADTUSED).
  - ADTLAST - Relative byte address of the last record in use on the disk.
  - ADTCYL - Number of cylinders on the user's disk.
  - Unit Type - A 1-byte field describing the type of the disk: 08 for a 2314, 09 for a 3330.
  - A bit mask called the QMSK, which keeps track of the status of the records on disk. The QMSK is described in more detail below.
  - Another bit map, called the QQMSK, which is used only for 2314 disks and performs a function similar to that of QMSK.

Figure 48 shows the structure of the master file directory. Figure 44 shows the relationship of the Master File Directory, which resides on disk, to data blocks brought into storage for file management purposes, for example, FSTs and chain links.

KEEPING TRACK OF R/W DISK STORAGE: QMSK AND QQMSK

Because large areas of disk space need not be contiguous in CMS, but are composed of 800-byte blocks chain-linked together, disk space management needs to determine only the availability of blocks, not extents. The status of the blocks on any read/write disk (which blocks are available and which are currently in use) is stored in a table called QMSK. The term QMSK is derived from the fact that a 2311 disk drive has four 800-byte blocks per track. One block is a "quarter-track", or QTRK, and a 200-byte area is a "quarter-quarter-track", or QQTRK. The bit mask for 2314, 2319, 3340, or 3330 records is called the QMSK, although each 800-byte block represents less than a quarter of a track on these devices.

On a 2314 or 2319 disk, the blocks are actually grouped fifteen 800-byte blocks per even/odd pair of tracks. An even/odd pair of tracks is called a track group. On a 3330 disk, the blocks are grouped fourteen 800-byte blocks per track. On a 3340 disk, the blocks are grouped into eight 800-byte blocks per track.

When the system is not in use, a user's QMSK resides on the Master File Directory; during a session it is maintained on disk, but also resides in main storage. QMSK is of variable length, depending on how many cylinders exist on the disk.

Each bit is associated with a particular block on the disk. The first bit in QMSK corresponds to the first block, the second bit to the second block, and so forth, as shown in Figure 49.

When a bit in QMSK is set to 1, it indicates that the corresponding block is in use and not available for allocation. A 0-bit indicates that the corresponding block is available. The data blocks are referred to by relative block numbers throughout disk space management, and the disk I/O routine, DMSDIO, finally converts this number to a CCHHR disk address.

A table called QQMSK indicates which 200 byte segments (QQTRK) are available for allocation and which are currently in use. QQMSK contains 100 entries, which are used to indicate the status of up to 100 QQTRK records. An entry in QQMSK contains either a disk address, pointing to a QQTRK record that is available for allocation, or zero. QQMSK is used only for 2314 files; for 3330, 3340, and 3350, the first chain link occupies the first 200-byte area of an 800-byte block.

The QMSK and QQMSK tables for read-only disks are not brought into storage, since no space allocation is done for a disk while it is read-only. They remain, as is, on the disk until the disk is accessed as a read/write disk.

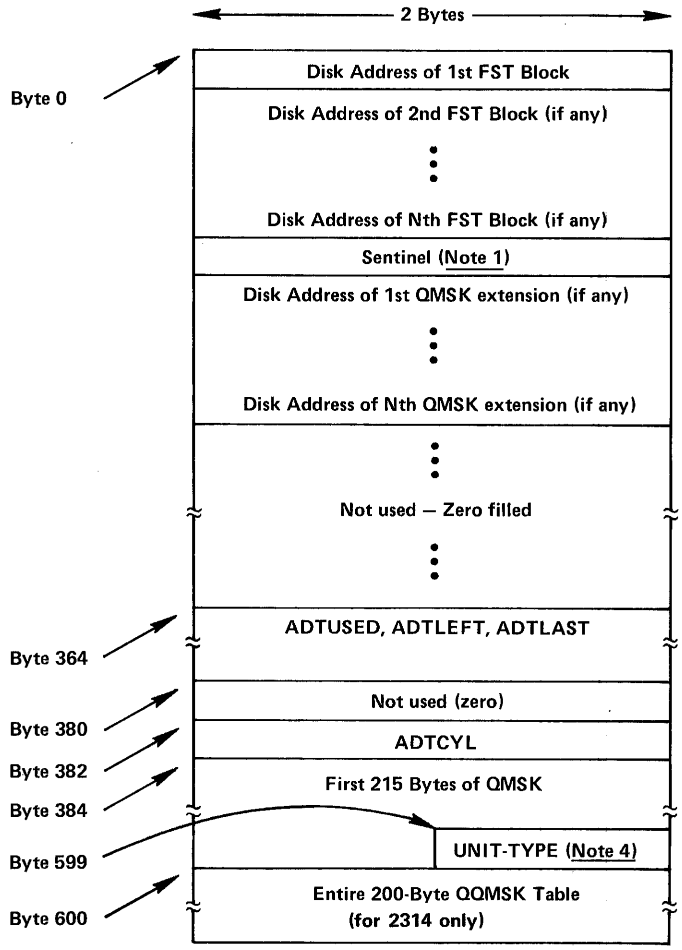
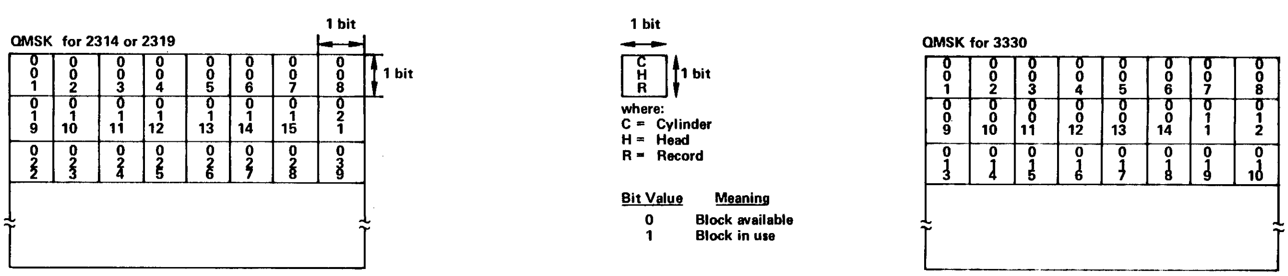


Figure 48. Structure of the Master File Directory



Number of QMSK Extensions Required (if any)	Number of Cylinders on Disk			
	2314 or 2319	3330	3340	3350
0	1 - 11	1 - 6		
1	12 - 54	7 - 30		
2	55 - 96	31 - 54		
3	97 - 139	55 - 78		
4	140 - 182	79 - 102		
5	183 - 203	103 - 126		
6	-	127 - 150		
7	-	151 - 174		
8	-	175 - 198		
9	-	199 - 223		
10	-	224 - 246		

Figure 49. Disk Storage Allocation Using the QMSK Data Block

DYNAMIC STORAGE MANAGEMENT: ACTIVE DISKS AND FILES

CMS disks and files contained on disk are physically mapped using the data blocks described above: for disks, the QMSK, QQMSK, and the MFD; for files, the FST, chain links, and 800-byte file records. In storage, all of this data is accessed by means of two DSECTs whose addresses are defined in the DSECT NUCON, ADTSECT and AFTSECT.

Managing Active Disks: The Active Disk Table

The ADTSECT DSECT maps information in the active disk table (ADT). This information includes data contained in the MFD, FST blocks, the QMSK, and QQMSK. The DSECT comprises of ten "slots," each representing one CMS virtual disk. A slot contains significant information about the disk such as a pointer to the MFD for the disk, a pointer to the first FST block and pointers to the QMSK and QQMSK, if the disk is a R/W disk. Also contained in ADTSECT is information such as the number of cylinders on the disk, the number of records on the disk.

Managing Active Files: The Active File Table

Each open file is represented in storage by an active file table (AFT). The AFT (defined by the AFTSECT DSECT) contains data found on disk in FSTs, chain links, and data records. Also contained in the AFT is such information as the address of the first chain link for the file, the current chain link for the file, the address of the current data block, the fileid information for the file. Figure 39 shows the relationship between the AFT and other CMS data blocks.

CMS ROUTINES USED TO ACCESS THE FILE SYSTEM

DMSACC is the control routine used to access a virtual disk. In conjunction with DMSACM and DMSACF, DMSACC builds, in virtual storage, the tables CMS requires for processing files contained on the disk. The list below shows the logical flow of the main function of DMSACC.

Access a Virtual Disk: DMSACC

DMSACC: Scans the command line to determine which disk is specified.

DMSLAD: Looks up the address of the ADT for the disk specified on the command line.

DMSACC: Determines whether an extension to a disk has been specified on the command line and ensures that it is correctly specified.

DMSLAD: In the case where an extension has been specified, calls DMSLAD to ensure that the extension disk exists.

DMSLAD: Ensures that the specified disk is not already accessed as a R/W disk.

DMSFNS: In the case where the specified disk is replacing a currently accessed disk, closes any open files belonging to the duplicate disk.

DMSACC: Verifies the parameters remaining on the command line.

DMSALU: Releases any free storage belonging to the duplicate disk via a call to DMSFRE. Also, clears appropriate entries in the ADT for use by the new disk.

DMSACM: (Called as the first instruction by DMSACF) Reads, from the Master File Directory, QMSK, and the QQMSK for the specified disk; also, DMSACM updates the ADT for the specified disk using information from the MFD.

DMSACF: Reads into storage all the FST blocks associated with the specified disk.

DMSACC: Handles error processing or processing required to return control to DMSINT.

INPUT/OUTPUT OPERATIONS

CMS input/output operations for disk, tape, and unit record devices are always synchronous. Disk and tape I/O is initiated via a privileged instruction, DIAGNOSE, whose function code requests CP to perform necessary error recovery. Control is not returned to CMS until the operation is complete, except for tape rewind or rewind and unload operations, which return control immediately after the operation is started. No interruption is ever received as the result of DIAGNOSE I/O. The CSW is stored only in the event of an error.

Input/output operations to a card reader, card punch, or printer are initiated via a normal START I/O instruction. After starting the operation, CMS enters the wait state until a device end interruption is received from the started device. Because the I/O is spooled by CP, CMS does not handle any exceptional conditions other than not ready, end-of-file, or forms overflow.

CMS input/output operations to the terminal may be either synchronous or asynchronous. Output to the terminal is always asynchronous, but a program may wait for all terminal input/output operations to complete by calling the console wait routine. Input from the terminal is usually synchronous but a user may cause CMS to issue a read by pressing the attention key. A program may also asynchronously stack data to be read by calling the console attention routine.

UNIT RECORD I/O PROCESSING

Seven routines handle I/O processing for CMS: DMSRDC, DMSPUN, and DMSPRT handle the READCARD, PUNCH, and PRINT commands and pass control to te actual I/O processors, DMSCIO (for READCARD and PUNCH) or DMSPIO (for PRINT). DMSCIO and DMSPIO issue the SIO instructions that cause I/O to take place. Two other routines, DMSIOW and DMSITI, handle synchronization processing for I/O operations. Figure 50 shows the overall flow of control for I/O operations.

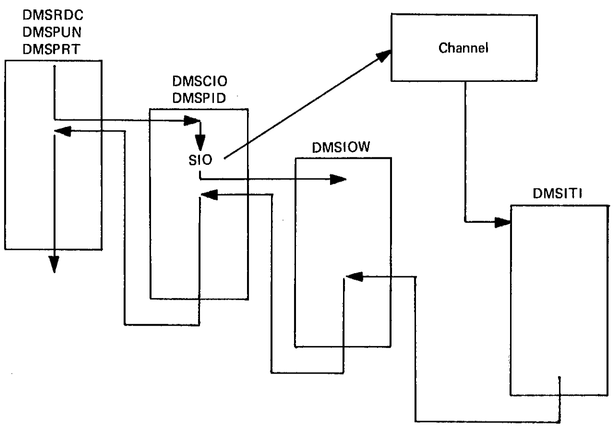


Figure 50. Flow of Control For Unit Record I/O Processing

The following are more detailed descriptions of the flow of control for the read, punch, and print unit record control functions.

Read A Card

DMSRDC: Initializes block length and unit record size.

DMSCIO: Initializes areas to read records.

DMSCIO: Issues an SIO command to read a record.

DMSIOW: Sets the wait bit for the virtual card reader and load the I/O old PSW from NUCON. This causes CMS to enter a wait state until the read I/O is complete.

DMSITI: Ensures that this interrupt is for the virtual reader. If not, the I/O old PSW is loaded, returning CMS to a wait state. If the interrupt is for the reader, DMSITI resets the wait bit in the I/O old PSW and loads it, causing control to return to DMSIOW.

DMSIOW: Places the symbolic name of the interrupting device in the PLIST and passes control to the calling routine.

DMSCIO: Checks for SENSE information and handle I/O errors, if necessary.

DMSCWR: Displays a control record at the console.

DMSSCN: If another control record is encountered, formats it via DMSSCN.

DMSCWR: Displays the new control record at the console.

DMSFNS: Closes the file when end-of-file occurs.

DMSRDR: Issues a CP CLOSE command to close the card reader.

Punch a Card

DMSPUN: Ensures that a virtual punch is available; processes PUNCH command options.

DMSSTT: Verifies the existence of the file and returns its starting address.

DMSPUN: If requested, sets up a header record and calls DMSCWR to write it to the console.

DMSBRD: Reads a block of data into the read buffer; continues reading until the buffer is filled.

DMSCIO: Initializes areas to punch records.

DMSCIO: Issues the SIO instruction to punch the contents of the buffer.

DMSCIO: Issues a call to DMSIOW to wait for completion of the punch I/O operation.

DMSIOW: Sets the wait bit on for the virtual punch device and loads the I/O old PSW from NUCON. This causes CMS to enter a wait state until the punch operation completes.

DMSITI: Ensures that this interrupt is for the punch. If not, the I/O old PSW is loaded returning CMS to a wait state. If the interrupt is for the punch, DMSITI resets the wait bit in the I/O old PSW and then loads the PSW, returning control to DMSIOW.

DMSIOW: Places the symbolic name of the interrupting device in the PLIST and passes control to DMSCIO.

DMSCIO: Checks for SENSE information and handles I/O errors, if any.

DMSPUN: Handles error returns and resets constants for the next punch operation.

DMSFNS: Closes the file and returns control to the command handler, DMSINT.

Print a File

DMSPRT: Determines the device type of the printer. Checks out the specified fileid. Checks out the options specified on the PRINT command line.

DMSSCN: Verifies the existence of the file and returns its starting address.

DMSPRT: Determines the record size to be printed and sets up an appropriate buffer area via a call to DMSFRE.

DMSFRE: Obtains storage space to be used as a buffer.

DMSPRT: Determines whether the file to be printed is a library member or an input file.

DMSBRD: Reads a record; continues reading until the buffer is filled. When the buffer is filled, calls DMSPIO to issue the SIO instruction to begin the print operation.

DMSPIO: Issues the print SIO instruction and then calls DMSIOW to wait until the the I/O operation completes.

DMSIOW: Sets the wait bit for the virtual printer device and load the I/O old PSW from NUCON. This causes CMS to enter a wait state until the print operation completes.

DMSITI: Ensures that the interrupt is for the printer. If not, the I/O old PSW is reloaded, returning CMS to a wait state. If the interrupt is for the printer, DMSITI resets the WAIT bit in the I/O old PSW and loads that PSW, returning control to DMSICW.

DMSIOW: Places the symbolic name of the device in the last word of the PLIST and passes control to DMSPIO.

DMSPIO: Performs channel testing and handles errors. TIO instructions and sense SIO instructions are issued during the test processing. These operations are synchronized using DMSIOW and DMSITI in the manner described above. When the I/O completes successfully, control returns to DMSPRT.

DMSPRT: Determines whether all file records have been printed. If so, control returns to the caller. Otherwise, the address of the buffer is updated and more print operations are performed.

Printer Carriage Control Characters Used by DMSPIO

CMS supports the use of ASCII control characters and machine carriage control characters for the printed output. Part of the CMS implementation depends upon the fact that the set of ASCII control characters has almost nothing in common with the set of machine control characters. There are two exceptions to this, the characters X'C1' and X'C3'. These two characters, when interpreted as ASCII control characters, have the following meanings:

C1 = Skip to channel 10 before print.

C3 = Skip to channel 12 before print.

The same characters, when interpreted as machine control characters, have the following meanings:

C1 = Write, then skip to channel 8 after print.

C3 = Do not write, but skip to channel 8 immediately.

In printing lines containing carriage control characters, CMS has the capability of operating in two modes. In the first mode, which may be called ASCII control characters or machine control characters of either type are recognized and properly interpreted, except that the two conflicting characters are always interpreted as ASCII control characters. In the second mode, which may be called machine-only, only machine control characters are recognized, and the two conflicting characters are treated as machine.

The DMSPIO function uses a bit in the plist to indicate which of the two modes is in effect for printing.

The PRINTL macro always uses ASA control character or machine control character mode.

The PRINT command with the CC option always runs in ASCII control character or machine control character mode.

OS simulation output, which is used, for example, by the MOVEFILE command, uses the RECFM field in the DCB or in the FILEDEF command to determine which mode is to be used. If FA, VA, or UA is specified, then ASCII control character or machine control character mode is used. If FM, VM, or UM is specified, then machine-only mode is used. If no control character specification is included with the RECFM, then it is assumed that the output line begins with a valid data character, rather than with a control character, and single spacing is always used.

HANDLE INTERRUPTIONS

Figure 40 lists the CMS modules that process interruptions for CMS. CMS modules are described briefly in "CMS Module Description." SVC 9 interruption processing is described in "Maintaining an Interactive Console Environment."

DISK I/O IN CMS

Files residing on disk are read and written using DMSDIO. DMSDIO has two entry points: DMSDIOR, which is entered for a read I/O operation, and DMSDIOW, which is entered for a write operation.

The actual disk I/O operation is performed using the DIAGNOSE code 18 instruction. A return code of 0 from CP indicates a successful completion of the I/O operation. If the I/O is not successful, CP performs error recording, retry, recovery, or ABEND procedures for the virtual machine.

Read or Write Disk I/O

DMSDIO: Initializes the CCW to perform read operations.

DMSLAD: Obtains the address of the disk from which to read or write.

DMSDIO: Determines the size of the record to be read or written.

DMSFRE: Gets enough storage to contain the record if the request is for a record longer than 800 bytes.

DMSDIO: Reads records continually until all records for the file have been read.

DMSFRE: Returns the buffer to free storage if the record was longer than 800 bytes.

DMSDIO: Returns to the caller.

MANAGE CMS FREE STORAGE

DMSFRE handles requests for CMS free storage. The sections of CMS storage have the following uses:

- DMSNUC (X'00000' to approximately X'03000') - This is the nucleus constant area. It contains pointers, flags, and other data maintained by the various system routines.
- Low-core DMSFREE free storage area (approximately X'03000' to X'0E000') - This area is a free storage area, from which requests from DMSFREE are allocated. The top part of this area contains the file directory for the system disk (SSTAT). If there is enough room (as there will be in most cases), the FREETAB table also occupies this area, just below the SSTAT.
- Transient program area (X'0E000' to X'10000') - Because it is not essential to keep all nucleus functions resident in storage all the time, some of them are made "transient." This means that when they are needed, they are loaded from the disk into the transient program area. Such programs may not be longer than two pages, because that is the size of the transient area. (A page is 4096 bytes of virtual storage.)
- CMS nucleus (X'10000' to X'20000') - Segment 1 of storage contains the reentrant code for the CMS nucleus routines. In shared CMS systems, this is the protected segment. That is, this segment must consist only of reentrant code, and may not be modified under any circumstances. This fact implies certain system restrictions for functions which require that storage be modified, such as the fact that DEBUG breakpoints or CP ADSTOP commands cannot be placed in this segment, in a saved system.
- User program area (X'20000' to loader tables) - User programs are loaded into this area by the LOAD command. Storage allocated by means of the GETMAIN macro instruction is taken from this area, starting from the high address of the user program. In addition, this storage area can be allocated from the top down by DMSFREE, if not enough storage is available in the low-core DMSFREE storage area. Thus, the effective size of the user program area is reduced by the amount of free storage which has been allocated from it by DMSFREE.
- Loader tables (top pages of storage) - The top of storage is occupied by the loader tables, which are required by the CMS loader. These tables indicate which modules are currently loaded in the user program area (and the transient program area after a LOAD command). The size of the loader tables can be varied by the SET LDRTBLS command.

Types of Allocated Free Storage

Free storage can be allocated by means of the GETMAIN or DMSFREE macros.

Storage allocated by means of the GETMAIN macro is taken from the user program area, beginning with the high address of the user program.

Storage allocated by means of the DMSFREE macro can be taken from several areas:

First, DMSFREE requests are allocated from the low-address free storage area. If requests cannot be satisfied from there, they will be satisfied from the user program area.

In addition, requests are further broken down between requests for user storage and nucleus storage, as specified in the TYPE parameter of the DMSFREE macro. These two types of storage are kept in separate 4K pages. It is possible, if there are no 4K pages completely free in low storage, for no storage of one type to be available in low storage, while there is storage of the other type available there.

GETMAIN Free Storage Management Pointers

All GETMAIN storage is allocated in the user program area, starting from the end of the user's actual program. Allocation begins at the location pointed to by NUCON pointer MAINSTRT. The location MAINHIGH in NUCON is the pointer to the highest address of GETMAIN storage.

When the STRINIT macro is executed, both MAINSTRT and MAINHIGH are initialized to the end of the user's program, in the user program area. As storage is allocated from the user program area to satisfy GETMAIN requests, the MAINHIGH pointer is adjusted upward. Such adjustments are always in multiples of doublewords, so that this pointer is always on a doubleword boundary. As the allocated storage is released, this pointer is adjusted downward.

The pointer MAINHIGH can never be higher than FREELOWE, the pointer to the lowest address of DMSFREE storage allocated in the user program area. If a GETMAIN request cannot be satisfied without extending MAINHIGH above FREELOWE, GETMAIN takes an error exit, indicating that insufficient storage is available to satisfy the request.

The area between MAINSTRT and MAINHIGH may contain blocks of storage that are not allocated, and that are therefore available for allocation by a GETMAIN instruction. These blocks are chained together, with the first one pointed to by the NUCON location MAINLIST.

The format of an element on the GETMAIN free element chain is as follows:

```
        <------------ 4 bytes ------------>
        ┌─────────────────────────────────┐
        | FREPTR -- pointer to next free  |
  0(0)  |      element in the chain, or 0 |
        |      if there is no next element|
        ├─────────────────────────────────┤
        | FRELEN -- length, in bytes, of  |
  4(4)  |    this element                 |
        |                                 |
        ├─────────────────────────────────┤
        |  Remainder of this free element |
        .                                 .
        .                                 .
        .                                 .
```

DMSFREE Free Storage Pointers

The pointers FREEUPPR and FREELOWE in NUCON indicate the amount of storage which DMSFREE has allocated from the high portion of the user program area. These pointers are initialized to the beginning of the system loader tables.

The pointer FREELOWE is the pointer to the lowest address of DMSFREE storage in the user program area. As storage is allocated from the user program area to satisfy DMSFREE requests, this pointer is adjusted downward. Such adjustments are always in multiples of 4K, so that this pointer is always on a 4K boundary. As the allocated storage is released, this pointer is adjusted upward when whole 4K pages are completely free.

The pointer FREELOWE can never be lower than MAINHIGH, the pointer to the highest address of GETMAIN storage. If a DMSFREE request cannot be satisfied without extending FREELOWE below MAINHIGH, then DMSFREE takes an error exit, indicating that insufficient storage is available to satisfy the request.

The FREETAB free storage table is kept in free storage, usually just below the master file directory for the system disk. If there was no space available there, then FREETAB was allocated from the top of the user program area. This table contains one byte for each page of virtual storage. Each such byte contains a code indicating the use of that page of virtual storage. The codes in this table are as follows:

USERCODE (1): If the page is assigned to user storage.

NUCCODE (2): If the page is assigned to nucleus storage.

TRNCODE (3): If the page is part of the transient program area.

USARCODE (4): If the page is part of the user program area.

SYSCODE (5): If the page is none of the above.

In these cases, the page is assigned to system storage, system code, or the loader tables.

Other DMSFREE storage pointers are maintained in the DMSFRT control section, in NUCON. The most important fields there are the four chain header blocks.

Four chains of elements are not allocated to be associated with DMSFREE storage: The low-storage nucleus chain, the low-storage user chain, the high-storage nucleus chain, and the high-storage user chain. For each of these chains, exists a control block consisting of four words, with the following format:

```
        <------------ 4 bytes ------------->
        ┌──────────────────────────────────┐
        |POINTER -- pointer to the first   |
  0(0)  |   free element on the chain, or  |
        |   zero, if the chain is empty.   |
        |──────────────────────────────────|
        | NUM -- the number of elements on |
  4(4)  |        the chain.                |
        |                                  |
        |──────────────────────────────────|
        | MAX -- the value in this word is |
  8(8)  |   the size of the largest free   |
        |   element on the chain.          |
        |──────────────────────────────────|
        | FLAGS- | SKEY - | TCODE -| Unused|
 12(C)  | Flag   |Storage |FREETAB |       |
        | byte   | key    | code   |       |
        └──────────────────────────────────┘
```

These fields have the following meanings and uses:

POINTER This field points to the first element on this chain of free elements. If there are no elements on this free chain, then the POINTER field contains a zero.

NUM This field contains the number of elements on this chain of free elements. If there are no elements on this free chain, then this field contains a zero.

MAX This field is used for the purpose of avoiding searches which will fail. It contains the size, in bytes, of the largest element on the free chain. Thus, a search for an element of a given size will not be made if that size exceeds the MAX field.

FLAGS The following flags are used:

FLCLN (X'80')
Clean-up flag - This flag is set if the chain must be cleaned up. This is necessary in the following circumstances:

- If one of the two high-core chains contains a 4K page that is pointed to by FREELOWE, then that page can be removed from the chain, and FREELOWE can be increased.

- All completely non-allocated 4K pages are kept on the user chain, by convention. Thus, if one of the nucleus chains (low-core or high-core) contains a full page, then this page must be transferred to the corresponding user chain.

FLCLB(X'40')
Clobbered flag - Set if the chain has been destroyed.

FLHC (X'20')
High-core chain - Set for both the nucleus and user high-core chains.

FLNU (X'10')
Nucleus chain - Set for both the low-core and high-core nucleus chains.

FLPA (X'08')
Page available - This flag is set if there is a full 4K page available on the chain. Note that this flag may be set even if there is no such page available.

SKEY This one-byte field contains the storage key assigned to storage on this chain.

TCODE This one-byte field contains the FREETAB table code for storage on this chain.

Each element on the free chain has the following format:

```
       <------------ 4 bytes ------------>
       ┌─────────────────────────────────┐
       | POINTER -- pointer to the next  |
  0(0) |    element in the free chain    |
       |                                 |
       ├─────────────────────────────────┤
       | SIZE -- size of this free       |
  4(4) |    element, in bytes            |
       |                                 |
       ├─────────────────────────────────┤
       | Remainder of this free element  |
       .                                 .
       .                                 .
       .                                 .
```

When the user issues a variable length GETMAIN, the control program reserves 6 1/2 pages for CMS usage; this is a designed and set value. If the user wants more space, for example, for more directories, he should free up from the high end of storage some of the variable GETMAIN area.

As indicated in the illustration above, the POINTER field points to the next element in the chain, or contains the value zero if there is no next element. The SIZE field contains the size of this element, in bytes.

All elements within a given chain are chained together in order of descending storage address. This is done for two reasons:

1. Because the allocation search is satisfied by the first free element that is large enough, the allocated elements are grouped together at the top of the storage area, and prevent storage fragmentation. This is particularly important for high-storage free storage allocations, because it is desirable to keep FREELOWE as high as possible.

2. If free storage does become somewhat fragmented, the search causes as few page faults as possible.

As a matter of convention, completely nonallocated 4K pages are kept on the user chain rather than the nucleus chain. This is because requests for large blocks of storage are made, most of the time, from user storage rather than from nucleus storage. Nucleus requests need to break up a full page less frequently than user requests.

DMSFRE Method of Operation

A description of the algorithms which allocate and release blocks follows. The descriptions are based on the assumption that neither AREA=LOW nor AREA=HIGH was specified in the DMSFREE macro call. If either was specified, then the algorithm must be appropriately modified.

ALLOCATING USER FREE STORAGE: When DMSFREE with TYPE=USER (the default) is called, the following steps are taken to satisfy the request. As soon as one of the steps succeeds, then processing can terminate. DMSFRE:

1. Searches low-storage user chain for a block of the required size.

2. Searches the high-storage user chain for a block of the required size.

3. Extends high-storage user storage downward into the user program area, modifying FREELOWE in the process.

4. For fixed requests, there is nothing more to try. For variable requests, DMSFRE puts all available storage in the user program area onto the high-storage user chain, and then allocates the largest block available on either the high-storage user chain or the low-storage user chain. The allocated block is not satisfactory, if it is not larger then the minimum requested size.

ALLOCATING NUCLEUS FREE STORAGE: When DMSFREE with TYPE=NUCLEUS is called, the following steps are taken in an attempt to satisfy the request, until one succeeds. DMSFREE:

1. Searches the low-storage nucleus chain for a block of the required size.

2. Gets free pages from low-storage user chain, if any are available, and removes them to the low-storage nucleus chain.

3. Searches the highstorage nucleus chain for a block of the required size.

4. Gets free pages from the high-storage user chain, if they are available, and removes them to the highstorage nucleus chain.

5. Extends high-storage nucleus storage downward into the user program area, modifying FREELOWE in the process.

6. For fixed requests, there is nothing more to try. For variable requests, DMSFRE puts all available pages from the user chains and the user program area onto the nucleus chains, and allocates the largest block available on either the low-storage nucleus chains or the high-storage nucleus chains.

RELEASING STORAGE: When DMSFRET is called, the block being released is placed on the appropriate chain. At that point, the cleanup operation is performed, if necessary, to advance FREELOWE, or to move pages from the nucleus chain to the corresponding user chain.

Similar cleanup operations are performed, when necessary, after calls to DMSFREE, as well.

Relative Efficiency of DMSFREE Requests

The types of DMSFREE request in decreasing order of efficiency, are as follows:

1. User fixed storage requests, any size.

2. Nucleus fixed storage requests, for small blocks (less than one page in size).

3. Nucleus fixed storage request, for large blocks.

4. User variable storage requests. (Variable requests are no less efficient than fixed requests, if the maximum block size requested can be allocated.)

5. Fixed variable storage requests, if the maximum block size requested cannot be allocated.

Releasing Allocated Storage

STORAGE ALLOCATED BY GETMAIN: Storage allocated by the GETMAIN macro instruction may be released in any of the following ways:

- A specific block of such storage may be released by means of the FREEMAIN macro instruction.
- The STRINIT macro instruction releases all storage allocated by any previous GETMAIN requests.
- Almost all CMS commands call the STRINIT routine. Thus, executing almost any CMS command causes all GETMAIN storage to be released.

STORAGE ALLOCATED BY DMSFREE: Storage allocated by the DMSFREE macro instruction may be released in either of the following ways:

- A specific block of such storage may be released by means of the DMSFRET macro instruction.
- Whenever any user routine or CMS command abends (so that the routine DMSABN is entered), and the ABEND recovery facility of the system is invoked, all DMSFREE storage with TYPE=USER is released automatically.

Except in the case of ABEND recovery, storage allocated by the DMSFREE macro is never released automatically by the system. Thus, storage allocated by means of this macro instruction should always be released explicitly by means of the DMSFRET macro instruction.

DMSFRE Service Routines

The system uses the DMSFRES macro instruction to request certain free storage management services. The options and their meanings are as follows:

- INIT1--DMSINS calls this option to invoke the first free storage initialization routine, to allow free storage requests to access the system disk. Before this routine is invoked, no free storage requests may be made. After this routine has been invoked, free storage requests may be made, but these are subject to the following restraints until the second free storage management initialization routine has been invoked:

-- All requests for user storage are changed to requests for nucleus storage.

-- Only partial error checking is performed by the DMSFRET routine. In particular, it is possible to release a block that was never allocated.

-- All requests that are satisfied in high storage must be temporary, because all high storage allocated is released when the second free storage initialization routine is invoked.

When CP's saved system facility is used, the CMS system is saved at the point just after the system disk has been accessed. This means that it is necessary for DMSFRE to be used before the size of virtual storage is known, because the saved system can be used on any size virtual machine. Thus, the first initialization routine initializes DMSFRE so that limited functions can be requested, while the second initialization routine performs the initialization necessary to allow the full functions of DMSFRE to be requested.

- INIT2--This option is called by DMSINS to invoke the second initialization routine. This routine is invoked after the size of virtual storage is known, and it performs the initialization necessary to allow all the functions of DMSFRE to be used. The second initialization routine performs the following steps:

  -- Releases all storage that has been allocated in the highstorage area.

  -- Allocates the FREETAB free storage table. This table contains one byte for each 4096-byte page of virtual storage, and so cannot be allocated until the size of virtual storage is known. It is allocated in the low-address free storage area, if there is enough room available. If not, then it is allocated in the higher free storage area. For a 256K virtual machine, FREETAB contains 64 bytes; for a 16 million byte machine, it contains 4096 bytes.

  -- The FREETAB table is initialized, and all storage protection keys are initialized.

  -- All completely non-allocated 4K pages on the nucleus free storage chain are removed to the user chain. Any other necessary cleaning up operations are performed.

- CHECK--This option can be called at any time for system debugging purposes. It invokes a routine that performs a thorough check of all free storage chains for consistency and correctness. Thus, it checks to see whether any free storage pointers have been destroyed.

- CKON--This option turns on a flag which causes the CHECK routine described in the preceding paragraph to be invoked each time any call is made to DMSFREE or DMSFRET. This can be useful to pinpoint a problem that is, for example, destroying free storage management pointers. Care should be taken when using this option, because the CHECK routine is coded to be thorough rather than efficient. Thus, after the CKON option has been invoked, each call to DMSFREE or DMSFRET takes many times as long to be completed as before. This can impact the efficiency of system functions.

- CKOFF--Use of this option turns off the flag that was turned by the CKON option, described in the preceding paragraph.

- UREC--This option is called by DMSABN during the ABEND recovery process to release all USER storage.

- CALOC--This option is called by DMSABN after the ABEND recovery process has been completed. It invokes a routine that returns, in register 0, the number of doublewords of free storage that have been allocated. This figure is used by DMSABN to determine whether ABEND recovery has been successful.

Storage Protection Keys

In general, the following rule applies: system storage is assigned the storage key of X'F', while user storage is assigned the key of X'E'. This is the storage key associated with the protected areas of storage, not to be confused with the PSW or CAW key used to access that storage.

The specific key assignments are as follows:

- The NUCON area is assigned the key of X'F', with the exception of a half-page containing the OPSECT and TSOBLOKS areas, which has a key of X'E'.

- Free storage allocated by DMSFREE is broken up into user storage and nucleus storage. The user storage has a protection key of X'E', while the nucleus storage has a key of X'F'.

- The transient program area has a key of X'E'.

- The CMS nucleus code has a storage key of X'F'. In saved systems, this entire segment is protected by CP from modification even by the CMS system, and so must be entirely reentrant.

- The user program area is assigned the storage key of X'E', except for those pages which contain Nucleus DMSFREE storage. These latter pages are assigned the key of X'F'.

- The loader tables are assigned the key of X'F'.

<u>CMS System Handling of PSW Keys</u>

The CMS nucleus protection scheme protects the CMS nucleus from inadvertent destruction by a user program. This mechanism, however, does not prevent a user from writing in system storage intentionally. Because a CMS user can execute privileged instructions, he can issue a LOAD PSW (LPSW) instruction and load any PSW key he wishes. If a user defeats nucleus protection in this way there is nothing to prevent his program from:

- Modifying nucleus code

- Modifying a table or constant area

- Losing files by modifying a CMS file directory

In general, user programs and disk-resident CMS commands run with a PSW key of X'E', while nucleus code runs with PSW key of X'0'.

There are, however, some exceptions to this rule. Certain disk-resident CMS commands run with a PSW key of X'0', because they need to modify nucleus pointers and storage. On the other hand, the nucleus routines called by the GET, PUT, READ and WRITE macros run with a user PSW key of X'E', to increase efficiency.

Two macros, DMSKEY and DMSEXS, are available for changing the PSW key. The DMSKEY macro changes the PSW key to the user value or the nucleus value. DMSKEY NUCLEUS causes the current PSW key to be placed in a stack, and a value of 0 to be placed in the PSW key. DMSKEY USER causes the current PSW key to be placed in a stack, and a value of X'E' to be placed in the PSW key. DMSKEY RESET causes the top value in the DMSKEY stack to be removed and re-inserted into the PSW.

It is a CMS requirement when a routine terminates, that the DMSKEY stack must be empty. This means that a routine should execute a DMSKEY RESET macro instruction for each DMSKEY NUCLEUS macro instruction and each DMSKEY USER macro instruction executed by the routine.

The DMSKEY key stack has a maximum depth of seven for each routine. In this context, a "routine" is anything invoked by an SVC call. The DMSEXS ("execute in system mode") macro instruction is useful in situations where a routine is running with a user PSW key, but wishes to execute a single instruction with the nucleus PSW key. The single instruction may be specified as the argument to the DMSEXS macro, and that instruction is executed with a system PSW key.

<u>CP Handling For Saved Systems</u>

The explanation of saved system nucleus protection depends on the VSK, RSK, VPK and RPK:

1. Virtual Storage Key (VSK) - This is the storage key assigned by the virtual machine using the virtual SSK instruction.

2. Real Storage Key (RSK) - This is the actual storage key assigned by CP to the 2K page.

3. Virtual PSW Key (VPK) - This is the PSW storage key assigned by the virtual machine, by means of an instruction such as LPSW (Load PSW).

4. Real PSW Key (RPK) - This is the PSW storage key assigned by CP, which is in the real hardware PSW when the virtual machine is running.

When there are no shared segments in the virtual machine, then storage protection works as it does on a real machine. RSK=VSK for all pages, and RPK=VPK for the PSW.

However, when there is a shared segment (as in the case of segment 1 of CMS in the saved system), it is necessary for CP to protect the shared segment. For non-CMS shared systems, it does this by, essentially, ignoring the values of the VSKs and VPK, and assigning the real values as follows: RSK=0 for each page of the shared segment, RSK=F for all other pages, and RPK=F, always, for the real PSW. The SSK instruction is ignored, except to save the key value in a table in case the virtual machine later does an ISK to get it back.

For the CMS saved system, the RSKs and RPK are initialized as before, but resetting the virtual keys has the following effects:

- If the virtual machine uses an SSK instruction to reset a VSK, CP does the following: If the new VSK is nonzero, CP resets the RSK to the value of the VSK; if the new VSK is zero, CP resets RSK to F.

- If the virtual machine uses a LPSW (or other) instruction to reset the VPK, CP does the following: If the new VPK is nonzero, CP resets the RPK to the value of the VPK; if the new VPK is zero, CP resets RPK to F.

- If the VPK=0 and the RPK=F, storage protection may be handled differently. In a real machine, a PSW key of 0 would allow the program to store into any storage location, no matter what the storage key. But under CP, the program gets a protection violation, unless the RPK of the page happens to be F.

  Because of this, there is extra code in the CP program check handling routine. Whenever a protection violation occurs, CP checks to see if the following conditions hold:

  -- The virtual machine running is the saved CMS system, running with a shared segment.

-- The VPK = 0. The virtual machine is operating as though its PSW key is 0.

-- The RSK of the page into which the store was attempted is nonzero, and different from the RPK.

If any one of these three conditions fails to hold, then the protection violation is reflected back to the virtual machine.

If all three of these conditions hold, then the RPK (the real protection key in the real PSW) is reset to the RSK of the page into which the store was attempted.

EFFECT ON CMS: In CMS, this works as follows: CMS keeps its system storage in protect key F (RSK = VSK = F), and user storage in protect key E (RSK = VSK = E).

When the CMS supervisor is running, it runs in PSW key 0 (VPK = 0, RPK = F), so that CMS gets a protection violation the first time it tries to store into user storage (VSK = RSK = E). At that point, CP changes the RPK to E, and lets the virtual machine re-execute the instruction which caused the protection violation. There is not another protection violation until the supervisor goes back to storing into system-protected storage.

RESTRICTIONS ON CMS: There are several coding restrictions which must be imposed on CMS if it is to run as a saved system.

The first and most obvious one is that CMS may never modify segment 1, the shared segment, which runs with a RSK of 0, although the VSK = F.

A less obvious, but just as important, restriction, is that CMS may never modify with a single machine instruction (except MVCL) a section of storage which crosses the boundary between two pages with different storage keys. This restriction applies not only to SS instructions, such as MVC and ZAP, but also to RS instructions, such as STM, and to RX instructions, such as ST and STD, which may have nonaligned addresses on the System/370. An exception is the MVCL instruction which can be restarted after crossing a page boundary because the registers are updated when the paging exception occurs.

This restriction also applies to I/O instructions. If the key specified in the CCW is zero, then the data area for input may not cross the boundary between two pages with different storage keys.

OVERHEAD: It can be seen that this system is most inefficient when "storage-key thrashing" occurs -- when the virtual machine with a VPK of 0 jumps around, storing into pages with different VSK's.

Error Codes from DMSFREE, DMSFRES, and DMSFRET

A nonzero return code, upon return from DMSFRES, DMSFREE or DMSFRET, indicates that the request could not be satisfied. Register 15 contains this return code, indicating which error has occurred. The codes below apply to the DMSFRES, DMSFREE and DMSFRET macros.

Code	Error
1	DMSFREE -- Insufficient storage space is available to satisfy the request for free storage . In the case of a variable request, even the minimum request could not be satisfied.
2	DMSFREE or DMSFRET -- User storage pointers destroyed.
3	DMSFREE or DMSFRET -- Nucleus storage pointers destroyed.
4	DMSFREE -- An invalid size was requested. This error exit is taken if the requested size is not greater than zero. In the case of variable requests, this error exit is taken if the minimum request is greater than the maximum request. However, the error is not detected if DMSFREE is able to satisfy the maximum request.
5	DMSFRET -- An invalid size was passed to the DMSFRET macro. This error exit is taken if the specified length is not positive.
6	DMSFRET -- The block of storage which is being released was never allocated by DMSFREE. Such an error is detected if one of the following errors is found: a. The block is not entirely inside either the free storage area in low storage or the user program area between FREELOWE and FREEUPPR. b. The block crosses a page-boundary which separates a page allocated for user storage from a page allocated for nucleus type storage. c. The block overlaps another block already on the free storage chain.
7	DMSFRET -- The address given for the block being released is not a doubleword boundary.
8	DMSFRES -- An illegal request code was passed to the DMSFRES routine. Because all request codes are generated by the DMSFRES macro, this error code should never appear.
9	DMSFRE, DMSFRET, or DMSFRES -- An unexpected internal error occurred.

The DMSFRES Macro

CMS uses the DMSFRES macro to request special internal free storage management services. Use of this macro by non-system routines causes unpredictable results. The format is:

```
| label | DMSFRES | option |
```

where 'option' is one of the following:

INIT1 Performs the CMS system first initialization routine.

INIT2 Performs the CMS system second initialization routine.

CHECK Invokes a routine that checks the validity of all current free storage management pointers.

CKON Sets a flag that causes the CHECK to be invoked for each call to DMSFREE or DMSFRET.

CKOFF Turns off the above flag.

UREC Assists ABEND recovery, by releasing all USER-type DMSFREE storage allocations.

CALOC Assist ABEND recovery, by computing the total amount of allocated storage, excluding the system disk MFD and the FREETAB table.

For a full discussion of the meanings of these options, refer to "DMSFRE Service Routines."

The DMSKEY Macro

CMS uses the DMSKEY macro to modify the PSW storage protection key so that the nucleus code can store data into protected storage. The format is:

```
| [label] | DMSKEY | {NUCLEUS[,NOSTACK]|   |
|         |        |  USER[,NOSTACK]|      |
|         |        |  LASTUSER[,NOSTACK]|  |
|         |        |  RESET}               |
```

where:

NUCLEUS The nucleus storage protection key is placed in the PSW, and the old contents of the second byte of the PSW is saved in a stack. Use of this option allows the program to store into system storage, which is ordinarily protected.

USER The user storage protection key is placed in the PSW, and the old contents of the second byte of the PSW is saved in a stack. Use of this option prevents the program from inadvertently modifying nucleus storage, which is protected.

LASTUSER The SVC handler traces back through its system save areas for the active user routine closest to the top of the stack, and the storage key in effect for that routine is placed in the PSW. The old contents of the second byte of the PSW is saved in a stack. This option should be used only by system routines that should enter a user exit routine.

NOSTACK This option may be used with any of the above options to prevent the system from saving the second byte of the current PSW in a stack. If this is done, then no DMSKEY RESET need be issued later.

RESET The second byte of the PSW is changed to the value at the top of the PSW key stack, and removed from the stack. Thus, the effect of the last DMSKEY NUCLEUS or USER or LASTUSER request is reversed. This option should may not be used to reverse the effect of a DMSKEY macro for which the NOSTACK option was specified. A DMSKEY RESET macro must be executed for each DMSKEY NUCLEUS, USER or LASTUSER macro that was executed and that did not specify the NOSTACK option. Failure to observe this rule results in program abnormal termination.

The DMSEXS Macro

System commands running in user protect status use the DMSEXS macro to execute a single instruction with a system protect key in the PSW. This macro instruction can be used in lieu of two DMSKEY macros. The format is:

```
| [label] | DMSEXS | op-code,operands |
```

The op-code and the operands of the instruction to be executed must be given as arguments to the DMSEXS macro.

For example, execution of the sequence,

```
USING NUCON,0
DMSEXS OI,OSSFLAGS,COMPSWT
```

would cause the OI instruction to be executed with a zero protect key in the PSW. This sequence would turn on the COMPSWT flag in the nucleus. It would be reset with

```
DMSEXS NI,OSSFLAGS,255-COMPSWT
```

The instruction to be executed may be an EX instruction.

Register 1 cannot be used in any way in the instruction being executed.

SIMULATE NON-CMS OPERATING ENVIRONMENTS

The following contains descriptions for: access method support for non-CMS operating systems, CMS simulation of OS functions, and CMS implementation of DOS/VS functions.

ACCESS METHOD SUPPORT FOR NON-CMS OPERATING ENVIRONMENTS

OS Access Method Support

An access method governs the manipulation of data. To make the execution of OS generated code easier under CMS, the processing program must see data as OS would present it. For instance, when the processors expect an access method to acquire input source records sequentially, CMS invokes its sequential access method and passes data to the processors in the format that the OS access methods would have produced. Therefore, data appears in storage as if it had been manipulated using an OS access method. For example, block descriptor words (BDW), buffer pool management, and variable records are maintained in storage as if an OS access method had processed the data. The actual writing to and reading from the I/O device is handled by CMS file management.

The work of the volume table of contents (VTOC) and the data set control block (DSCB) is done by a master file directory (MFD) to maintain disk contents and a file status table (FST) for each data file. All disks are formatted in physical blocks of 800 bytes.

CMS continues to maintain the OS format, within its own format, on the auxiliary device, for files whose filemode number is 4. That is, the block and record descriptor words (BDW and RDW) are written along with the data. If a data set consists of blocked records, the data is written to and read from the I/O device in physical blocks, rather than logical records. CMS also simulates the specific methods of manipulating data sets.

To accomplish this simulation, CMS supports certain essential macros for the following access methods:

- BDAM (direct)--identifying a record by a key or by its relative position within the data set.
- BPAM (partitioned)--seeking a named member within an entire data set.
- BDAM/QSAM (sequential)--accessing a record in a sequence relative to
- VSAM (direct or sequential)--accessing a record sequentially or directly by key or address. CMS support of OS VSAM files is based on DOS/VS access method services and the virtual storage access method (VSAM). Therefore, the OS user is restricted to those services available under DOS/VS AMS and VSAM.

CMS SUPPORT FOR THE VIRTUAL STORAGE ACCESS METHOD

CMS simulation of OS and DOS includes support for the virtual storage access method (VSAM). The description of this support is in three parts:

- A description of the access method services program (AMSERV), which allows you to create and update VSAM files.
- A description of support for VSAM functions under CMS/DOS.
- A description of support for VSAM functions for the CMS OS simulation routines.

The routines that support VSAM reside in three discontiguous shared segments (DCSSs).

- The CMSAMS DCSS, which contains the DOS/VS AMS code to support AMSERV processing.
- The CMSVSAM DCSS, which contains actual DOS/VS VSAM code, and the CMS/VSAM OS interface program for processing OS VSAM requests.
- The CMSDOS DCSS, which contains the code that supports DOS requests under CMS.

Note: DMSVSR, which performs completion processing for CMS/VSAM support, resides in the CMS nucleus.

CREATING THE DOSCB CHAIN

The DLBL command creates a control block called a DOSCB in CMS free storage. The ddname specified in this DLBL command is associated with the ddname parameter in the program's ACB.

The DOSCB contains information defining the file for the system. The information in the DOSCB parallels the information written on the label information cylinder of a real DOS SYSRES unit, e.g. the name, and mode (volume serial number) of the data set, its logical unit specification, and its data set type (SAM or VSAM). The anchor for this chain is at location DOSFIRST in NUCON.

EXECUTING AN AMSERV FUNCTION

The CMS AMSERV command invokes the module DMSAMS, which is the CMS interface to the DOS/VS access method services (AMS) program. Module DMSAMS loads DOS/VS AMS code contained in the CMSAMS DCSS by means of the LOADSYS DIAGNOSE 64. The AMS code requires the services of DOS/VS code that resides in the CMSVSAM DCSS so that DCSS is also loaded via LOADSYS DIAGNOSE 64 when the VSAM master catalog is opened. Figure 51 shows the relationship in storage between the interface module DMSAMS and the CMSAMS and CMSVSAM DCSSs.

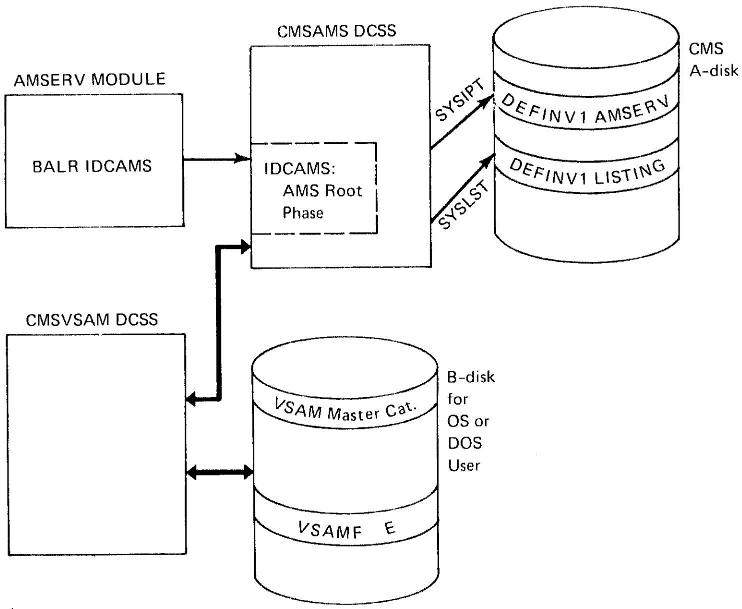


Figure 51. Relationship in Storage Between the CMS Interface Module DMSAMS and the CMSAMS and CMSVSAM DCSSs

The following is a general description of the DMSAMS method of operation.

DMSAMS first determines whether the user is in the CMS/DOS environment. If not, a SET DOS ON (VSAM) command is issued to load the CMSDOS segment and initialize the CMS/DOS environment. In this case, DMSAMS must also issue ASSGN commands for the disk modes in the DOSCB chain created by the OS user's DLBL commands. An ASSGN is also issued for SYSCAT, the VSAM master catalog.

DMSAMS then issues the ASSGN command for the SYSIPT and SYSLST files, assigning them to the user's A-disk. DLBL commands are then issued associating these units with files on the user's A-disk. Input to the AMSERV processor is the SYSIPT file, which has the filetype AMSERV. Output from AMSERV processing is placed in the SYSLST file, which has a filetype of LISTING.

DIAGNOSE 64 (LOADSYS) is then issued to load the CMSAMS DCSS, which contains the DOS/VS AMS code. A DOS/VS SVC 65 is issued to find the address of the DOS/VS AMS root phase, IDCAMS. When the SVC returns with the address of IDCAMS, a branch is made to IDCAMS, giving control to "live" DOS/VS routines.

IDCAMS expects parameters to be passed to it when it receives control. DMSAMS passes dummy parameters in the list labeled AMSPARMS.

After the root phase IDCAMS receives control, the functions in the file specified by the filename on the AMSERV command are executed.

In performing the functions requested in this file, AMS may require execution of DOS/VS VSAM phases located in the CMSVSAM DCSS. The CMSVSAM DCSS is loaded when AMS opens the VSAM catalog for processing.

On return from DOS/VS code, DMSAMS purges the CMSAMS DCSS, and issues DLBL commands for the SYSIPT and SYSLST files to clear the DOSCB's for these ddnames.

Control is then passed to DMSVSR, which purges the CMSVSAM DCSS. If the user program was not in the CMS/DOS environment when DMSAMS was entered, the SET DOS OFF command is issued by DMSVSR. Upon return from DMSVSR, DMSAMS performs minor housekeeping tasks and returns control to CMS.

EXECUTING A VSAM FUNCTION FOR A DOS USER

When a VSAM function, such as an OPEN or CLOSE macro, is requested from a DOS program, CMS routes control through the CMSDOS DCSS to the CMSVSAM DCSS, thus giving control to DOS/VS VSAM phases. Figure 52 shows the relationships in storage between the user program, the CMSDOS DCSS, and the CMSVSAM DCSS. The description below illustrates the overall logic of that control flow.

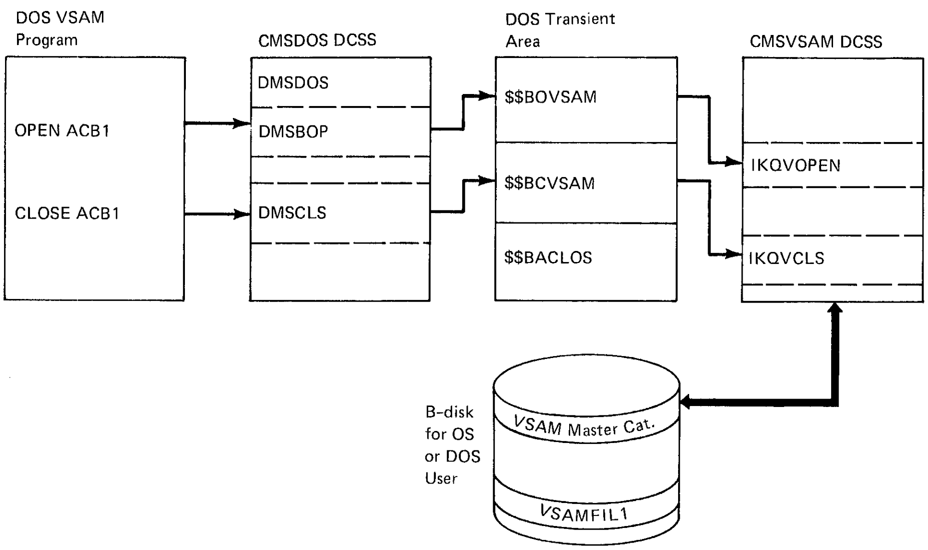


Figure 52. The Relationships in Storage Between the User Program and the CMSDOS DCSS and the CMSVSAM DCSSs

CMS/DOS SVC HANDLING

Module DMSDOS handles all CMS/DOS SVCs. There are four CMS/DOS routines that handle VSAM requests: DMSDOS, DMSBOP, DMSCLS, and DMSXCP. Within DMSDOS, several SVC functions support VSAM requests. These are described in "Simulating a DOS Environment Under CMS."

DMSDOS VSAM Processing

DMSDOS VSAM processing involves handling of SVC 65 (CDLOAD), which returns the address of a specified phase to the caller. DMSDOS searches both the shared segment table and the nonshared segment table for the CMSDOS and CMSVSAM segments, because both could be in use. Both of these segment tables contain the name of each phase comprising that segment followed by the fullword address of that phase within the segment.

During SVC 65 processing, DMSDOS checks to see if the address of IKQLAB is being requested. IKQLAB is the VSAM routine that returns the label information generated by DLBLs and EXTENT cards in DOS/VS systems. If this is the case, DMSDOS saves the address of IKQLAB in NUCON for later use by DMSXCP.

If VSAM has not been loaded, a DIAGNOSE 64 (LOADSYS) is issued to load the CMSVSAM DCSS.

DMSBOP VSAM Processing

When DMSBOP is entered to process ACBs, it checks to see if CMSVSAM is loaded. If VSAM has not been loaded, DIAGNOSE 64 is issued to load the CMSVSAM DCSS. DMSBOP then initializes the transient work area and issues a DOS OPEN via SVC 2 to bring the VSAM OPEN $$BOVSAM transient into the DOS transient area.

When VSAM processing completes, control returns to the user program directly.

DMSCLS VSAM Processing

DMSCLS processing is nearly the same as processing for DMSBOP. When DMSCLS is entered, it checks for an ACB to process. If there is one, the $$BCVSAM transient work area is initialized and SVC 2 is issued to FETCH the VSAM CLOSE transient $$BCVSAM into the DOS transient area. When the VSAM CLOSE routines complete processing, control returns to the user program, as in the case of OPEN.

DMSXCP VSAM Processing

When DMSXCP processes an EXCP request, it determines if the request is from IKQLAB (i.e. to read the SYSRES label information). If so, the label information area record is filled in from the appropriate DOSCB. (DMSXCP determines that the caller is IKQLAB by comparing the address of the caller with the address stored in NUCON by DMSDOS, as described above.)

EXECUTING A VSAM FUNCTION FOR AN OS USER

OS user requests for VSAM services are handled by DOS/VS VSAM code that resides in the CMSVSAM DCSS. To access this code, OS VSAM requests are intercepted by the CMS module DMSVIP, the interface between the OS VSAM requests and the CMS/DOS and DOS/VS VSAM routines.

Because DMSVIP is in the CMSVSAM segment, it is available only when that segment is loaded. Module DMSVIB, which resides in the CMS nucleus, is a bootstrap routine to load the CMSVSAM segment and pass control to DMSVIP.

DMSVIP receives control from VSAM request macros in three ways: via SVC (e.g. OPEN and CLOSE), via a direct branch using the address of DMSVIP in the ACB, and via a direct branch to the location of DMSVIP whose address is 256 bytes into the CMSCVT (CMSCVT is a CMS control block that simulates the OS CVT control block).

This last technique is used by the code generated from the OS VSAM control block manipulation macros (GENCB, SHOWCB, TESTCB, MODCB). That is, the address at 256 into CVT is assumed to be that of a control block that is at displacement X'12' has the address of the VSAM control block manipulation routine. To ensure that DMSVIP receives control from these requests, the address of DMSVIP is stored at 256 bytes into CMSCVT. However, until the CMSVSAM segment is loaded, the address at CMSCVT+256 is the address of module DMSVIB rather than the address of DMSVIP. The address of DMSVIP replaces that of DMSVIB when CMSVSAM is loaded. Both DMSVIB nd DMSVIP have pointers to themselves at 12 bytes into themselves to ensure that this technique works.

Figure 53 shows the relationships in storage between the user program, the OS simulation and interface routines, and the CMSDOS and CMSVSAM DCSSs.

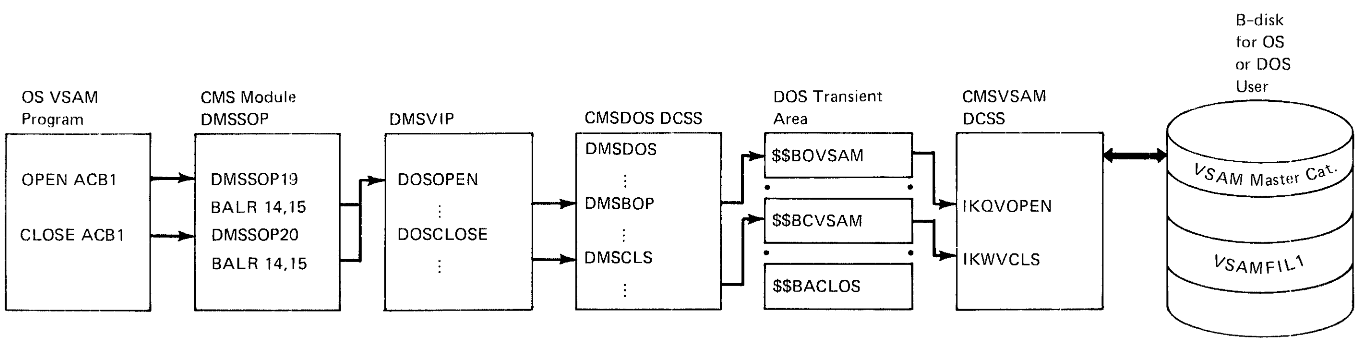


Figure 53. Relationship in Storage Between the User Program, the OS Simulation and Interface Routines, and the CMSDOS and CMSVSAM DCSSs.

The description below illustrates the overall logic of that control flow.

DMSVIP Processing

DMSVIP gains control from DMSSOP when an OS SVC 19, 20 or 23 (CLOSE TYPE=T) is issued. It also gains control on return from execution of a VSAM function, as described below. DMSVIP performs five main functions:

- Initializes the CMS/DOS environment for OS VSAM processing.
- Simulates an OS VSAM OPEN macro.
- Simulates an OS VSAM CLOSE macro.
- Simulates an OS VSAM control block manipulation macro (GENCB, MODCB, SHOWCB, or TESTCB).
- Processes OS VSAM I/O macros.

Initializing the CMS/DOS Environment for OS VSAM Processing

DMSVIP gets control when the first VSAM macro is encountered in the user program. Initialization processing begins at this time. The CMSDOS DCSS is loaded by issuing the command SET DOS ON (VSAM). ASSGN commands are also issued at this time according to the user-issued DCBL's as indicated in the DOSCB chain. Once this initialization completes, DMSVIP processes the VSAM request.

After the initialization, DMSVIP first checks to determine which VSAM function is being requested, OPEN, CLOSE, or a control block manipulation macro.

Simulate an OS VSAM OPEN

For OPEN processing, the DOSSVC bit in NUCON is set on and control passes to DMSBOP via SVC 2. Once the CMS/DOS routines are in control, execution of the VSAM function is the same as for the DOS VSAM functions described above.

On return from executing the OPEN routine, the address of another entry point to DMSVIP, at label DMSVIP2, is placed in the ACB for the data set just opened, the DOSSVC bit is turned off, and control is passed to DMSSOP, which returns to the user program. DMSVIP2 is the entry point for code that performs linkage to the VSAM data management phase IKQVSM. This is done after the first OPEN because it is assumed that, once opened, the user performs I/O for the phase, e.g. a GET or PUT operation.

When the linkage routine is entered, the DOSSVC bit is set on and control is given to the VSAM data management routine IKQVSM. On return from IKQVSM DMSVIP turns off the DOSSVC bit and returns control to the user program. (Refer to Simulate OS VSAM I/O Macros in this section.)

Simulate an OS VSAM CLOSE

For CLOSE processing, the DOSSVC bit is set on and control is passed to the CMS/DOS routine DMSCLS via SVC 2. As in the case of OPEN, once control passes to the CMS/DOS routine, execution of the VSAM function is the same as for the DOS VSAM functions described above.

On return from executing the VSAM CLOSE, the DOSSVC bit is turned off and control passes to DMSSOP, which returns to the user program.

Simulate OS VSAM Control Block Manipulation Macros

DMSVIP simulates the GENCB, MODCB, SHOWCB, and TESTCB control block manipulation macros.

GENCB PROCESSING: When a GENCB macro is issued with BLK=ACB or BLK=EXLST specified, the GENCB PLIST is passed unmodified to IKQGEN for execution. If GENCB is issued with BLK=RPL and ECB=address specified, the PLIST is rearranged to exclude the ECB specification, because DOS/VS does not support ECB processing. The GENCB PLIST is then passed to IKQGEN for execution.

MODCB, SHOWCB, AND TESTCB PROCESSING: When MODCB, SHOWCB, or TESTCB is issued, the OS ACB, RPL, and EXLST control blocks are reformatted, if necessary, to conform to DOS/VS formats.

For MODCB and SHOWCB, the requests are passed to IKQTMS for processing. When MODCB is issued with EXLST= specified, ensure that the exit routines return control to entry point DMSVIP3.

For TESTCB, check for any error routines the user may have specified. If the TESTCB specified RPL= and IO=COMPLETE, a not equal result is passed to the user. All other TESTCB requests are passed to DOS and the new PSW condition code indicates the results of the test.

If an error return is provided for TESTCB, the address of DMSVIP4 is substituted in the PLIST. This allows DMSVIP to regain control from VSAM so that the DOSSVC bit can be turned off. The error routine is then given control after the address is returned to the PLIST.

Simulate OS VSAM I/O Macros

DMSVIP simulates the OS GET, PUT, POINT, ENDREQ, ERASE, and CHECK I/O macros.

GET, PUT, POINT, ENDREQ, and ERASE Processing:

First, the OS request code in register 0 is mapped to a DOS/VS request code. The RPL or chain of RPLs is rearranged to DOS format (unless that has already been done).

If there is an ECB address in the OS RPL, a flag is set in the new DOS RPL and the ECB address is saved at the end of the RPL.

Asynchronous I/O processing is simulated by setting active exit returns inactive in the user EXLST. The exception to this is the JRNAD exit which need not be set inactive since it is not an error exit. Setting error exits to be inactive prevents VSAM from taking an error exit, thus allowing such an exit to be deferred until a CHECK can be issued for it.

The DOS macro is then issued via a BALR to IKQVSM.

DOS error codes returned in the RPL FDBK field that do not exist in OS are mapped to their OS equivalents. If the user has specified synchronous processing, this return code is passed unchanged in register 15.

For asynchronous processing, return codes are cleared before return and any exit routines set inactive are reactivated in the EXLST. Also, all ECBs are set to WAITING status.

CHECK PROCESSING: For CHECK processing, return codes in the RPL FDBK field are checked to determine the results of the I/O operation. If there is an active exit routine provided for the return code, control is passed to that routine. Also, all WAITING ECBs are posted with an equivalent completion code.

If no active exit routine is provided or if the exit routine returns to VSAM, the return code is placed in register 15 and control is returned to the instruction following the CHECK.

CMS/VSAM Error Return Processing

Two types of support for error routine processing are provided in DMSVIP. Entry point DMSVIP3 provides support for user exit routines; entry point DMSVIP4 provides support for ERET error returns.

USER EXIT ROUTINE PROCESSING: DMSVIP provides support for OS VSAM I/O error exits at entry point DMSVIP3. At this entry point the DOSSVC bit is turned off and the user storage key is restored.

The address of the user routine is recovered from VIP's saved exit list (either the primary exit list in the work area or the overflow exit list, OEXLSA).

Control then passes to the appropriate exit routine. If the routine is one that returns to VSAM, the DOSSVC flag is set ON and VSAM processing continues.

DMSVIP can save the addresses of up to 128 exit routines during execution of a user program.

ERET ERROR ROUTINE PROCESSING: DMSVIP provides support for OS VSAM ERET exit routines used in conjunction with the TESTCB macro. This support is located at entry point DMSVIP4. At DMSVIP4, the DOSSVC bit is turned off and the user storage key is restored. The address of the ERET routine is recovered from the work area and control passes to that routine.

The ERET routine may not return control to VSAM.

COMPLETION PROCESSING FOR OS AND DOS VSAM PROGRAMS

When an OS or DOS VSAM program completes, control is passed to module DMSVSR, which "cleans up" after VSAM. DMSVSR can be called from three routines after OS processing:

- DMSINT, if processing completes without system errors or serious user errors.
- DMSEXT, if the user program is used as part of an EXEC file.
- DMSABN, if there are system errors or the user program abnormally terminates.

After DOS VSAM processing completes, DMSVSR is called by DMSDOS.

DMSVSR issues an SVC 2 to execute the DOS transient routine $$BACLOS. $$BACLOS first checks for any OPEN VSAM files. If any are open, SVC 2 is issued to $$BCLOSE (DMSCLS) to close the files.

If there are no open files or if all ACB's have been closed, $$BACLOS issues SVC 2 to $$BEOJ4, an entry point in DMSVSR. At $$BEOJ4, a PURGESYS DIAGNOSE 64 is issued to purge the CMSVSAM DCSS. DMSVSR then checks to see if an OS program has completed processing. If this is the case, the SET DOS OFF command is issued and control returns to the caller.

OS SIMULATION BY CMS

When in a CMS environment, a processor or a user-written program is executing and utilizing OS-type functions, OS is not controlling this action, CMS is in control. Consequently, it is not OS code that is in CMS, but routines to simulate, in terms of CMS, certain OS functions essential to the support of OS language processors and their generated code.

These functions are simulated to yield the same results as seen from the processing program, as specified by OS program logic manuals. However, they are supported only to the extent stated in CMS documentation and to the extent necessary to successfully execute OS language processors. The user should be aware that restrictions to OS functions as viewed from OS exist in CMS.

Certain TSO Service routines are provided to allow the Program Products to run under CMS. The routines are the Command Scan and Parse Service Routines and the Terminal I/O Service Routines. In addition the user must provide some initialization as documented in TSO TMP Service Routine initialization. The OS functions that CMS simulates are shown in Figure 54.

TSO Service Routine Support

TSO macros that support the use of the terminal monitor program (TMP) service routines are contained in TSOMAC MACLIB. The macro functions are as described in the TSO TMP documentation with the exception of PUTLINE, GETLINE, PUTGET, and TCLEARQ.

Before using the TSO service routines, the calling program performs the following initialization:

1. Stores the address of the command line as the first word in the command processor parameter list (CPPL). The TSOGET macro puts the address of the CPPL in register 1.
2. Initializes CMS storage using the STRINIT macro.
3. Clears the ECT field that contains the address of the I/O work area (ECTIOWA).
4. Issues the STACK macro to define the terminal as the primary source of input.

CMS Simulation of OS Control Block Functions

Most of the simulated supervisory OS control blocks are contained in the following two CMS control blocks:

CMSCVT simulates the communication vector table (CVT). Location 16 contains the address of the CVT control section.

SVC Number	OS Macro Function	Simulation Routine	Comments
00	XDAP	DMSSVT	Reads or writes direct access volumes
01	WAIT	DMSSVN	Waits for an I/O completion
02	POST	DMSSVN	Posts the I/O completion
03	EXIT	DMSSLN	Returns from linked phase
04	GETMAIN	DMSSMN	Conditionally acquires user free storage
05	FREEMAIN	DMSSMN	Releases user-acquired free storage
06	LINK	DMSSLN	Links control to another load phase
07	XCTL	DMSSLN	Deletes, then links control to another load phase
08	LOAD	DMSSLN	Reads another load phase into storage
09	DELETE	DMSSLN	Deletes a loaded phase
10	GETMAIN/ FREEMAIN	DMSSMN	Manipulates free user storage
	GETPOOL	DMSSMN	Simulates an SVC10
11	TIME	DMSSVT	Gets the time of day
13	ABEND	DMSSAB	Terminates processing
14	SPIE	DMSSVT	Processes program interruptions
18	BLDL/FIND	DMSSVT	Manipulates simulated partitioned data files
19	OPEN	DMSSOP	Activates a data file
20	CLOSE	DMSSOP	Deactivates a data file
21	STOW	DMSSVT	Manipulates partitioned directories
22	OPENJ	DMSSOP	Activates a data file
23	TCLOSE	DMSSOP	Temporarily deactivates a data file
24	DEVTYPE	DMSSVT	Obtains device-type physical characteristics
25	TRKBAL	DMSSVT	Effective NOP
35	WTO/WTOR	DMSSVT	Communicates with the terminal
40	EXTRACT	DMSSVT	Effective NOP
41	IDENTIFY	DMSSVT	Adds entry to loader table
42	ATTACH	DMSSVT	Effective LINK
44	CHAP	DMSSVT	Effective NOP
46	TTIMER	DMSSVT	Accesses or cancels timer
47	STIMER	DMSSVT	Sets timer interval and timer exit routine
48	DEQ	DMSSVT	Effective NOP
51	SNAP	DMSSVT	Dumps specified storage areas
56	ENQ	DMSSVT	Effective NOP
57	FREEDBUF	DMSSVT	Releases a free storage buffer
60	STAE	DMSSVT	Allows processing program to decipher abend condition
62	DETACH	DMSSVT	Effective NOP
63	CHKPT	DMSSVT	Effective NOP
64	RDJFCB	DMSSVT	Obtains information from FILEDEF command
68	SYNAD	DMSSVT	Handles data set error conditions
69	BACKSPACE	DMSSVT	Backs up to the beginning of the previous record
–	GET/PUT	DMSSQS	Manipulates data records
–	READ/WRITE	DMSSBS	Manipulates data blocks
–	NOTE/POINT	DMSSCT	Accesses or changes relative track address
–	CHECK	DMSSCT	Tests ECB for completion and errors
93	TGET/TPUT	DMSSVN	Terminal processing
94	TCLEARQ	DMSSVN	Clears input queue
96	STAX	DMSSVT	Adds or deletes an attention exit level

Figure 54. OS Functions that CMS Simulates

CMSCB allocated from system free storage whenever a FILEDEF command or an OPEN (SVC 19) is issued for a data set. The CMS control block consists of the CMS file Control block (FCB) for the data file management under CMS, and simulation of the job file control block (JFCB), input/output block (IOB), and data extent block (DEB). The name of the data set is contained in the FCB, and is obtained from the FILEDEF argument list, or from a predetermined file name supplied by the processing problem program.

CMS also utilizes portions of the supplied data control block (DCB) and the data event control block (DECB). The TSO control blocks utilized are the command program parameters list (CPPL), user profile table (UPT), protected step control block (PSCB), and environment control table (ECT).

Operating System Simulation Routines

CMS provides a number of routines to simulate certain operating system functions by programs such as the Assembler and the FORTRAN and PL/I compilers. Some of the SVC simulation routines are located in the disk resident transient module DMSSVT. Whenever one of the SVC routines in DMSSVT or is invoked, that routine is loaded into the transient area. The following paragraphs describe how these simulation routines work.

XDAP-SVC 0: Writes and reads the source code spill file, SYSUT1, during language compilation for PL/I Optimizer and ANSI COBOL Compilers.

WAIT-SVC 1: Causes the active task to wait until one of more event control blocks (ECBs) have been posted. For each specified ECB that has been posted one is subtracted from the number of events specified in the WAIT macro. If the number of events is zero by the time the last ECB is checked control is returned to the user. If the number of events is not zero after the last ECB is checked and the number of events is not greater than the number of ECBs, the active task is put into a wait state until enough ECBs are posted to set the number of events at zero. When the event count reaches zero the wait bits are turn off in any ECBs that have not been posted and control is returned to the user. If the number of events specified is greater than the number of ECBs the system abnormally terminates with an error message. All options of WAIT are supported.

POST-SVC 2: Causes the specified event control block (ECB) to be set to indicate the occurrence of an event. This event satisfies the requirements of a WAIT macro instruction. All options of POST are supported. The bits in the ECB are set as follows:

Bit	Setting
0	0
1	1
2-7	Value of specified completion code

EXIT-SVC 3: This SVC is for CMS internal use only. It is used by the CMS routine DMSSLN to acquire an SVC SAVEAREA on return fron an executing program that had been given control by LINK (SVC 6), XTCL (SVC 7) or ATTACH (SVC 42).

GETMAIN-SVC 4: Control is passed to the GETMAIN entry point in the DMSSMN storage resident routine. The mode is determined: VU, VC, EC. A call is made to GETBLK to obtain the block of storage. Control blocks of two fullwords precede each section of available storage: (1) the address of the next block, (2) the size of this block. The head of the pointer string is located at the words MAINSTRT - initial free block, and MAINLIST - address of first link in chain of free block pointers. All options of GETMAIN are supported.

FREEMAIN-SVC 5: Releases a block of free storage. If the block is part of segmented storage, a control block of two fullwords is placed at the beginning of the released area. Adjustment is made to include this block in the chain of available areas. All options of FREEMAIN are supported.

LINK-SVC 6: Program transfer is controlled by the nucleus routine, DMSSLN. The LINK macro causes program control to be passed to a designated phase. If the COMPSWT bit within the byte OSSFLAGS is on, loading is done by calling LOADMOD to bring a CMS MODULE file into storage. If this flag is off, dynamic loading is initiated by calling LOAD. A GETMAIN is issued to obtain enough storage so that the loader (DMSLDR) may relocate the phase in storage. A chain of link request blocks is built to record the old SVC PSW, and the location and size of the phase storage area. If the routine is already in storage, determined by scanning the load request chain, no LOAD or LOADMOD is done. Control is passed directly to the routine. CMS ignores the DCB and HIARCHY options; all other options of LINK are supported.

XCTL-SVC 7: XCTL first deletes the current phase from storage. Processing then continues as for LINK-SVC 6, as previously described. CMS ignores The DCB and HIARCHY options; all other options of XCTL are supported.

LOAD-SVC 8: Control is passed to DMSSLN8 located in DMSSLN when a LOAD macro is issued. If the requested phase is not in storage, a LOAD or LOADMOD is issued to bring it in. Control is then returned to the caller. CMS ignores the DCB and HIARCHY options; all other options of LOAD are supported.

DELETE-SVC 9: Control is passed to DMSSLN9 located in DMSSLN when a DELETE macro is issued. Upon entry, DELETE checks to see whether the module specified was loaded using LOADMOD or dynamically loaded by LOAD or INCLUDE. If it was loaded by LOADMOD control is returned to the user. If it was dynamically loaded, the responsibility count is decremented by one and if it reaches zero, the storage is released using FREEMAIN, and control is returned to the user. All options of DELETE are supported. Code 4 is returned in register 15 if the phase is not found.

GETMAIN/FREEMAIN-SVC 10: Control is passed to the SVC 10 entry point in DMSSMN. Storage management is analogous to SVC 4 and 5, respectively. All options of GETMAIN and FREEMAIN are supported. Subpool specifications are ignored.

GETPOOL: Gets control via an OS LINK macro to IECQBFGI. IECQBFGI allocates an area of free storage using GETMAIN, sets up a buffer control block in the free storage, stores the address of the buffer control block in the DCB, and then returns control to the caller.

TIME-SVC 11: This routine (TIME) located in DMSSVT receives control when a TIME macro instruction is issued. A call is made (by SIO or DIAGNOSE) to the RPQ software chronological timer device, X'OFF'. The real time of day and date are returned to the calling program in a specified form: decimal (DEC) binary (BIN), or timer units (TU). All options of TIME except MIC are supported.

ABEND-SVC 13: This routine (DMSSAB) receives control when either an ABEND macro or an unsupported OS/360 SVC is issued. If an SVC 13 was issued with the DUMP option and either a SYSUDUMP or SYSABEND ddname had been defined via a call to DMSFLD (FILEDEF), a SNAP (SVC 51) specifying PDATA=ALL is issued to dump user storage to the defined file. A check is made to see if there are any outstanding STAE requests. If not, or if an unsupported SVC was issued, DMSCWR is called to type a descriptive error message at the terminal. Next, DMSCWT is called to wait until all terminal activity has ceased, and then, control is passed to the ABEND recovery routine. If a STAE macro was issued, a STAE work area is built and control is passed to the STAE exit routine. After the exit routine is complete, a test is made to see if a retry routine was specified. If so, control is passed to the retry routine. Otherwise, control passes to DMSABN unless the task that had the ABEND was a subtask. In that case, the resume PSW in the link block for the subtask is adjusted to point to an EXIT instruction (SVC 3). The EXIT frees the subtask, and the attaching task is redispatched.

SPIE-SVC 14: This routine (SPIE) receives control when a SPIE macro instruction is issued. When it gets control, SPIE inserts the new program interruption control area (PICA) address into the program interruption element (PIE). The program interruption element resides in the program interruption handler (DMSITP). It then returns the address of the old PICA to the calling program, sets the program mask in the calling program's PSW, and returns to the calling program. All options of SPIE are supported.

BLDL/FIND(Type D)-SVC 18: SVC to entry points in DMSSOP. If an OS disk is specified, DMSSVT branches and links to DMSROS. See BLDL and FIND under description of BPAM routines in DMSSVT.

STOW-SVC 21: See STOW under description of BPAM routines in DMSSVT.

OPEN/OPENJ-SVC 19/22: OPEN simulates the data management function of opening one or more files. It is a nucleus routine and receives control from DMSITS when an executing program issues an OPEN macro instruction. The OPEN macro causes an SVC to DMSSOP. DMSSOP simulates the OPEN macro. The DISP and RDBACK options are ignored by CMS; all other options of OPEN and OPENJ are supported.

CLOSE/TCLOSE-SVC 20/23: CLOSE and TCLOSE are simulated in the nucleus routine DMSSOP. It receives control whenever a CLOSE or TCLOSE macro instruction is issued. The CLOSE macro causes an SVC to DMSSOP. DMSSOP simulates the CLOSE macro. CMS ignores the DISP option; all other options of CLOSE and TCLOSE are supported.

DEVTYPE-SVC 24: This routine (DEVTYPE), located in DMSSVT, receives control when a DEVTYPE macro is issued. Upon entry, DEVTYPE moves Device Characteristic Information for the requested data set into a user specified area, and then returns control to the user. All options of DEVTYPE are supported.

TRKBAL-SVC 25: TRKBAL is a NOP located in DMSSVT.

WTO/WTOR-SVC 35: This routine (WTO), located in DMSSVT, receives control when either a WTO or a WTOR macro instruction is issued. For a WTO, it constructs a calling sequence to the DMSCWR function program to type the message at the terminal. (The address of the message and its length are provided in the parameter list that results from the expansion of the WTO macro instruction.) It then calls the DMSCWT function program to wait until all terminal I/O activity has ceased. Next, it calls the DMSCWR function program to type the message at the terminal and returns to the calling program. All options of WTO and WTOR are supported except those concerned with multiple console support.

For a WTOR macro instruction, this routine proceeds as described for WTO. However, after it has typed the message at the terminal it calls the DMSCRD function program to read the user's reply from the terminal. When the user replies with a message, it moves the message to the buffer specified in the WTOR parameter list, sets the completion bit in the ECB, and returns to the calling program.

EXTRACT-SVC 40: This routine (EXTRACT), located in DMSSVT receives control when an EXTRACT macro is issued. Upon entry, EXTRACT clears the user provided answer area and returns control to the user with a return code of 4 in register 15.

IDENTIFY-SVC 41: Located in DMSSVT, this routine creates a new load request block with the requested name and address if both are valid. The new entry is chained from the existing load request chain. The new name may be used in a LINK or ATTACH macro.

ATTACH-SVC 42: Located in DMSSLN, ATTACH operates like a LINK (SVC 6), with additional capabilities. The user is allowed to specify an exit address to be taken upon return from the attached phase; also, an ECB is posted when the attached phase has completed; and a STAI routine can be specified in case the attached phase abends. The DCB, LPMOD, DPMOD, HIARCHY, GSPV, GSPL, SHSPV, SHSPL, SZERO, PURGE, ASYNCH, and TASKLIB options are ignored; all other options of ATTACH are supported. Because CMS is not a multitasking operating system, a phase requested by the ATTACH macro must return to CMS.

CHAP-SVC 44: CHAP is a NOP located in DMSSVT.

TTIMER-SVC 46: Checks to ensure that the value in the timer (hex location 50) was set by an STIMER macro. If it was, the value is converted to an unsigned 32 bit binary number specifying 26 microsecond units and is returned in register 0. If the timer was not set by an STIMER macro a zero is returned in register 0, after setting register 0, the CANCEL option is checked. If it is not specified, control is returned to the user. If it is specified, the timer value and exit routine set by the STIMER macro are cancelled and control is returned to the user. All options of TTIMER are supported.

STIMER-SVC 47: Checks to see if the WAIT option is specified. If so, control is returned to the user. If not, the specified timer interval is converted to 13 microsecond units and stored in the timer (hex location 50). If a timer completion exit routine is specified, it is scheduled to be given control after completion of the specified time interval. If not, no indication of the completion of the time interval is scheduled. After checking and handling any specified exit routine address, control is returned to the user. All options of STIMER are supported. The TASK option is treated as though the REAL option had been specified.

DEQ-SVC 48: DEQ is a NOP located in DMSSVT.

SNAP-SVC 51: Control is passed to SNAP in DMSSVT when a SNAP macro is issued. SNAP fills in a PLIST with a beginning and ending address and calls DMPEXEC. DMPEXEC dumps the specified storage along with the registers and low storage to the printer. Control is then returned to SNAP and SNAP checks to see if any more addresses are specified. It continues calling DMPEXEC until all the specified addresses have been dumped to the printer. Control is then returned to the user. The DCB, SDATA, and PDATA options are ignored by CMS; all other options of SNAP are supported.

ENQ-SVC 56: ENQ is a NOP located in DMSSVT.

FREEDBUF-SVC 57: This routine (FREEDBUF) located in DMSSVT receives control when a FREEDBUF macro is issued. Upon entry, FREEDBUF sets up the correct DSECT registers and calls the FREEDBUF routine in DMSSBD. This routine returns the dynamically obtained buffer (BDAM) specified in the DECB to the DCB buffer control block chain. Control is then returned to the DMSSVT routine which returns control to the user. All the options of FREEDBUF are supported.

STAE-SVC 60: This routine (STAE) located in DMSSVT receives control when a STAE macro is issued. Upon entry, STAE creates, overlays or cancels a STAE control block (SCB) as requested. Control is then returned to the user with one of the following return codes in register 15:

Code	Meaning
00	An SCB is successfully created, overlaid or cancelled.
08	The user is attempting to cancel or overlay a non-existent SCB.

Format of SCB

```
 0 ┌─────────────────────────┐
   │0 or pointer to next SCB │
 4 ├─────────────────────────┤
   │exit address             │
 8 ├─────────────────────────┤
   │parameter list address   │
12 └─────────────────────────┘
```

DETACH-SVC 62: DETACH is a NOP located in DMSSVT.

CHKPT-SVC 63: CHKPT is a NOP located in DMSSVT.

RDJFCB-SVC 64: This routine (RDJFCB) receives control when a RDJFCB macro instruction is issued. When it gets control, RDJFCB obtains the address of the JFCB from the DCBEXLST field in the DCB and sets the JFCB to zero. It then reads the simulated JFCB located in CMSCB that was produced by issuing a FILEDEF into the closed area. RDJFCB calls the STATE function program to determine if the associated file exists. If it does, RDJFCB returns to the calling program. If the file does not exist, RDJFCB sets a switch in the DCB to indicate this and then returns to the calling program. RDJFCB is located in DMSSVT. All the options of RDJFCB are supported.

Note: The switch set by the RDJFCB is tested by the FORTRAN object-time direct-access handler (DIOCS) to determine whether or not a referenced disk file exists. If it does not, DIOCS initializes the direct access file.

SYNAD-SVC 68: Located in DMSSVT, SYNAD attempts to simulate the functions SYNADAF and SYNADRLS. SYNADAF expansion includes an SVC 68 and a high-order byte in register 15 denoting an access method. SYNAD prepares an error message line, swap save areas and register 13 pointers. The message buffer is 120 bytes: bytes 1-50, 84-119 blank; bytes 51-120, 120S INPUT/OUTPUT ERROR nnn ON FILE: "dsname"; where nnn is the CMS RDBUF/WRBUF error code. All the options of SYNAD are supported.

SYNADRLS expansion includes SVC 68 and a high order byte of X'FF' in register 15. The save area is returned, and the message buffer is returned to free storage.

BACKSPACE-SVC 69: Also in DMSSVT. For a tape, a BSR command is issued to the tape. For a direct access data set, the CMS write and read pointers are decremented by one. Control is passed to BACKSPACE in DMSSVT when a BACKSPACE macro is issued. BACKSPACE decrements the read write pointer by one and returns control to the user. No physical tape or disk adjustments are made until the next READ or WRITE macro is issued. All the options of BACKSPACE are supported.

TGET/TPUT-SVC 93: Located in DMSSVN, this routine receives control when a TGET or TPUT macro is issued. It is provided to support TSO service routines needed by program products. TGET reads a terminal line; TPUT writes a terminal line. The return code is zero if the operation was successful and a four if an error was encountered.

TCLEARQ-SVC 94: TCLEARQ is located in DMSSVN and causes the terminal input queue to be cleared via a call to DESBUF. At completion a return is made to the user.

STAX-SVC 96: Located in DMSSVT, STAX gets and chains a CMSTAXE control block for each STAX SVC issued with an exit routine address specified. The chain is anchored by TAXEADDR in DMSNUC. If no exit address is specified the most recently added CMSTAXE is cleared from the chain. If an error occurs during STAX SVC processing, a return code of eight is placed in register 15. The only option of STAX which may be specified is 'EXIT ADDRESS'.

GET/PUT: See the DMSSQS prolog for description.

READ/WRITE: OS READ and WRITE macros branch and link to DMSSBS. DMSSBS branches and links to DMSSEB and, if the disks is an OS disk, DMSSEB branches and link to DMSROS. See DMSSBS for description.

NOTE/POINT/FIND(type C): OS NOTE, POINT, and FIND (type c) macros branch and link to entry points in DMSSCT. If the disk is an OS disk, DMSSCT branches and links to DMSROS. See DMSSCT for descriptions.

CHECK: See the DMSSCT prolog for description.

Notes on using the OS simulation routines:

- CMS files are physically blocked in 800-byte blocks, and logically blocked according to a logical record length. If the filemode of the file is not 4, the logical record length is equal to the DCBLRECL and the file must always be referenced with the same DCBLRECL, whether or not the file is blocked. If the filemode of the file is 4, the logical record length is equal to the DCBBLKSI and the file must always be referenced with the same DCBBLKSI.

- When writing CMS files with a filemode number other than four, the OS simulation routines deblock the output and write it on a disk in unblocked records. The simulation routines delete each 4-byte block descriptor word (BDW) and each 4-byte record descriptor word (RDW) of variable length records. This makes the OS-created files compatible with CMS-created files and CMS utilities. When CMS reads a CMS file with a filemode number other than four, CMS blocks the record input as specifies and restores the BDW and RDW control words of variable length records.

  If the CMS filemode number is four, CMS does not unblock or delete BDWs or RDWs on output. CMS assumes on input that the file is blocked as specified and that variable length records contain block descriptor words and record descriptor words.

- To set the READ/WRITE pointers for a file at the end of the file, a FILEDEF command must be issued for the file specifying the MOD option.

- A file is erased and a new one created if the file is opened and all the following conditions exist:

  -- The OUTPUT or OUTIN option of OPEN is specified.

  -- The TYPE option of OPEN is not J.

  -- The dataset organization option of the DCB is not direct access or partitioned.

  -- A FILEDEF command has not been issued for data set specifying the MOD option.

- The results are unpredictable if two DCBs read and write to the same data set at the same time.

Command Flow of Commands Involving OS Access

ACCESS COMMAND FLOW: The module DMSACC gets control first when you invoke the ACCESS command. DMSACC verifies parameter list validity and sets the necessary internal flags for later use. If the disk you access specifies a target mode of another disk currently accessed, DMSACC calls DMSALU to clear all pertinent information in the old active disk table. DMSACC then calls DMSACF to bring in the user file directory of the disk. As soon as DMSACF gets control, DMSACF calls DMSACM to read in the master file directory of the disk. Once DMSACM reads the label of the disk, and determines that it is an OS disk, DMSACM calls DMSROS (ROSACC) to complete the access of the OS disk. Upon returning from DMSROS, DMSACM returns immediately to DMSACF, bypassing the master file directory logic for CMS disks. DMSACF then checks to determine if the accessed disk is an OS disk. If it is an OS disk, DMSACF returns immediately to DMSACC, bypassing all the user file directory logic for OS disks. DMSACC checks to determine if the accessed disk is an OS disk; if it is, another check determines if the accessed disk replaces another disk to issue an information message to that effect. Another check determines if you specified any options or fileid and, if you did, a warning message appears on the terminal. Control now returns to the calling routine.

FILEDEF COMMAND FLOW: DMSFLD gets control first when you issue a CMS FILEDEF command. DMSFLD adds, changes, or deletes a FILEDEF control block (CMSCB) and returns control to the calling routine.

LISTDS COMMAND FLOW: The module DMSLDS gets control first when you invoke the LISTDS command. DMSLDS verifies parameter list validity and calls module DMSLAD to get the active disk table associated with the specified mode. DMSLDS reads all format 1 DSCB and if you specified the PDS option and the data set is partitioned, DMSLDS calls DMSROS (ROSFIND) to get the members of the data set. After displaying the DSCB (or DSCB) on you console, DMSLDS returns to the calling routine.

MOVEFILE COMMAND FLOW: The module DMSMVE gets control first when you issue a CMS MOVEFILE command. DMSMVE calls DMSFLD to get an input and output CMSCB and, if the input DMSCB is for a disk file, DMSMVE calls DMSSTT to verify the existence of the input file and get default DCB parameters in absence of CMSCB DCB parameters. DMSMVE uses OS OPEN, FIND, GET, PUT, and CLOSE macros to move data from the input file to the output file. After moving the specified data, control returns to the calling routine.

QUERY COMMAND FLOW: The module DMSQRY gets control first when you invoke the QUERY command. DMSQRY verifies parameter list validity and calls DMSLAD to get the active disk table associated with the specified mode. DMSQRY displays all the information that you requested on your console. When DMSQRY finishes, control returns to the calling routine.

RELEASE COMMAND FLOW: The module DMSARE gets control first when you invoke the RELEASE command. DMSARE verifies parameter list validity and checks to determine if the disk you want to release is accessed. If the disk you want to release is currently active, DMSARE calls DMSALU to clear all pertinent information associated with the active disk. DMSALU first checks the active disk table for any existing CMS tables kept in free storage. If the disk you want to release is an OS disk, DMSALU does not find any tables associated with a CMS disk. If the disk is an OS disk, DMSALU releases the OS FST blocks (if any) and clears any OS FST pointers in the OS file control blocks. DMSALU then clears the active disk table and returns to DMSARE. DMSARE then clears the device table address for the specified disk and returns to the calling routine.

STATE COMMAND FLOW: The module DMSSTT gets control first when you invoke the STATE command. DMSSTT verifies the parameter list validity and calls module DMSLAD to get the active disk table associated with the specified mode. Upon return from DMSLAD, DMSSTT calls DMSLFS to find the file status table (FST) associated with the file you specified. Once DMSLFS finds the associated FST, it checks to determine if the file resides on an OS disk. If it does, DMSLFS calls DMSROS (ROSSTT) to read the extents of the data set. Upon return from DMSROS, DMSLFS returns to DMSSTT. DMSSTT then copies the FST (or OS FST) to the FST copy in statefst and returns to the calling routine.

OS Access Method Modules--Logic Description

DMSACC MODULE: Once DMSACC determines that the disk you want to access is an OS disk, it bypasses the routines that perform 'LOGIN UFD' and 'LOGIN ERASE'.

If the disk you want to access replaces an OS disk, message DMSACC724I appears at your terminal.

If you specified any options or fileid in the ACCESS command to an OS disk, a warning message, DMSACC230W, appears to notify you that such options or fileid were ignored. DMSACC returns to the calling routine with a warning code of 4.

DMSACF MODULE: DMSACF verifies that the disk you want to access is an OS disk and, if it is, exits immediately.

DMSACM MODULE: DMSACM saves the disk label and VTOC address in the ADT block if the disk is an OS disk. DMSACM checks to determine if a previous access to an OS disk loaded DMSROS. If not, DMSACM calls DMSSTT to verify that DMSROS text exists. Upon successful return from STATE, DMSACM loads DMSROS text into the high storage area with the same protect key and calls the OS access routine (ROSACC) of DMSROS to read the format 4 DSCB of the disk. Upon successful return from DMSROS, control returns to the calling routine. Any other errors are treated as general logon errors.

DMSALU MODULE: If the disk is an OS disk, DMSFRET returns the OS FST blocks (if any) to free storage. DMSALU clears the OS FST pointer in all active OS file control blocks, decrements the DMSROS usage count and, if the usage count is zero, clears the address of DMSROS in the nucleus area. DMSALU also calls DMSFRET to returns to free storage the area which DMSROS occupies.

DMSARE MODULE: DMSARE ensures that the disk you want to relase is an OS disk. DMSARE calls DMSALU to release all OS FST blocks and, if necessary, to free the area DMSROS occupies. Upon return from DMSALU, DMSARE clears the common CMS and OS active disk table.

DMSFLD MODULE

- DSN -- If you specify the parameter DSN as '?', FILEDEF displays the message DMSFLD220R to request you to type in an OS data set name with the format Q1.Q2.QN. Q1, Q2, and QN are the qualifiers of an OS data set name. If you specify the parameter DSN as Q1.Q2.QN, FILEDEF assumes that Q1, Q2, and QN are the qualifiers of an OS data set name, and stores the qualifiers with the format Q1.Q2.QN in a free storage block and chains the block to the FCB.
- CONCAT -- If you specify the CONCAT option, FILEDEF assumes that the specified FILEDEF is unique unless a filedef is outstanding with a matching ddname, filename, and filetype. This allows you to specify more than one FILEDEF for a particular ddname. The CONCAT option also sets the FCBCATML bit in the FCB to allow the OS simulation routine to know the FCB is for a concatenated MACLIB.
- MEMBER -- If you specify the member option, filedef stores the member name in FCBMEMBR in the FCB to indicate that the OS simulation routine should set the read/write pointer to point to the specified BPAM file member when OPEN occurs.

DMSLDS MODULE: DMSLDS saves the return register, sets itself with the nucleus protection key, clears the dsname key, and initializes its internal flag.

DMSLDS verifies parameter list validity. The data set name must not exceed 44 characters, and the disk mode (the last parameter before the options) must be valid. DMSLDS joins the quailifiers with dots (.) to form valid data set names. If you specify the data set name as a question mark (?), DMSLDS prompts you to enter the dsname in exactly the same form as the dsname which appears on the disk.

DMSLDS calls DMSLAD to find the active disk table block. If you specify filemode as an asterisk (*), DMSLAD searches for all ADT blocks. If you specify the filemode as alphabetic, DMSLAD finds only the ADT block for the specified filemode.

If you specify the dsname (which is optional), DMSLDS sets the channel programs to read by key. If you did not specify a dsname, DMSLDS searches the whole VTOC for format 1 DSCBS and displays all the requested information contained in the DSCB on your console. If you specify the format option, the RECFM, LRECL, BLKSI, DSORG, DATE, LABEL, FMODE, and data set name appear on you console; otherwise, only the FMODE and data set name appear.

If you specify the PDS option, DMSLDS calls the 'find' routine (rosfind) in DMSROS to read the member directory and pass back, one at a time, in the fcbmembr field of CMSCB the name of each member of the data set. This occurs if the data set is partitioned.

After processing finishes, DMSLDS resets the nucleus key to the same value as the user key, puts the return code in register 15, and returns to the calling routine.

DMSLFS MODULE: DMSLFS verifies that the FST being searched for has an OS disk associated with it. DMSLFS calls the DMSROS state routine (ROSSTT) to verify that the data set exists and CMS supports the data set attributes. Upon return from DMSROS, a return code of 88 indicates that the data set was not found, and DMSLDS starts the search again using the next disk in sequence. Any other errors, such as a return code 80, cause DMSLFS to exit immediately. A return code of 0 from DMSROS indicates that the data set is on the specified disk. From this point on, execution occurs common to both CMS and OS disks.

DMSMVE MODULE: If you specify the PDS option and the input is from a disk, DMSMVE sets the FCBMVPDS bit and issues an OS FIND macro before opening an output DCB to position the input file at the next member. DMSMVE then stores the input member name in the output CMSCB for use as the output filename. After reaching end-of-file on a member, the message DMSMVE225I appears, DMSMVE closes the output DCB, and passes control to find the next member. After moving all the members to separate CMS files, movefile displays message DMSMVE226I, closes the input and output DCBS, and returns control to the calling routine.

DMSROS MODULE:

- ROSACC Routine -- ROSACC gets control from DMSACM after DMSACM determines that the label of the disk belongs to an OS disk. The ROSACC routine reads the format 4 DSCB of the disk to further verify the validity of the OS disk. ROSACC updates the ADT to contain the address of the high extent of the VTOC (if the disk is a DOS disk) or the address of the last active format 1 DSCB (if the disk is an OS disk), and the number of cylinders in the disk. If the disk is a DOS disk, ROSACC sets a flag in the ADT. Information messages appear to notify you that the disk was accessed in read-only mode. If the disk is already accessed as another disk, another information message appears to that effect. Finally ROSACC zeroes out the ADTFLG1 flag in the ADT, sets the ADRFLG2 flag to reflect that an OS disk was accessed, and returns control to the calling routine.

- ROSSTT Routine -- Verifies the existence of an OS data set and verifies the support of the data set attributes.

  Note: Within the ROSSTT description, any reference to FCB or CMSCB implies a DOSCB if DOS is active.

  ROSSTT gets control from DMSSTT after DMSSTT determines that the STATE operation is to an OS disk. The ROSSTT routine searches for the correct FCB which a previous FILEDEF associated with the data set. If the DOS environment is active, ROSSTT locates the correct DOSCB that defines a data set described by a previous DLBL. If ROSSTT finds an active FST, control passes to ROSSTRET; otherwise, ROSSTT acquires the dsname block, places its address in the FCB, and moves the dsname in the FCB to the acquired block. ROSSTT acquires an FST block, chains it to the FST chain, and fills all general fields (dsname, disk address, and disk mode). ROSSTT now reads the format 1 DSCB for the data set and checks for unsupported options (BDAM, ISAM, VSAM, and read protect).

  Errors pass control back to the calling routine with an error code. ROSSTT groups together all the extents of the data set (by reading the format 3 DSCB if necessary) and checks them for validity. ROSSTT bypasses any user labels that may exist and displays a message to that effect. Next, ROSSTT moves the DSCB1 BLKSIZE, LRECL, and RECFM parameters to the OS FST and passes control to rosstret.

- ROSSTRET Routine -- If the disk is not a DOS disk, rosstret passes control back to the caller. If the specified disk is a DOS disk, rosstret fills in the OS FST BLKSIZE, LRECL, and RECFM fields that were not specified in the DSCB1. If the CMSCB fields are zero, rosstret defaults them to BLKSIZE=32760, LRECL=32670, and RECFM=U. Control then returns to the calling routine.

- ROSRPS Routine -- ROSRPS reads the next record of an OS data set. Upon entry to the ROSRPS entry point, ROSRPS calls CHKXTNT and, if the current CCHHR is zero, SETXTNT to ensure the CCHHR and extent boundaries are correctly set. ROSRPS then calls DISKIO and, if necessary, CHKSENSE and GETALT to read the next record. If no errors exist or an unrecoverable error occurred, control returns to the user with either a zero (I/O OK) or an 80 (I/O error) in register 15. If an unrecoverable error occurs, ROSRPS updates the CCWS and buffer pointers as necessary and recalls CHKXTNT and DISKIO to read the next record.

- ROSFIND Routine -- ROSFIND sets the CCHHR to point to a member specified in FCBMEMBR or, if the FCBMVPDS bit is on, sets the CCHHR to point to the next member higher than FCBMEMBR and sets a new member name in FCBMEMBR.

  Upon entry at the ROSFND entry point, ROSFND sets up a CCW to search for a higher member name if the FCBMVPDS bit is on, or an equal member name if the FCBMVPDS bit is off. It then calls SETXTNT, DISKIO and, if needed, CHKSENSE and GETALT to read in the directory block that contains the member name requested. After reading the block, it is searched for the requested member name. If the member name is not found, an error code 4 returns to the calling routine. If an I/O error occurs while trying to read the PDS block, an error code 8 returns to the calling routine. If the member name is found, TTRCNVRT is called to convert the relative track address to a CCHH and pass the address of the member entry to the calling routine.

- ROSNTPTB Routine -- ROSNTPTB gets the current TTR, sets the current CCHHR to the value of the TTR, and backspaces to the previous record.

  Upon entry at the ROSNTPTB entry point, ROSNTPTB checks to determine if a NOTE, POINT, or BSP operation was requested.

  If register 0 is zero, NOTE is assumed. The note routine calls CHRCNVRT to convert the CCHH to a relative track and returns control to the calling routine with the TTR in register 0.

  If register 0 is positive upon entry into DMSROS, POINT is assumed and ROSNTPTB loads a TTR from the address in register 0 and calls TTRCNVRT and SETXTNT to convert the TTR to a CCHHR. Then control returns to the calling routine.

  If register 0 is negative upon entry into DMSROS, BSP (BACKSPACE) is assumed. The backspace code checks to determine if the current position is the beginning of a track. If not, the backspace code decrements the record number by one and control then returns to the calling routine. If the current position is the beginning of a track, the backspace code calls CHRCNVRT to get the current CCHH. The backspace code then calls rdcnt to get the current record number of the last record on the new track, calls setxtnt to set the new extent boundaries, and returns control to the calling routine.

DMSSCT MODULE:

- NOTE Routine -- Upon entry to note, DMSSCT checks to determine if the DCB refers to an OS disk. If it does, DMSSCT calls DMSROS (ROSNTPTB) to get the current TTR. Control then returns to the user.

- POINT Routine -- Upon entry to point, DMSSCT checks to determine if the DCB refers to an OS disk. If it does, DMSSCT calls DMSROS (ROSNTPTB) to reset the current TTR, calls CKCONCAT and returns control to the calling routine.

- CKCONCAT Routine -- Upon entry to CKCONCAT, DMSSCT checks to determine if the FCB MACLIB CONCAT bit is on. If it is on, DCBRELAD+3 sets the correct OS FST pointer in the FCB and returns control to the calling routine. If the FCB MACLIB CONCAT bit is off, control returns to the calling routine.

- FIND (type_C) Routine -- If the DCB refers to an OS disk, DMSSCT calls DMSROS (ROSNTPTB) to update the TTR and control returns to the calling routine.

DMSSEB MODULE:

- EOBROUTN Routine -- If the FCB OS bit is on, control passes to OSREAD. Otherwise, if no special I/O routine is specified in FCBPROC, control passes to EOB2 in DMSSEB.

- OSREAD Routine -- DMSSEB calls DMSROS to perform a read or write and then control passes to EOBRETRN which, in turn, passes control back to DMSSBS. DMSSBS passes control back to the routine calling the read or write macro operation.

DMSSOP MODULE -- If the MACLIB CONCAT option is on in the CMSCB, OPEN checks the MACLIB names in the global list and fills in the addresses of OS FSTS for any MACLIBS on OS disks. The CMSCB of the first MACLIB in the global list merges and initializes CMSCBS.

If the CMSCB refers to a data set on an OS disk, DMSSOP checks to ensure that the data set is accessible and the DCB does not specify output, BDAM, or a key length. If any errors occur, error message DMSSOP036E appears and DMSSOP does not open the DCB. DMSSOP fills them in from the OS FST for the data set.

If the CMSCB fcbmembr field contains a member name (filled in by FILEDEF with the member option), DMSSOP issues an OS FIND macro to position the file pointer to the correct member. If an error occurs on the call to the FIND macro, error message DMSSOP036E appears and DMSSOP does not open the DCB.

DMSSVT MODULE:

- BSP (backspace) Routine -- Upon entry, backspace checks for the FCB OS bit. If it is on, the BSP routine calls DMSROS (ROSNTPTB) to backspace the TTR and control returns to the calling routine.

- FIND (type_D) Routine -- Upon entry to find, the find routine checks the FCB OS bit. If it is on, the FIND routine takes the OS FST address from the CMSCB or, if the CONCAT bit is on, from the global MACLIB list. The FIND routine then calls DMSROS (ROSFIND) to find the member name and TTR. DMSROS searches for a matching member name or, if the FCBMVPDS option is specified, a higher member name. If the DMSROS return code is 0 or 8, or if the FCBCATML bit is not on, control returns to the calling routine with the return code from DMSROS. If the return code is 4 and the FCBCATML bit is on, DMSSVT checks to determine if all the global MACLIBS were searched. If they were, control returns to the calling routine with the DMSROS return code. If they were not, DMSSVT issues the FIND on the next MACLIB in the global list.

- BLDL Routine--BLDL list = FF LL NAME TTR KZC DATA

  If the DCB refers to an OS disk, the BLDL routine fills in the TTR, C-byte and data field from the OS data set.

DMSQRY MODULE:

- SEARCH Routine -- The search routine ensures that any OS disk currently active is included in the search order of all disks currently accessible.

- DISK Routine -- The disk routine displays the status of any or all OS disks using the following form:

'MODE(CUU): (NO. CYLS.), TYPE R/O - OS.'

__DMSSTT__ __MODULE__ -- DMSSTT verifies that the disk being searched is an OS disk. DMSSTT calls DMSLFS to get the FST associated with the data set. Upon return from DMSLFS, DMSSTT checks the return code to ensure that CMS supports the data set attributes. A return code of 81 or 82 indicates that CMS does not support the data set and message DMSSTT229E occurs to that effect. DMSSTT then clears the FST copy with binary zeros, and moves the filename, filetype, filemode, BLKSIZE, LRECL, RECFM, and flag byte to the FST copy. From this point on, common code execution occurs for both CMS and OS disks.

__Routines Common to All of DMSROS__

- CHRCNVRT Routine -- The CHRNCVRT routine converts a CCHH address to a relative track address.
- CHKSENSE Routine -- CHKSENSE checks sense bits to determine the recoverability of a unit check error if one occurs.
- CHKXTNT Routine -- CHKXTNT checks to determine if the end of split cylinder or the end of extent occurred, and, if so, updates to the next split cylinder or extent.
- DISKIO Routine -- DISKIO starts I/O operation on a CCW string via a DIAGNOSE X'20'.
- GETALT Routine -- GETALT switches reading from alternate track to prime track, and from prime track to alternate track.
- RDCNT Routine -- RDCNT reads count fields on the track to determine the last record number on the track.
- SETXTNT Routine -- SETXTNT sets osfstend to the value of the end of the extent and, if a new extent is specified, sets CCHHR to the value of the start of the extent.

SIMULATING A DOS ENVIRONMENT UNDER CMS

CMS/DOS is a functional enhancement to CMS that provides DOS installations with the interactive capabilities of a VM/370 virtual machine. CMS/DOS operates as the background DOS partition; the other four partitions are unnecessary, since the CMS/DOS virtual machine is a one-user machine.

CMS/DOS provides read access to real DOS data sets, but not write or update access. Real DOS private and system relocatable, source statement, and core-image libraries can be read. This read capability is supported to the extent required to support the CMS/DOS linkage editor, the DOS/PLI and DOS/VS COBOL compilers, the FETCH routine, and the RSERV, SSERV, and ESERV commands. No read or write capability exists for the DOS procedure library, except for copying procedures from the procedure library (via the PSERV command) or displaying the procedure library (via the DSERV command).

CMS/DOS does not support the standard label cylinder.

INITIALIZING DOS AND PROCESSING DOS SYSTEM CONTROL COMMANDS

Initialization of the CMS/DOS operating environment requires the setting of flags and the creation of certain data areas in storage. Once initialized, these flags and data areas may then be changed by routines invoked by the system control commands.

Five modules are described in this section:

- DMSSET Activates the CMS/DOS environment control blocks to be used during CMS/DOS processing.
- DMSOPT Sets or resets compiler execution-time options.
- DMSASN Relates logical units to physical units.
- DMSLLU Lists the assignments of CMS/DOS physical units.
- DMSDLB Associates a DTF with a logical unit for CMS/DOS processing.

__DMSSET--Initializing the CMS/DOS Operating Environment__

DMSSET initializes the CMS/DOS operating environment as follows:

- Verifies that the mode, if specified, is for a DOS formatted disk.
- Stores appropriate data in the SYSRES LUB and PUB.
- Locates and loads the CMS/DOS discontiguous shared segment. Saves (in NUCON) the addresses of the two major CMS/DOS data blocks, SYSCOM, BGCOM, and the address of the CMS/DOS discontiguous shared segment (CMSDOS).
- Sets the DOSMODE and DOSSVC bits in DOSFLAGS in NUCON.
- Assigns (via ASSGN) the SYSLOG logical unit as the CMS virtual console.

The CMS/DOS operating environment is entered when the CMS SET DOS ON command is issued, invoking the module DMSSET.

Data Areas Prepared for Processing During CMS/DOS Initialization

Several data areas are prepared for processing during initialization. The main CMS data area, NUCON, is modified to contain the addresses of two DOS data areas, SYSCOM and BGCOM.

The SYSCOM DSECT is the DOS system communications region. It consists mainly of address constants, including the addresses of the AB option table, the PUB ownership table, and the FETCH table. It also includes such information as the number of partitions (always one for CMS/DOS) and the length of the PUB table.

The BGCOM DSECT is the partition communication region. It includes such information as the date, the location of the end of supervisor storage, the end address of the last phase loaded, the end address of the longest phase loaded, bytes used to set the language translator and supervisor options, and the addresses of many other DOS data areas such as the LUB, PUB, NICL, FICL, PIB, PIB2TAB, and the PCTAB.

The LUB and PUB tables are also made available during initialization. The LUB is the logical unit block table. It acts as an interface between the user's program and the CMS/DOS physical units. It contains an entry for each symbolic device available in the system.

Each of the symbolic names in the LUB is mapped into an element in the PUB, the physical unit block table. The PUB table contains an entry for each channel and device address for all devices physically available to the system and also contains such information as device type code, CMS disk mode, tape mode setting, and 7-track indicator.

Two bits are set in DOSFLAGS in NUCON, DOSMODE and DOSSVC. DOSMODE specifies that this virtual machine is running in the CMS/DOS operating environment. DOSSVC indicates whether OS or DOS SVCs are operative in the operating environment. If DOSSVC is set, DOS SVCs are used; otherwise, OS SVCs are operative.

SETTING OR RESETTING SYSTEM ENVIRONMENT OPTIONS

Once the CMS/DOS environment is initialized, the flags and control blocks set during initialization can be modified and manipulated to perform the functions specified by commands entered at the console. This section describes the modules that set and reset the system environment options. That is, they set those options that control compiler execution and that control the configuration of logical and physical units in the system.

DMSOPT--Setting and Resetting Compiler Options

The CMS/DOS OPTION command invokes module DMSOPT, which sets either the default options for the compiler or the options specified on the command line. The nonstandard language translator options switch and the job duration indicator byte are altered. Options are set using two control words located in the partition communication region (BGCOM). Bits in bytes JCSW3 or JCSW4 are set, depending on the options specified.

DMSASN--Associate System or Programmer Logical Units with Physical Units

Module DMSASN is invoked when the ASSGN command is entered. DMSASN first scans the command line to ensure that the logical unit being assigned is valid for the physical unit specified (for example, SYSLOG must be assigned to either the virtual console or the virtual printer). Once the command line is checked, PUB and LUB entries are modified to reflect the specified assignment.

For the PUB entry, the device type is determined (via DIAG 24) and the device type code is placed in the PUB. Other modifications are made to the PUB depending on the specified assignment. The LUB entry is then mapped to its corresponding PUB.

DMSLLU--List the Assignments of CMS/DOS Logical Units

The function of DMSLLU is to request a list of the physical units assigned to logical units. It performs this function by referencing information located in the CMS/DOS data blocks, specifically SYSCOM, LUB, and PUB. Another data block, the next in class (NICL) table is also referenced.

The information on the command line is scanned and the appropriate items are displayed at the user's console. If an option (EXEC or APPEND) is specified, an EXEC file is created ($LISTIO EXEC A1) to contain the output. If EXEC is specified, any existing $LISTIO EXEC A1 file is erased and a new one is created. If APPEND is specified, the new file is appended to the existing file.

DMSDLB--Associate a DTF Table Filename with a Logical Unit

DMSDLB is invoked when the CMS/DOS DLBL command is entered. DMSDLB associates a DTF (Define The File) table filename with a logical unit. This function is performed by creating a control block called a DOSCB, which contains information defining a DOS file used during job execution. DLBL is valid only for sequential or VSAM disk devices.

This information parallels the label information written on a real DOS SYSRES unit under DOS/VS. The DOSCB contains such information as the name, type, and mode of the referenced dataset, its device type code, its logical unit specification, and its dataset type (SAM or VSAM).

A DOSCB is created for each file specified by the user during a terminal session. The DOSCBs are chained to each other and are anchored in NUCON at the field DOSFIRST. The chain remains intact for the entire session, unless an ABEND occurs or the user specifically clears an entry in the the DOSCB chain. A given DOSCB is accessed when an OPEN macro is issued from an executing user program.

The overall logic flow for DMSDLB is as follows:

1. Scans the command line to ensure that any options entered are valid (i.e. anything to the right of the open parenthesis).
2. Processes the first operand (ddname or *). When ddname is specified, loop through the DOSCB chain to find a matching ddname. If none is found, DMSDLB calls DMSFRE to get storage to create a new DOSCB for this file. The old copy of the DOSCB is then saved so that, in case of errors during processing, it can be retrieved intact. The new copy of the DOSCB contains updates and DOSCB replaces the old copy if there are no errors.
3. The mode specification is checked to ensure that it is a valid mode letter; if the file is a CMS file, the mode letter must specify a CMS disk. If DSN has been specified, the mode letter must be for a non-CMS disk.
4. Process each option on the command line appropriately.
5. If EXTENT or MULT is specified, a separate block of free storage is obtained to contain information about the extent, for example, a block is obtained to contain the DOS data set name.
5. Check for errors. If there are errors, any blocks created during processing are purged and an error message is issued. If there are no errors, restore the old block, which has been modified to reflect current processing, and return control to DMSITS.

PROCESS CMS/DOS OPEN AND CLOSE FUNCTIONS

The CMS/DOS OPEN routines are invoked in response to DOS OPEN macros. They operate on DTF (define the file) tables and ACB (access method control block) tables created when the DTFxx and ACB macros are issued from an executing user program. These tables contain information such as the LOG unit specification for the file, the DTF type of the file, the device code for the file, and so forth. The information in the tables varies depending on the type of DTF specified, i.e. the table generated by a unit record DTF macro is slightly different than the table generated by a DTF disk macro.

Five routines are invoked to perform OPEN functions, DMSOPL, DMSOR1, DMSOR2, DMSOR3, and DMSBOP. DMSCLS performs the CLOSE function.

Opening Files Associated With DTF Tables

Depending on the type of OPEN macro issued from a user program, one of five CMS/DOS OPEN routines could be invoked. OPENR macros give control to DMSOR1 and, depending on the DTF type specified, DMSOR2 or DMSOR3 may be invoked. These three routines (DMSOR1,DMSOR2, and DMSOR3) request the relocation of a specified file. DMSOPL is invoked by the DOS/VS compilers when they need access to a source statement library. These routines are mainly interface routines to DMSBOP, which performs the main function of opening the specified file. Each of the routines calls DMSBOP.

DMSBOP is the CMS/DOS routine that simulates the DOS/VS OPEN function. The basic function of DMSBOP is the initialization of DTF tables, i.e. setting fields in specified DTFs for use by the DOS/VS LIOCS routines.

When a DOS problem program is compiling, a list of DTFs and ACBs is built. At execution time, this list is passed to DMSBOP. The logic flow of DMSBOP is as follows:

1. Scans the list of DTF and ACB addresses, handling each iteam in the list in line. When the OPEN macro expands, register 1 points to the name of the $$B transient to receive control ($$BOPEN) and register 0 points to the list of DTF/ACB addresses to be opened.
2. When an ACB is encountered in the table, control is passed directly to the VSAM OPEN routine, $$BOVSAM. The VSAM routine is responsible for opening the file and returning control to DMSBOP.
3. When a DTF is encountered in the table, DMSBOP itself handles the OPEN:
   a. For reader/punch files (DTFCD), the OPEN bit in the DTF table is turned on.
   b. For printer files (DTFPR), if two IOAREAs are specified, the IOREG is loaded with the address of the appropriate IOAREA. Next, the PUB index byte associated with the logical unit specified in the DTF is checked to ensure that a physical device has been assigned and the PUB device code is then analyzed. The OPEN bit in the DTF table is then turned on.
   c. For console files (DTFCN), no OPEN logic is required.

d. For tape files (DTFMT), the PUB device type code must specify TAPE. If an IOREG is specified (for output tapes only), the address of the appropriate IOAREA is placed in it. For input files, there is separate processing for tapes with standard label, nonstandard label, and no label. For output tapes, both tape data files and work tape files are treated as no label tapes.

e. For disk files (DTFxx), the LUB is verified to ensure that the logical unit has been assigned. A check is made to ensure that the DOSCB exists for the DTF filename. For disk output files, the address of the appropriate IOAREA is placed in IOREG. For disk input files, the existence of the file is verified via a call to DMSSST. Also, EXTENT information is initialized and the OPEN bit is posted.

f. DTFDT and DTFCP are separate DTF types that could describe any of the above devices.

4. After all files in the table have been opened, DMSBOP returns control to the problem program via SVC 11.

5. If errors are encountered during DMSBOP processing, an error message is issued and return is made via SVC 6.

Closing Files Associated With DTFs

The CMS/DOS routine that processes CLOSE requests is DMSCLS, whose logic is analogous to that of DMSBOP, the OPEN routine described above: when CLOSE expands, register 1 points to $BCLOSE and register 0 points to the list of DTF/ACB addresses. The same table containing DTFs and ACBs used to open files is also used to close those files. Each entry in the table is processed as it occurs, with control passing to a VSAM CLOSE routine ($$BCVSAM) when an ACB is encountered. The OPEN bit is then turned off.

PROCESS CMS/DOS EXECUTION-RELATED CONTROL COMMANDS

The CMS/DOS FETCH and DOSLKED commands simulate the operation of the DOS/VS fetch routines and the DOS/VS Linkage Editor. The three CMS modules that perform this simulation are:

- DMSFET--Provide an interface to interpret the DOS FETCH command line and execute the phase, if START is specified on the command line.
- DMSFCH--Bring into storage a specified phase from a system or private core-image library or from a CMS DOSLIB library.
- DMSDLK--Link edit the relocatable output of the CMS/DOS language translators to create executable programs.

DMSFET and DMSFCH--Bring a Phase into Storage for Execution

The DOS/VS FETCH function is simulated by CMS modules DMSFET and DMSFCH. The main control block used during a FETCH operation is FCHSECT, which contains addressing information required for I/O operations.

The FETCH command line invokes module DMSFET. This module first validates the command line and issues a FILEDEF for the DOSLIB file. It then issues a FILEDEF for a DOSLIB file. DMSFET then issues a DOS SVC 4, which invokes the module DMSFCH to perform the actual FETCH operation.

DMSFCH first determines where the phase to be fetched resides. The search order is private core-image library, DOSLIB, system core-image library. If the phase is not found in any of these libraries, DMSFCH assumes that the FETCH is for a phase in a system or private core-image library. To find a DOSLIB library member, OS OPEN and FIND macros are issued (SVC 19 and 18).

When the member is found, OS READ and CHECK macros are issued to read the first record of the file (the member directory). This record contains the number of text blocks and the length of the member.

All addressing information is stored in FCHSECT and the text blocks that the phase are read into storage. If the read is from a CMS disk, issue the OS READ and CHECK macros to read the data. If the read is from a DOS disk, first determine whether this is the first read for the DOS discontiguous shared segment (DCSS). If this is the case, CCW information is relocated to ensure that the DCSS code is reentrant. For all reads for a DOS disk, a CP READ DIAG instruction is issued. When the entire file is read, it is relocated (if it is relocatable).

If a DOSLIB is open, close it using an OS SVC 20 and return control to DMSFET. DMSFET then checks to see whether START is specified and, if so, an SVC 202 is issued for the CMS START command to execute the loaded file.

When all FETCH processing is complete, control returns to the CMS command handler, DMSITS.

Simulate the Functions of the DOS/VS Linkage Editor: DMSDLK

CMS simulation of the DOS/VS Linkage Editor function directly parallels the DOS/VS implementation of that function. For detailed information on the logic of the function, see the publication _DOS/VS Linkage Editor Logic_, Order No. SY33-8556.

Note that the modules comprising the DOS/VS Linkage Editor are prefixed by the letters IJB and are separate CSECTs. ALL of these CSECTs have counterparts contained within the one CMS module, DMSDLK. They are treated as subroutines within that module, but perform the same functions as their independent DOS/VS counterparts and have been named using the same naming conventions as for the DOS/VS CSECTs. For example, the IJBESD CSECT in DOS/VS is paralleled by the CMS DMSDLK subroutine DLKESD.

A brief dscription of the logic follows. The CMS/DOS DOSLKED command invokes the module DMSDLK, which is entered at subroutine DLKINL. DLKINL performs initialization and is later overlaid by the text buffer and the linkage editor tables. DLKINL starts to read from a DOSLNK file and processes ACTION statements, if there are any.

On encountering the first non-ACTION card (or if there is no DOSLNK file), the main flow is entered. Depending on the input on the DOSLNK or the TEXT file, records from either of those files may be read or records from a relocatable library may be read. The type of card image read determines the subroutine to which control is given for further processing.

An ENTRY card indicates the end of the input to the linkage editor. At this point, a map is produced by subroutine DLKMAP. DLKRLD is then entered to finish the editing of object modules by relocating the address constants. If the phases are to be relocatable, relocation information is added to the output on the DOSLIB. Updating of the DOSLIB library is performed by DLKCAT using the OS STOW macro.

A significant deviation from DOS/VS code is the use of OS macros, in some instances, rather than DOS/VS macros. To take advantage of CMS support of partitioned data sets, the OS OPEN, FIND, READ, CHECK, and CLOSE macros are issued rather then their DOS/VS counterparts.

SIMULATE DOS SVC FUNCTIONS

All SVC functions supported for CMS/DOS are handled by the CMS module DMSDOS. DMSDOS receives control from DMSITS (the CMS SVC handler) when that routine intercepts a DOS SVC code and finds that the DOSSVC flag in DOSFLAGS is set in NUCON.

DMSDOS acquires the specified SVC code from the OLDPSW field of the current SVC save area. Using this code, DMSDOS computes the address of the routine where the SVC is to be handled.

Many CMS/DOS routines (including DMSDOS) are contained in a discontiguous shared segment (DCSS). Most SVC codes are executed within DMSDOS, but some are in separate modules external to DMSDOS. If the SVC code requested is external to DMSDOS, its address is computed using a table called DCSSTAB; if the code requested is executed within DMSDOS, the table SVCTAB is used to compute the address of the code to handle the SVC.

The items below show the SVCs supported by CMS/DOS simulation routines, the name of the macro that invokes a given SVC code, the CMS module that executes the code, and a brief statement describing how the SVC function is performed.

<u>SVC 0</u>: <u>EXCP</u> -- Handled by module DMSXCP...reads from CMS or DOS/VS formatted disks. CCWs are converted to appropriate CMS I/O requests, for example, RDBUF/WRBUF, CARDRD/CARDPH. The CCB is posted (indicating I/O completion) using CMS return information. If a non-zero return code is returned, a CANCEL is performed. I/O requests to DOS disks are handled using CP DIAGNOSE instructions.

<u>SVC 1</u>: <u>FETCH</u> -- Handled by DMSFCH...loads a problem program phase into core and executes it, if execution is requested. For details on how FETCH works, see the section "Bring a Phase into Storage for Execution: DMSFET and DMSFCH."

<u>SVC 2</u>: <u>FETCH</u> -- Handled by DMSFCH...loads a $$$$B-Transient phase into core and executes it, if execution is requested. For details on how FETCH works, see the section "Bring a Phase into Storage for Execution: DMSFET and DMSFCH."

<u>SVC 4</u>: <u>FETCH</u> -- Handled by DMSFCH...loads a problem program phase into user storage and executes it, if execution is requested. For details on how FETCH works, see the section "Bring a Phase into Storage for Execution: DMSFET and DMSFCH."

<u>SVC 5</u>: <u>MVCOM</u> -- Handled by DMSDOS...provides the user with a way of altering bytes 12 through 23 of the partition communication region (BGCOM). Checks to ensure that the specified field is correct length and then moves the information to the specified field.

<u>SVC 6</u>: <u>CANCL</u> -- Handled by DMSDOS...cancels a CMS/DOS session. Processing depends on value in register 15 on entry; if above 256 the request is from a system program. If below 256, request is from a user program. Processing continues with control passing to EOJ code, described below.

<u>SVC 7</u>: <u>WAIT</u> -- Handled by DMSDOS...informs system programs to wait for a system event to take place before processing can continue. WAIT is an effective NOP for CMS/DOS.

<u>SVC 8</u>: Handled by DMSDOS...temporarily returns control to a problem program. The address of the problem to which control is being passed is contained in register 0. This address is stored in the SVC save area OLDPSW field and control is passed to the CMS SVC handler (DMSITS).

<u>SVC 9</u>: Handled by DMSDOS...returns control to system program (i.e. a user program has been given control, as in the case of SVC 8, and must return control to the system routine, a $$$$B-Transient routine, that called it).

<u>SVC 11</u>: Handled by DMSDOS...returns control to a problem program from a $$$$-B transient routine. Uses the SVC save area OLDPSW field to return to the calling program.

<u>SVC 12</u>: Handled by DMSDOS...resets flags in the linkage control byte of the Partition Communication Region (BGCOM) to zero; also, provides the user the capability to use a mask to set the value of this same byte. In both

cases, the SVC routine that handles the request performs an AND operation to accomplish the function.

SVC 14: EOJ -- Handled by DMSDOS...normally terminates execution of a problem program. Clears control blocks and resets control words.

SVC 16: Handled by DMSDOS...establishes linkage with or terminates linkage to a user's program check routine. Locates the appropriate PC option table entry. If contents of register 0 is zero, terminates linkage: stores a zero into the routine address field of the PC option table. If register 0 is non-zero, the address of the PC routine and the save area address is passed to the STXIT macro. If a STXIT PC routine is already active, the complement of the new routine address is placed in the PC option table; if no STXIT PC routine is active, both the new routine address and the save area address are placed in the PC option table.

SVC 17: Handled by DMSDOS...provides supervisory support for the exit macro. Locates appropriate PC option table entry and restores user's registers and PSW. Stores the address of the PC routine in the PC option table and returns to the next sequential address in the interrupted program.

SVC 26: Handled by DMSDOS...validates address limits. Checks the limits passed in registers 1 and 2 and either returns control to the caller or writes an error message.

SVC 33: COMRG -- Handled by DMSDOS...provides the address of the partition communication region (BGCOM). Returns the address of BGCOM in register 1.

SVC 34: Handled by DMSDOS...supports the GETIME macro. Updates the date field in the partition communications region (BGCOM).

SVC 37: Handled by DMSDOS...establishes linkage to or terminates linkage from a user's abonormal termination routine. Locate the AB table entry. If register 0 ccontains zeros, terminates linkage: if the AB routine is active, stores zeros into the routine address field of the AB option table. If the AB routine is not active, stores zeros into both the routine address field and the save area field of the AB option table.

If register 0 is non-zero, establishes linkage: passes the address of the AB routine and the save area address to the STXIT AB macro. If STXIT AB is active, the complement of the AB routine address is stored in the AB option table. If STXIT AB is not active, both the address of the new AB routine and the address of the save area are placed in the option table.

SVC 40: POST -- Handled by DMSDOS...signals the completion of a system event.

SVC 50: Handled by DMSDOS...issues an error message and terminates the command. Issued by a LIOCS routine when that routine is requested to perform a function it could not perform.

SVC 61: GETVIS -- Handled by DMSDOS...used by VSAM to obtain scratch storage; also, obtains storage for a relocatable VSAM routine. Storage is obtained from the user free storage area and the address of the storage is returned in Register 1.

SVC 62: FREEVIS -- Handled by DMSDOS...returns storage obtained by a GETVIS. Address of the area to be returned is pointed to by Register 1.

SVC 63: USE -- Handled by DMSDOS...VSAM uses SVC 63 to ensure that system resources are updated serially, so that two or more attempts to modify the same data at the same time do not succeed. A table of counters (RURTBL) is kept for system resources. These counters are posted when a request is made for system resources. If a resource is already in use, a return code of eight is placed in gister 0. If the resource is available, a zero is returned in Register 0.

SVC 64: RELEASE -- Handled by DMSDOS...VSAM uses SVC 64 to release a system resource obtained via USE SVC. The appropriate counter in RURTBL is decremented by one each time a resource is released.

SVC 65: CDLOAD -- Handled by DMSDOS...loads a relocatable VSAM phase into storage unless that phase has already been loaded.

If an anchor table is available, it is searched for the phase. If the phase is found, its load point, entry point, and length are returned in registers 0, 1, and 14, respectively, and register 15 contains zeros.

If the phase is not found in the anchor table, DMSFCH is called to search for it. If the phase is found in the discontiguous shared segment, return is made to the requestor as above.

If the phase was found, but not loaded, storage is ontained for it via the GETVIS SVC. DMSFCH is called again to load the phase into the storage just obtained. An anchor table is then built in the user area (unless one already exists) and return to the caller is then made as described above.

SVC 66: RUNMODE -- Handled by DMSDOS...determines whether the problem program is running in real or virtual mode. Register 0 contains zero on return if the program is running in virtual mode.

SVC 75: SECTVAL -- Handled by DMSDOS...used by VSAM I/O routines to obtain a sector number for 3330 or 3340 devices. The appropriate sector value is calculated from input supplied in ser registers 1 and 0. The sector number (from 0 to 127) is returned in register 0.

Certain DOS SVCs are treated as no-ops by CMS/DOS and other DOS/VS SVCs are not supported. These are listed below.

SVCS TREATED AS NO-OP BY CMS/DOS

SVC	Action
10:	Sets timer interval
18:	STXIT (IT)
20:	Establishes linkage to OC
22:	Seizes (interruption enable/disable)
24:	Sets timer interval
35:	Holds a track
36:	Frees a track
41:	Dequeues a resource
42:	Enqueues a resource
52:	Returns remaining timer interval (Register 0 is also cleared)
67:	PFIX, fixes pages in real storage
68:	PFREE, frees pages in real storage
71:	SETPFA
85:	RELPAG
86:	FCEPGOUT
87:	PAGEIN

SVCS NOT SUPPORTED BY CMS/DOS: The following SVCs cause an error message to be generated and are treated as a CANCL (SVC 6).

SVC	Action
3:	Forces dequeue
13:	Sets switches in BGCOM
15:	Heads queue and executes channel program
19:	Returns from user's IT
23:	Loads phase header
25:	Issues HIO
27:	Special HIO
28:	Returns from user's MR
29:	Multiple WAITM support
30:	Waits for a QTAM element
31:	Posts a QTAM element
32:	Reserved for IBM use
38:	Initializes a subtask
39:	Terminates a subtask
43:	Reserved for IBM use
44:	External unit checks record
45:	Emulator interface
46:	OLTEP in supervisor state
47:	Multiple WAITF support
48:	Fetches a CRT trans
49:	Reserved by IBM
51:	Returns phase header
53:	Reserved by IBM
54:	Frees real page frames
55:	Gets real page frames
56:	Gets or frees PUB of POWER device
57:	Makes POWER dispatchable
58:	Interface between JCL and supervisor
59:	Interface between EOJ and supervisor
60:	EREP and CRT I/O areas address
69:	REALAD
70:	VIRTAD
72:	GETCBUF/FREECBUF
73:	SETAPP
74:	Fixes pages in real storage for restart
76:	Initializes for recording of RMSR I/O error
77:	TRANSCSW
78:	Reserved for IBM use
79:	Reserved for IBM use
80:	Reserved for IBM use
81:	Reserved for IBM use
82:	Reserved for IBM use
83:	Reserved for IBM use
84:	Reserved for IBM use
88 and up:	Reserved for IBM use

PROCESS CMS/DOS SERVICE COMMANDS

DMSSRV--Copies books from a system or private source statement library to a specified output device.

DMSPRV--Copies DOS procedures from a DOS system procedure library to a specified output device.

DMSRRV--Copies modules from a system or private relocatable library to a specified output device.

DMSDSV--Lists the directories of DOS private or system libraries.

DMSDSL--Deletes members (phases) of a DOSLIB library; compresses a DOSLIB library; lists the members (phases) of a DOSLIB library.

ESERV--De-edits, displays or punches, verifies, and updates edit assembler macros from the source statement library.

TERMINATE PROCESSING THE CMS/DOS ENVIRONMENT

DMSBAB--Gives control to an abnormal termination routine once linkage to such a routine has been established via the STXIT AB macro.

DMSITP--Processes program interrupts and SPIE exits.

DMSDMP--Simulates the $$BDUMP and $$BPDUMP routines; issues a CP DUMP command directing the dump to an offline printer.

PERFORM MISCELLANEOUS CMS FUNCTIONS

CMS BATCH FACILITY

The CMS Batch Facility is a function of CMS. It provides a way of entering individual user jobs through an active CMS machine from the virtual card reader rather than from the console. The batch facility reissues the IPL command after each job.

The CMS Batch Facility consists of two modules: DMSBTB, the bootstrap routine (a nonrelocatable CMS module file) and DMSBTP, the processor routine (a relocatable CMS text file that runs free storage).

GENERAL OPERATION OF DMSBTB

The bootstrap module, DMSBTB, loads the processor routine DMSBTP and the user exit routines BATEXIT1 and BATEXIT2 (if they exist) into free storage.

DMSBTB first ensures that DMSINS (CMS initialization) has set the BATRUN and BATLOAD flags on in the CMS nucleus constant area indicating that either an explicit batch initial program load command has been issued or that the CMSBATCH command has been issued immediately after initial program load has taken place. If not, error message DMSBTB101E is typed and the batch console returns to a normal CMS interactive environment. STATE (DMSSTT) is then called to confirm the existence of the processor file DMSBTP TEXT. If the file does not exist, error message DMSTBT100E is typed and the batch console returns to the CMS interactive environment.

Using the "state" copy of the file status table (FST) for DMSBTP, DMSBTB computes the size of DMSBTP TEXT file by multiplying the logical record length by the number of logical records (no DS constants). A free storage request is made for the size of DMSBTP and the address of the routine is then stored at ABATPROC in the NUCON area of the CMS nucleus.

The existence of the user exit routines is determined by STATE. If they exist, their sizes are included in the request for free storage.

The free storage address is translated into graphic hexadecimal format and the CMS LOAD command is issued to load the DMSBTP TEXT file into the reserved free storage area. The user exit routines, BATEXIT1 TEXT and BATEXIT2 TEXT are also loaded at this time. If these files do not exist, an unresolved external reference error code is returned by the loader, but is ignored by DMSBTB because these routines are optional. If an error (other than unresolved names) occurs, error message DMSBTB101E is typed and the batch console returns to the CMS interactive environment.

The loader tables are searched for the address of the ABEND entry point DMSBTPAB in the loaded batch processor. When the entry is found, its address and that of entry DMSBTPLM are stored in ABATABND and the ABATLIMT respectively, in the NUCON area of the CMS nucleus. If the ABEND entry point is not found in the tables, error message DMSBTB101E is typed and the batch console returns to the CMS interactive environment.

The BATLOAD flag is set off to show that DMSBTP has been loaded, the BATNOEX flag is set on to prevent user job execution until DMSBTP encounters a /JOB card and finally, control is returned to the command processor DMSINT.

If an error message is issued, DMSERR is called to type the message, and the BATRUN and BATLOAD flags are set off before control is returned to CMS. This allows the normal CMS interaction to resume.

GENERAL OPERATION OF DMSBTP

The batch processor module DMSBTP simulates the function of the CMS console read module DMSCRD. This is accomplished by issuing reads to the virtual card reader, formatting the card-image record to resemble a console record and returning control to CMS to process the command (or data) request. DMSBTP also performs reads to the console stack if the stack is not empty, checks for and processes the /JOB card, ensuring that it is the first record in the user job, traps all CP commands to maintain system integrity and performs job initialization, cleanup, and job recovery.

Upon receiving control, DMSBTP checks the BATCPEX flag in NUCON. If the flag is set on, control was received from DMSCPF and a branch is made to the CP trap routine to verify that the command is allowable under batch. The function of that routine is described later. If the BATCPEX flag is off, control was received from DMSCRD (console read module) and DMSBTP checks for finished reads in the real batch console stack. If the number of finished reads is not zero, control is returned to DMSCRD to process the real console finished (stacked) reads. If the number of finished reads is zero, a record is read from the batch virtual card reader into the CARD buffer via an SVC call to CARDRD (DMSCIO). The record in the CARD buffer is typed on the console via the WRTERM macro. If the BATMOVE flag is set on (MOVEFILE executing from the console), the records in the file are not typed on the console.

The record in the reader buffer is scanned to compute its length with trailing blanks deleted. It is then moved to the CMS console read buffer and the computed length is stored in the original DMSCRD parameter list, whose address is passed by DMSCRD when it initially passes control to DMSBTP.

If the first user record is not a /JOB card, error message DMSBTP105E is typed and normal cleanup is performed with the BATTERM flag set on. This flag prevents another initial program load, since it is not needed at this time. Reads to the card reader are then issued until the next /JOB card is found.

If the first record is a /JOB card, DMSBTP branches to its /JOB card processing routine which calls DMSSCNN via a BALR. A check is made for the existence of the userid and account number on the card. If the fields exist, a CP diagnose '4C' is issued to start accounting recording for that userid and account number. If an error is returned from CP denoting an invalid userid, or if the userid or account number fields were missing on the /JOB card, error message DMSBTP106E is typed and normal cleanup is performed with the BATTERM flag set on.

The jobname, if provided on the /JOB card, is saved and a message is issued via SVC to inform the source userid that the job has started. The spooling devices are closed and respooled for continuous output, a CP QUERY FILES command is issued for information purposes and the implied CP function under CMS is disabled and the protection feature set off via SVC calls to SET (DMSSET). The BATPROF EXEC is executed via an SVC to EXEC. The BATNOEX flag, which is set by DMSBTB to suppress user job execution until the /JOB card is detected, is set off. The BATUSEX flag is set on (for DMSCPF) to signal the start of the actual user job, and a branch is taken to read the next card from the reader file (user job).

After reading the /JOB card, DMSBTP continues reading and checks for a /* card, a /SET card, or a CP command. If a card is none of these, DMSBTP passes control back to the command processor DMSINT for processing of the command (or data).

If a /* card is read and it is the first card of the new job, it is assume to be a precautionary measure and thus ignored by DMSBTP which then reads the next card. If it is not the first card a check is made for the BATMOVE flag. If the flag is on, the /* card indicates an end-of-file condition for the MOVEFILE operation from the console (reader) and is consequently translated to a null line for the MOVEFILE command.

If the BATMOVE flag is not on, the /* card is and end-of-job indicator and an immediate branch is taken to the end-of-job routine for cleanup and reloading of CMS batch.

When a CP command is encoutered DMSBTP branches to a routine that first checks a table of CP commands allowable in batch. If the command is allowed, a check is made for a reader or other spool device in the command line. If the CP command is allowed but would alter the status of the batch reader or any spooling device or certain disks, or if the command is not allowed at all, error message DMSBTP107E is typed, and the next card is read.

If the CP command is LINK, the device address is stored in a table so that DMSBTP can detach all user disk devices at the end of the job.

A CP DETACH command is examined for a device address corresponding to the system disk, the IPL disk, the batch 195 work disk or any spool device. If the device to be detached is any of these, error message DMSBTP107E is displayed and the next card is read. Otherwise, DMSBTP returns control to DMSINT (or DMSCPF is the BATCPEX flag is set on) for processing of the command.

When a /SET control card is encountered, the card is checked for valid keywords, valid integer values (less than or equal to the installation default values), and if an error is detected, error message DMSBTP108E is typed. An abnormal termination message is also sent to the source userid and the job is terminated with normal cleanup performed. If the control card values are valid, the appropriate fields are updated in the user job limit table DMSBTPLM and the next card is read.

If DMSBTP detects a "not ready" condition at the reader, a message is typed at the console stating that batch is waiting for reader input. DMSBTP then issues the WAITD macro to wait for a reader interrupt. When first detecting the empty reader, DMSBTP calls the CP accounting routines via a CP diagnose '4C' to charge the wait time to the batch userid.

If a hard error is detected at the reader, DMSBTP sends an "intervention required" message to the system console and branches to its abnormal terminal routine and waits for an interruption for the reader by issuing the WAITD macro.

When a /* card is read (with the BATMOVE flag off) or when the end-of-file condition occurs at the reader, DMSBTP branches to the cleanup routine which sends the source userid a message stating that the job ended normally or abnormally (if cleaning upa fter an abnormal termination) and turns off the BATUSEX flag (for DMSCPF) to signal the end of the user user job. CONWAIT (DMSCWT) is called via SVC to allow any console I/O to finish, the spooling devices are closed (including the console), and all disks that were made available by issuing the CP LINK command are returned by issuing the CP DETACH command.

DMSBTP then relinquishes control by issuing the CP IPL command with the PARM BATCH option which loads a new CMS nucleus and the next job is started when CMS attempts its first read to the console.

A branch is made to the CMSBTP routine when DMSBTP itself detects an I/O error at the reader. However, the primary purpose of the routine is to receive control not only from DMSABN when there is an abnormal termination during the user job, but also from DMSITE, DMSPIO, and DMSCIO when a user job exceeds one of the batch job limits (BATXLIM flag is on). This routine, entry point DMSBTPAB, calls the CP DUMP routine via SVC and then branches to the cleanup routine which reloads CMS Batch and treat the remainder of the current job as a new job with no /JOB card. This has the effect of flushing the remainder of the job. This technique is used because batch must keep its reader spooled "continuous." Entry point DMSBTPAB is also used by the CMS commands that are disabled in CMS batch. In this case (BATDCMS flag set on), an error message is displayed and control returned to CMS.

When a CP command is called via an SVC in DMSBTP, the CMS CP module (DMSCPF) is acutally called to issue the DIAGNOSE instruction to invoke the CP command. DMSBTP calls DMSCPF by issuing a direct SVC 202 or by issuing the LINEDIT macro with the CPCOMM option that generates an SVC 203.

OTHER CMS MODULES MODIFIED IN CMS BATCH

Several CMS modules check whether CMS batch is running, and, if so, perform functions associated with batch operation. These are shown in the following list:

Module	Function Performed for CMS Batch
DMSINI	Passes batch parameters to DMSINS.
DMSINS	Uses batch IPL parameters to reload CMS Batch.
DMSLDR	Loads DMSBTP into free storage.
DMSCRD	Passes control to DMSBTP to read from the reader rather than from the console.
DMSITE	Accounts for virtual time used by batch job -- ABEND if over limit.
DMSPIO	Accounts for number of lines printed by batch job -- ABEND if over limit.

DMSCIO	Accounts for number of cards punched by batch job -- ABEND if over limit.
DMSABN	Passes control to batch ABEND routine in DMSBTP.
DMSERR	Passes control to batch ABEND routine instead of entering disabled wait state.
DMSMVE	Turns the BATMOVE flag on and off -- allows batch to treat moved blanks as data.
DMSSET	Disabled if batch running, except during batch initialization.
DMSRDC	Disabled if batch running.
DMSCPF	Distinguishes between CP command issued by user and by batch.
DMSFLD	Disallows reader device specification.
DMSDSK	Disk load not allowed in batch.

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 or 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 7 or 8 characters; the first three characters are DMK. Labels are indicated by prefixing with a comma and the six-character module identification.

Note: annotated flow diagrams are not to be construed to be 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/370 CP INTERRUPTION PROCESSING

SVC INTERRUPTIONS - PROBLEM STATE

DMKPSASV
Entry for SVC interruptions from problem or supervisor states. For problem mode and ADSTOP (SVC X 'B3'), the overlaid instruction is replaced

DMKCFMBK
Console function mode is entered.

DMKPSASV
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.

DMKPSA, SVCVER
For invalid parameter passing, error recording is not performed.

DMKIOEVR
The SVC is reflected back to the user.

DMKIOFVR
On correct parameter reflection, record the error.

DMKTRCSV
The DMKTRC module is called if TRACE SVC was invoked.

DMKPSA, REFSVCB
For EC mode machine or page 0 not in real storage. Per results of TRACE activity, go to the DMKDSPCH; if not successful, go to DMKDSPB.

DMKPRGRF
If tracing not active, flag user as being in instruction wait state and reflect the SVC back to the user.

DMKPSASV
If the virtual machine is in BC mode and page 0 is in real storage, generate and store an old SVC PSW. Then fetch the new SVC PSW.

DMKDSPB
If wait state is not indicated, store user's new PSW in RUNPSW, restore registers and dispatch via LPSW.

SVC INTERRUPTIONS - SUPERVISOR STATE

DMKPSASU
Entry is for a system failure and is a SVC 0 or SVC 4 ABEND condition.

DMKDMPDK
Perform partial or full real storage dump.

DMKCKPT
Checkpoint the system.

DMKCPINT
Perform an automatic IPL if indicated

DMKPSA,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 callers addressability, SAVEAREA address and return address are maintained in a new SAVEAREA.

DMKPSA, SVCRET
Entry via SVC 12 return control from the called routine to the calling routine and restores addressability via R12 and R13.

DMKPGSUL
If a nonresident module - unlock page to return it to DASD device.

DMKPSA, SVCRLSE
Entry via SVC 16 to release the current SAVEAREA used by SVC 8 and 12. Return to caller.

DMKPSA, SVCGET
Entry via SVC 20 to obtain a new SAVEAREA. Return to caller.

EXTERNAL AND CLOCK INTERRUPTION REFLECTION

DMKPSAEX
Entered via the interruption key on system console, adjust accounting to charge for supervisor overhead. If problem mode, ATTN 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

DMKQCNCL
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 3704/3705, 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
Exit to dispatcher.

DMKPSA, EXTTIMER
Upon CPU timer interruption, VMTLEVEL in VMBLOK as a real CPU timer interruption.

DMKTMRVT
Simulate the interruption.

DMKDSPCH
Exit to the dispatcher.

DMKPSA, EXTCKC
Upon clock comparator interruption reflection

DMKSCHTQ
Use the printer to unchain the active TRQBLOK. Call DMKSTKIO.

DMKSTKIO
Stack the block.

DMKDSPCH
Exit to dispatcher.

MONITOR INTERRUPTION PROCESSING

DMKMONTI
For monitor requests, with an operation code of X'AF', increment TOD with DMKPRGTI value and insert in TRQBLOK.

DMKSCHST
Insert the block in the request block chain.

DMKMONTI
Collect Monitor timer driven date for the enabled classes. On the successful decode of the class and code, branch to the appropriate data collection routine.

Class	Code	Function
0	0	Perform timer driven system clock and counter recording
	97	MONITOR tape header record
	98	MONITOR tape trailer record
	99	MONITOR suspension due to tape busy
1	0	Begin console read
	1	Console output
	2	End Console read
	3	Console sleep
2	2	Drop user from queue
	3	Add user to queue
	4	Add user to eligible list
4	0	User statistics
5	0	Instruction simulation
6	0+1	DASTAP
	0	Device statistics header
7	0	DASD seek channel program
8	2	Device statistics and system counters

Each collection routine calls buffer management for space for the collected data. If the MONITOR tape is busy MONITOR activity is suspended. If not busy call -

DMKSTKCP
to stack a CPEXBLOK for the event to call the scheduler.

DMKMON, EXIT2
Restore all registers to their previous values prior to the MONITOR CALL. Load the old program check PSW to resume processing.

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 IPL

DMKPRGIN
For supervisor state and MONITOR CALL save registers in in DMKPRGPR

DMKPRGMI
Do MONITOR CALL interruption processing (DMKMON).

DMKPRG, PRNSTAT
For paging exception X'11' and EC mode with

DMKVATEX
Translation on, process the exception.

DMKPRGIM
For paging exception, x '11' and EC mode with DMKVATPF

DMKVATPF
Translation off, and enabled for I/O interrupts and PAGEX on, process the pseudo page fault

DMKPRG, PAGEXCP
For all other page fault conditons go to DMKPTRAN

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

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, PRG01
For privileged or operational exception of a virtual machine in supervisor mode, examine ITRPR+1 if X'01' or '02' call -

DMKPRVLG
to 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 virutal machine is in EC mode, paging is performed

DMKDSPB
Check the new PSW.

DMKPRVLG
Validate the privileged operation indicated in VMINST+2 and perform the service.

Code	Operation
X'08'	SSK - Set storage key
X'09'	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 - Text I/O
X'9E'	HIO - Halt I/O
X'9F'	TCH - Text Channel
X'AC'	STNSM - Store, then AND system mask
X'AD'	STOSM - Store, then OR system mask
X'B1'	LRA - Load real
X'B202'	STIDP - Store CPU 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'	IPD - Insert PSW key
X'B20D'	PTLB - Purge TLB
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

DMKHVCAL
On privileged operations of DIAGNOSE X'83' and the associated function code, perform the service.

DMKVIO
Execute privileged I/O operations of SIO, HIO, TIO and TCH.

DMKTMRTN
Perform privileged operations related to TOD clock, TOD clock comparator and the CPU timer.

DMKPRGSM
Program interruption is reflected back to the user on invalid instruction operands, unsupported instruction operand codes and DIAGNOSE '83' function codes that are not a multiple of 4.

VIRTUAL I/O OPERATIONS AND INTERRUPTION PROCESSES

CTCA OPERATIONS BETWEEN TWO VIRTUAL MACHINES

DMKVIOEX
Virtual I/O operation is reflected to DMKVCA, the channel adapter module, for processing.

DMKVCAST
For SIO, check if the CTCA is coupled. If not coupled, call DMKDIASM.

DMKDIASM
Simulate return status.

DMKVCA, VCRSTART
For a coupled CTCA, 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.

DMKVCATS
CTCA TIO activity is determined by examining Y-side information to determine mode and activity.

DMKVCASH
CTCA HIO and HDV is processed by determining the conition code to present and whether the Y-side should be notified.

DMKVCARD
CTCA process results from RESET xxx or SYSTEM RESET commands. The CTCA status is reset but the CTCAs are not uncoupled.

DMKVCARS
Uncoupling CTCA is achieved in the VDEVBLOK (VDEVNRDY flag) idle CTCA 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 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.
DMKGDDK
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, DIAGRTN
On interruption return, check status.
DMKUNTFR
If no problem encountered, free storage used for CCW string and IOBLOK.
DMKGIO, DIAGRTN
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

DMKIOEX
Entry from DMKPRV to simulate I/O per VMBLOK's VMIST field.
DMKVIO, VIOSIO On detected SIO, call -
DMKSCNVU
To locate VCHBLOK, VCUBLOK, and VDEVBLOK for the cuu called per SIO instruction.
DMKVIOEX
Determine device availability and set condition code accordingly.
DMKIOSQV
If the operation is warranted, schedule the operation.
DMKVIO, VIOTIO
For TIO, check device status, pending interrupts, and set appropriate condition codes.
DMKVIO, VIOHIO
For HIO, check for dedicated channel, CE, CU, or device busy condition, and subchannel busy and set appropriate condition codes.
DMKVIO, 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.
DMKUNTFR
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

DMKVIOEX
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.
DMKQCNRD
Execute the read operation and call DMKVCHEX.
DMKVCNEX
Set return address in VCONCTL VCNRDRET field.
DMKVSPVP
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
Write data to user's terminal.
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 VDEVBLOKS 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

DMKGRBEN
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 3066 console activity.
DMKFREE
Performs enabling function. Gets storage for IOBLOK and TRQBLOK generation.
DMKGRB, LOGUSER
Form and write out the logo at the screen.
DMKGRB, ATTNINI
Unsolicited attention for RDEVBLOK (enabled).
DMKBLDVM
Build LOGON VMBLOK for logon process.
DMKCFMBK
Enter console function mode for terminal input.
DMKIOSQR
Schedule request to clear screen preparatory to logon.
DMKDSPCH
Exit to dispatcher to wait for interruption. Successful logon per the next interruption begins the operation of building the user's virtual machine.
DMKGRAIN
Local 3270 display and 3066 interruption entry from dispatcher.
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.
DMKFREE
Get storage for function and build CONTASK, IOBLOK, TRQBLOK.

DMKIOSQR
Issue the SIO.
DMKDSPCH
Wait for the response.
DMKGRA, RDINT
On the interruption response, go to the return processing address in the TRQBLOK extension TRQBCRT. For read return, determine function key action and write response (if appropriate) via KEYTBL. On response of CE and DE go to auxiliary processing address and execute the processing routines:

CONRETBF	- completion of a write CONTASK
RDMINT	- completion of a buffer read
GRFCFM	- execute console function
SETREJ	- set no accepted timer
SETMOR	- set more... timer delay
SETWNG	- set 10 second clear warning
RDEXIT	- clear buffers after PF keys
STRTREAD	- set read status
NOCTL	- process next CONTASK or go idle

DMKGRA, 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.
DMKGRBIC
Entry point to process CONTASKS queue for local 3270 and 3066 devices.
DMKFREE
Get storage for IOBLOK and TRQBLOK.
DMKGRB, BLDCCWS
Execute CONTASK, if appropriate. If not -
DMKDSPCH
Exit to dispatcher.

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

Read Address	(2) TIC Address
Unused	
Read Data Area	
	(3) Modified CCW
(4) TIC to RCWTASK	
Real Read CCW	
Real TIC CCW	

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 VDEVBLOK or the VDEVBLOK is busy queue the IOBLOK and return to caller.

DMKIOS,IOSTATDV
If available status, start the I/O and return to caller.

SIO Operations

DMKIOS, IOBSTART
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, IOS10KI
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.

Interruption Processing

DMKIOSIN
Entry from I/O new PSW. Check old PSW. If problem mode, save CPU status in the VMBLOK.

DMKSCNRN
Locate RCHBLOK, RCUBLOK, RDEVBLOKs for interruption unit.

DMKVIODC
Process dedicated channel interruption condition. If control unit end or channel available interruption occurs, restart the operation, if interruption does not occur stack it.

DMKIOSIN
If the IOBLOK is not active on RDEVBLOK interruption, call DMKIOS.

DMKIOS, IOSENSE
Schedule sense operation, then go to dispatcher.

DMKIOS, IOSRSTRT
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, DOSENSE
For split seek complete interrupt, rechain the seek and reschedule operations.

DMKSTKIO
Stack IOBLOK and restart any units freed by the interruptions.

DMKDSPCH
Exit to dispatcher.

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 DMKQCO 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, CNBREAK
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, CNSLOGF
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.

DMKQCNET
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, CNSCTPR
Write 'VM/370 Online' interpretation of response determines retry, or build new CONTASK and execute or stack or process next CONTASK.

DMKQCNET
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

DMKQCNRD
SVC entry to build CONTASK for input data. Set the input buffer to zeroes.

DMKFREE
Get storage to build CONTASK.

DMKQCN, ENQUEUE
Stack CONTASK on RDEVBLOK, if RDEVCON was zero. If not, exit to the appropriate interrupt handler per RDEVTYPC and RDEVTYPE or -

DMKSPCH
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, WAKEUPR
On first CONTASK or priority CONTASK, enqueue on chain from RDEVBLOK in appropriate location, then call related interrupt handler.

DMKQCN, WAKEMUP
If NORET or DEFRET specified, build and stack CPEXBLOK to alert the interruption handler and return via EXIT SVC otherwise go to specified interruption handler.

DMKQCNTO
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.

DMKQCN, DSCGTRQ
Build TRQBLOK, if needed, for 15 minute delay, schedule it, and exit via SVC.

DMKQCN, 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 avaiability.

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, TASKENQ
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, RNICHN1
Return via R7.

DMKRNHND
Entry via SVC to schedule resource control tasks.

DMKRNH, RNHNDTK
Build control CONTASK and enqueue it for execution.

DMKRNH, STKCPEX
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.

DMKFRE
Get storage to build CKPBLOK (telecommunications control block), if necessary.

DMKRNH, RNSBITS
Record active line and enabled terminal flag bits.

DMKQCNET
Clear CONTASK chains.

DMKQCNTO
Force disconnect to all active users.

DMKNLDMP
DUMP the 3704/3705.

DMKNLDR
Reload the named program.

DMKRNHND
On 'IPL complete' signal, reenable 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, COMDISC
Generate response to a disconnect task error.

DMKRNH, COMCNTL
Generate response to a control task error.

DMKRNH, UNSOLIT
Generate response to a unsolicited read.

DMKQCNET
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, RNSLOWDN
If unit exception, set RDEVSLOW and reschedule rejected CONTASKs.

DMKQCNET
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

3277 Remote 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, 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.

DMKIOS, 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.

Secondary Interruption Processor for 3270

DMKRGAIN
Entry from DMKIOS, examine line interruption condition. Discard any of the following and go to the dispatcher: nonbinary 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.

TP Code	Function
TP00	Error Handling CCW
TP01	Enable/disable function
TP02	Write EOT (sequence prior to polling and addressing)
TP03	Write polling or addressing characters
TP04	Handle station's status and sense message
TP05	Read response to addressing
TP06	Write response to text
TP07	NO-OP following POLL command
TP08	Unit exception condition (timeout)
TP09	All reset commands
TP10	Read/write text
TP11	Read response to text

DMKDSPCH
Exit to the dispatcher.

3270 Binary Synchronous Line Error Recovery

DMKBSCER
Entry via DMKIOS and SVC 8 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
Free IOERBLCK.

DMKIOSQR
Go back to scheduler.

DMKRGF, UNITCK
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, place its CORTABLE entry into the user's page list then 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.

DMKPTRFT
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 ountby 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

DMKPGAGT
Entered via SVC call to read in 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.

DMKPRA, EXIT
Put real storage address of the virtual page is passed back to the caller in R2.

DMKRAPT
Entered via SVC call to write out a page to DASD storage.

DMKPTRAN
Locate the page to be moved and lock it.

DMKPRAPT
Store all registers in CPEXBLOK and flag CPEXRO as a write request.

DMKPAGIO
Write the page.

DMKPRA, 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 existance.

DMKVATBC
Free shadow page, segment and copy segment, when user leaves EC mode or alters CR 0.

DMKVATRN
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 faults 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,

DMKDSPGH
Exit to dispatcher.

ALLOCATION AND DEALLOCATION OF DASD SPACE

DMKPGTPG
Entry to search and allocate a DASD page for paging/spooling.

DMKPGTSG
Search appropriate RECBLOK chain for available DASD page. If none found, locate next available cylinder and construct a new RECBLOK, calculate address of the allocated DASD page and place it in R1. Return to caller.

DMKPGTRPR
Entry to deallocate DASD page used for paging and Spooling. Via RDEVBLOK 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.

DMKPGTSR
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 DASD spool space to record a 3704/3705 dump. Scan RDEVBLOK and associated ALOCBLOK for enough contiguous available space to record the dump. When found, flag cylinder as allocated and build and chain the required RECBLOKs.

DMKPGTVG
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.

DMKPGTVG
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

DMKVMAPS
Entry via DMKPRT because a shared page (address in R2) has been detected by CP. The virtual machine (VMBLOK) that caused the page alteration has its named system released. The original page swap tables are copies.

DMKRMSG
The running virtual machine is informed of the share page violation.

DMKVMASH
Entered via DMPDSP/BALR, the shared page table are examined for hardware change bit being on. The resulting condition code is reflected to the caller.

TEMPORARY DISK STORAGE MANAGEMENT

DMKTDKGT
Entry to allocate temporary disk space (T-disk). With RO 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. Using preformatted IOBLOK from IOBSTACK, fill in the CCWs with DASD opcode and values derived from CPEXBLOK swap table and core table. Chain 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.

DMKPAG, UNTRANS
Upon interrupt return, unchain the CPEXBLOK from the intransit queue.

DMKSTPCP
Stack all deferred requests for execution.

DMKPAG, UNSTACK3
Return IOBLOK to IOBSTACK or free it.

DMKPAG, OVERHEAD
Calculate paging load and store it, the TOD, and other values in PSA.

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. Some registers and flag full entry condition. Examine VMSHRSYS for shared segments. If so, decrement use count. On 0 use count unchain the SHRTABLE from the active list.

DMKPGS, CKCLEAR
On NOCEAR exit to caller. If not, store number of release pages in R8.

DMKPGS, PAGOUT2
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.

DMKDSPCH
Exit to caller.

DMKPGSFS
Entry to examine user's page tables for a named system. Locate segment table and check each page table header for a named system. If found, set cc=0; if not, set cc=2 and return to caller.

DMKPGSPS
Entry to release storage containing a named system passed by the caller. 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.

FREE STORAGE MANAGEMENT

DMKFREE
Entry to obtain a block of storage, validate input doubleword request (R0).

DMKFRE, 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 subpool size not found condition get next large subpool size. Remove block from chain, put address in R1 and return to caller.

DMKFRE TRYSPLIT
For subpool that cannot honor request, start search a 30 doubleword end 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 entry DMKFRE, FREESUB.

DMKFRERS
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. Return to caller.

DMKFRET
Entry to restore block to subpool or free storage. Per R0 and R1 (number of doublewords to be released and and address of the first double word, respectively), the subpool sized block is returned to the appropriate subpool. Update the pointer in the SUBTABLE.

DMKFRE, FRET21
If subpool size block being returned is within the dynamic paging area, process as a block of more than 30 doublewords.

DMKFRE, FRET20
Blocks larger than 30 doublewords to be returned are merged into the free storage chain indicated by DMKFRELs.

DMKPTRFT
Restore page to dynamic page area; if a complete page is alloted, blocks belonging to the dynamic paging area can be built.

DMKFRE, FRET03
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.

CP INITIALIZATION AND TERMINATION PROCEDURES

Loading the Nucleus

DMKCKPT
Initial entry point to load the system after loading the first module, DMKCKP, from the system residence volume. Check CPID in PSA for startup method.

DMKSAVRS
For CPID equal to not warm or not CPCP, insert COLD and load the nucleus. Then branch to DMKCPINT, to perform CP initialization.

DMKCKP, NOTERM
ON CPID=WARM or CPCP, halt and drain all I/O devices and remember enabled terminals.

DMKCKP, NEXTCH
DMKRSPCV to validate warm start cylinder.

DMKCKP, CLOCKOK
Save accounting data, log message, SDFBLOKs and enabled terminals and lines on checkpoint cylinders.

DMKCKP, CHK05
Save spool records allocation and spool hold queue blocks on checkpoint cylinder.

DMKCKP, SHUTSYS
If normal shutdown indicated, issue message to system operator and load disabled wait state code X'008'.

System Initialization

DMKCPINI
Entry point to perform system initialization.

DMKCPI, KEYLOOP
Determine real storage size, initialize CORTABLE, Allocate free storage and initialize system paging tables

DMKCPI, CPIHIP
Check via HIO for online and ready status of all DMKRIO generated devices.

DMKCPI, CPISTCAW
Read volume labels and match to RDEVBLOK, RDEVSER.

DMKCIP, DMPALLOC
Allocate dump file to system device.

DMKCPI, ALOCLP
Build allocation block for CP-owned devices.

DMKCPI, MICTEST
Test for virtual machine assist feature availability If available, build MICBLOK and link to VMMICRO.

DMKCPI, NPSWS
Locate an available primary or alternate system console (PSA values).

DMKCPI, NOTCHNG
Build user directory page list per DMKSYSUD.

DMKLOGOP
Log on the system operator.

DMKCPI, STARTSYS
Force non nucleus modules to DASD page device.

DMKIOEFL
Initialize error recording cylinders.

DMKNLDR
Auto load 3704/3705; if appropriate.

DMKCPVAE
Enable 270X lines, if appropriate.

DMKPTRUL
Unlock CPI as initialization is complete.

DMKDSPCH
Await interrupts.

Warm Start

DMKWRMST
Entry from DMKCPI initialization. Check R2=01, if so go to DMKWRN, WARMCLR for cold start. Check Warm start cylinder for 8 byte XFFs identifier.

DMKWRM, ENABLERT
If enable records on, warm start cylinder, enable appropriate RDEVBLOKs.

DMKWRM, EN370S
If warm start record indicates, set flag for auto load of the named NCP program.

DMKWRM, ENR3270
Enable binary synchronous lines by clearing NICBLOK Offline flag, (if appropriate)

DMKWRM, ACNTRT
Build ACNTBLOK, load it with warm start cylinder data and chain it.

DMKWRM, WARMLOG
Build buffer and load it with the saved log message.

DMKWRM, WARMSPL
Build SPFBLOKs and fill with appropriate printer, punch and reader spool data.

DMKWRM, WARHOLD
Build SHQBLOK and move hold queue record data to the new block and chain it to the hold queue chain.

DMKWRM, WARMCLR
Clear 8 bytes of record 1 on the warm start cylinder. Check CPID again.

DMKCKSWM
For CPID=CKPT or FORCE, reconstruct spool checkpoint records.

DMKCKSIN
For CPID=NOT CKPT or NOTFORCE, initialize the checkpoint cylinders.

DMKCKSPL
Files in the systems spool hold queue are added to the checkpoint cylinder.

DMKWRM, GETDISK
Read in the remainder of warm start data.

Normal Shutdown

DMKCPSSH
Entry point results from involing CP SHUTDOWN

command. Close active spool files for callers or operator console.

DMKCPS, DASDCH
Via RDEVBLOK, locate and record DASD statistical data.

DMKCPS, DASDCHI
Put CPCP into CPID to denote shutdown.

DMKDMPRS
Set up CAW, CCWs and issue IPL to system residence device to reinitialize CP.

DMKCKPT
Save spooling and accounting data.

DMKMONSH
Stop monitor tape activity.

DMKCPI SHUTSYS
Sense shutdown flag, issue DMKCPI961W, enter disable wait state code X'006'.

Dump the System

DMKDMPDK
Entry occurs via ABEND000 condition or by pressing system console RESTART button. Save PSA values. Determine if dump is full or just CP portion.

DMKDMP, DMPMSG
Format and issue ABEND message to operator and transfer to DMKDMP and DMPDASD.

DMKDMP, DMPDASD
Write out a defined amount of storage or all storage to selected DASD device.

DMKDMP, DSKEND
Place sending record number and the system file number in the dump file SFBLOK.

DMKDMP, RECSRCH
Chain dump file RECBLOKs to RDEVBLOK, and link dump file SFBLOK onto the system reader chain.

DMKDSP RESTART
Restart the system on warm start indication.

DMKDMP, DMPTAPE
Dump CP storage or all storage to the selected Tape Drive per specified tape parameters.

DMKDMD RESTART
Restart the system, if warm start is indicated.

DMKDMP, DMPPRT
Dump CP storage or all storage to the selected printer.

DMKDMS RESTART
Restart the system, if warm start is indicated.

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.

DMKFMBK
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.

DMKVMIPL
User's page 0, set console address, IPL device address, VMBLOK flags IPL device type and class and user CAW. 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 program is loaded.

DMKVMI, IPLDONE
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 loading the current PSW at virtual location 0.

Virtual Machine Termination

DMKUSOLG
Entry is the result of user invoking LOGOFF. Set flags in VMBLOK indicating logout operation.

DMKUSO, USO006
Retain line communication, if HOLD operand specified.

DMKUSO, USO008
Adjust return address to not run the user.

DMKUSOFF
Set VMBLOK flags.

DMKTRCND
Called to reset tracing.

DMKPERT
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 userdispatch status.

DMKCFPRR, DMKGSPO
Reset the virtual machine.

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 VMDISCK 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.

DMKSCNFD
Matches the 8-byte command name against the table of matching command names, the truncations of command names, and the allowable abbreviations, starting at COMNBEGO.

The format of the table entry is:

Field	Number of Bytes
Command name	8
Class mask	2
Abbreviation count	2
Routine address	4

DMKCFM, CONFFIND
After a command match has been made, the privilege class of the command is matched with the user's privilege class, VMCLEVEL in the VMBLOK.

DMKCFM, CONFCALL
The last 4 bytes of a command contain the address of the routine that processes the command.

Figure 55 is a list of all CP commands and the associated processing modules.

Command	Entry Label
AUTOLOG	DMKLOGON
LOGIN	DMKLOGON
LOGON	DMKLOGON
DIAL	DMKDIAL
ATTACH	DMKVDBAT
ATTN	DMKCFMRQ
ADSTOP	DMKCPVAC
ACNT	DMKCPVAC
BEGIN	DMKCFMBE
BACKSPAC	DMKCSOBS
CHANGE	DMKCSUCH
CLOSE	DMKCSPCL
COUPLE	DMKDIACP
DISPLAY	DMKCDBDI
DCP	DMKCDBDC
DEFINE	DMKFENIN
DETACH	DMKVDBDE
DISCONN	DMKUSODS
DISABLE	DMKVPVDS
DMCP	DMKCDBDM
DRAIN	DMKCSODR
DUMP	DMKCDBDU
ECHO	DMKMSGEC
EXTERNAL	DMKCPBEX
ENABLE	DMKCPVEN
FLUSH	DMKCSOFL
FORCE	DMKUSOFL
FREE	DMKCSPFR
HALT	DMKCPVH
HOLD	DMKCSPHL
INDICATE	DMKTHIEN
IPL	DMKCFGIP
LINK	DMKLNKIN
LOADBUF	DMKCSOLD
LOADVFCB	DMKCSOVL
LOCATE	DMKCFDLO
LOCK	DMKCPVLK
LOGOFF	DMKUSOLG
LOGOUT	DMKUSOLG
MONITOR	DMKMCCCL
MESSAGE	DMKMSGMS
MSG	DMKMSGMS

Figure 55. CP Commands and Their Module Entry Points (Part 1 of 2)

Command	Entry Label
NETWORK	DMKNETWK
NOTREADY	DMKCPBNR
ORDER	DMKCSUOR
PURGE	DMKCSUPU
QUERY[1]	DMKCFMQU
READY	DMKCPBRY
REPEAT	DMKCSORP
REQUEST	DMKCFMRQ
RESET	DMKCPBRS
REWIND	DMKCPBRW
SYSTEM	DMKCPBSR
SAVESYS	DMKCFGSV
SET	DMKCFSET[2]
SHUTDOWN	DMKCPVSH
SLEEP	DMKCFMSL
SPACE	DMKCSOSP
SPOOL	DMKCSPSP
STORE	DMKCDSTO
START	DMKCSOST
STCP	DMKCDSCP
TAG	DMKCSTAG
TERMINAL	DMKCFTRM
TRACE	DMKTRACE
TRANSFER	DMKCSUTR
UNLOCK	DMKCPUVL
VARY	DMKCPVRY
WNG	DMKMSGWN
WARNING	DMKMSGWN
*	DMKCFM
CP	DMKCFM

[1]Major operand decode of QUERY is by a scan table at QRYLIST in DMKCFMQU. Depending on the operand match, DMKCQP, DMKCQG, or DMKCQR are called. The respective entry points are DMKCQPRV, DMKCQGEN, and DMKCQREY.
[2]Major operand decode (except for PFnn) is contained by the scan table starting label SETSTART in DMKCFSET.

Figure 55. CP Commands and Their Module Entry Points (Part 2 of 2)

DMKQCNRD
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

First Reflection for the Dispatched Virtual Machine

DMKDSPA
Entry for fast reflection activity. Perform user (PSA RUNUSER) accounting and determine validity of fast reflection by examination of DMKDSTAT values.

DMKDSP, RUNTIME
Do user accounting, then load the remaining time slice.

DMKDSP, SETQUANT+4
Build the PSW, then dispatch virtual machine with LPSW RUNPSW.

PSW Validation

DMKDSPB
Entry to dispatcher when the user's PSW has been external to DMKDSP.

DMKDSP, CKPSW
Verify the PSW change.

DMKDSP, UNSTACK
Unstack any pending interrupts for the user (if enabled).

DMKDSPCH
Go to the dispatcher.

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
For CPSTATUS=CPRUN, stop charging time to old virtual machine, start charging time to new virtual machine.

DMKDSP, WAITIME
For CPSTATUS=CPWAIT, if old virtual machine was not CP start charging CP with wait time.

DMKDSP, PROCWAIT
Via VNTLEVEL, allocate time to appropriate virtual machine time category.

DMKDSP, UNSTACK
For nonrunnable virtual machine, go to entry DMKDSP, DISPATCH.

DMKDSP, UNSTACK
For runnable user, check pending interruptions for the following:

- DMKDSP, CKPEND
  Per interruption (VMPERPND).
  Pseudo page faults (VMPGPND)
  External interruptions
- DMKDSP, UNSTIO
  I/O interruptions.

- DMKDSP, STORECSW
  I/O interruptions are reflected by swapping user PSWs and storing the unit address and status in low storage.

DMKDSP, NOTRACZA
Clear the pending bit in the VMBLOKs.

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, DSPMSG
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
Determine if virtual machine is allowed additional execution. If not, use DMKSCHDL entry.

Dispatching the New Virtual Machine

DMKDSP CKCPSTAK
Process a stacked IOBLOK or TRQBLOK as indicated via DMKDSPRQ. The new user IOBUSER/TRQBUSER is time stamped and a branch is made to IOBIRA/TRQBIRA.

DMKDSP, CKCPREQ
If system extending search CPEXBLOK for exit address of DMKPTRFD, DMKPTRFE, or DMKPTRFP. If none found load a wait state.

DMKDSP, CKPREQB
If not extending, unstack first CPEXBLOK. The new virtual machine is time stamped and branch taken to CPEXADD.

DMKDSP, CKUSER
Load last virtual machine with remaining time slice if applicable. Load the highest priority user in the dispatch queue, if available and applicable. If not enter the wait state to await an interruption.

Scheduling Users for Execution

DMKSCHDL
Entry to modify the user's status. If the user has the wait bit on in his running status (VMRSTAT), the user is not dispatchable or unqueue before the user's time slice has ended, the user has set favored execution option, or the user is not eligable for Q1.

DMKSCH, CKCPWAIT
Determine the running or not running of the real timer per VMBLOKs VMRSTAT, VMTLEVEL values.

DMKSCH, CKRSTAT
Process virtual machine, if currently not runnable.

DMKSCH, CKRUN
Process virtual machine, if currently runnable.

DMKSCH, CKWRITING
Add runnable virtual machine to active queue from eligible list search. Return to entry DMKDSP, CKCPSTAK.

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.

SPOOOLING VIRTUAL DEVICE TO REAL DEVICE

Processing Virtual Output Files

DMKVSPEX
Entry from DMKVIO to initiate SIO on a spooling device that is available (not busy and no interruptions pending).

DMKVSP, OPEN
Determine if output device needs to be opened.

DMKSPLOV
If yes, build message control blocks: SFBLOK and VSPCTLBLOK.

DMKPGTVG
Obtain a virtual buffer; the address is stored in VSPVAGE.

DMKPGTSG
Obtain a DASD page; the address is stored in VSPDPAGE.

DMKVSP, BUILDCTL
Assign a spoolid and the other user, record, and device values plus DMKCVTDT.

DMKCVTDT
Assigns the time stamp and date and stores it in SFBLOK.

DMKVSP, PRTCONT
Generate TAG record at the start of the spool data buffer.

DMKVSP, CCWOK
After CCW validity check, data and CCWs (if appropriate) are moved to the work buffer. Trailing blanks are truncated and when the buffer is full, it is written out to the DASD slot.

DMKVSPVP
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.

Closing Virtual Output Files

DMKVSPCO
Entry via CP CLOSE command. If device busy, defer close operation by building CPEXBLOK, stack it and exit to dispatcher.

DMKVSP, PRTEOF
On device not busy, write final buffer page to DASD storage.

DMKSPLCV
Queue closed virtual printer or punch spool file, queued to the read 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

DMKVSP, OPENRDR
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

DMKVSPCR
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
Locates 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 DMKSPLDL.

DMKSPLDL
Delete the file.

DMKVSP, DVICECLR
To clear the device, call DMKRPAGT.

DMKRPAGT
Releases the storage page.

DMKPGTVR
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 and interrupted for spooling unit record device. IOBRADD points to the RDEVBLOK RDEVTYPC input or output class.

DMKRSP, RSPLOUT
If RDEVSPOL indicates an available spool device (not active),

DMKFREE
Get storage for a work buffer and build a RSPLCTL block and link it to RDEVBLOK.

DMKRSP, PRNXTFIL
Search printer and punch SFBLOK chains for corresponding device and class. On a found condition, unchain the block, put its address in RSPSFBLK.

DMKSEPSP
If called, provides separators for output pages or cards.

DMKRSP, PROCESS1
Bring first spool data DASD page to the work buffer and convert CCW addresses to real device addresses.

DMKIOSQR
Start the spool device.

DMKRSP, PRNXTPAG
Repeat the process until done.

DMKRSP, REPEAT
Reprocess and reaccess the buffer, if multiple copies are specified.

DMKCKSPL
Checkpoint records the change to COPY count.

DMKSPLDL
Delete the file on completion (unless HOLD specified).

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

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.

DMKPGTVG
Get virtual buffer and place its address in RSPVPAGE.

DMKPGTSG
Get DASD buffer and place its address in SFBSTART and RSPDPAGE, linke together by pointers.

DMKIOSQR
Start the reader.

DMKDSPCH
Await the interruption.

DMKRSP, RDERGETID
Check that the first card in the buffer is the userid header. If so, proceed.

DMKRSP RDRCARDS
Preload the buffer with CCWs.

DMKIOSQR
Issue the SIO (SIO's of 42 cards per buffer load).

DMKRSP, 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.

DMKRSP, RDREXIT4
Release storage for virtual buffer, RSPLCTL and the SFBLOK.

DMKDSPCH
Exit to the dispatcher.

SPOOL FILE DELETION

DMKPLDL
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.

DMKSPLDL
Assume the delete routine is not running, build a CPEXBLOK to call DMKSPLDR.

DMKSPLDR
Sets the DELSW=X'80' (delete routine active).

DMKSTKCP
Stacks it and exits to caller.

DMKSPLDR
On unstacking the CPEXBLOK, if the SFBLOK is a system dump file, calls DMKDRDDD.

DMKDRDDD
Deallocates DASD buffers.

DMKSPL, NEXTSFB
For complete allocation chains of RECBLOKS, call DMKPGTSR

DMKPGTSR
deallocate DASD buffer and return to storage held by the dummy RECBLOKs.

DMKSPL, DELSTART
For incomplete allocation RECBLOK chains, deallocate by calling DMKPGTSD.

DMKPGTSD
Deallocates a page at a time via SFBSTART and the IOBLOK until the last page is reached.

DMKFRFT
Delete the SFBLOK, then go to DMKSPL and NEXTSFB.

DMKSPL, 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 store CPU 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 are stored in the DMKCCH module.

DMKIOG, CHANGEID
Set up pointers for machine check and channel check record area and extended logout areas.

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 records on each cylinder 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.

DMKIOG, CYLFULL
On a cylinder full condition, 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 CPU or storage error. Disable soft machine recording store logout area on the machine check and channel check recording cylinders. The system is enabled for hard machine checks with a pointer to the termination routine. DMKMCH, ENHARD for virtual user store status in VMBLOK. DMKMCH, MCHSYSIL for system damage timing facility or uncorrectable retry, multibit storage error post system operator message, flag system as terminated. Place wait state code, if first hard error, record it. If the fault occurred in problem state, terminate the active virtual machine.

DMKMCH, SOFTSTG
For corrected ECC or CPU 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, intermittient, 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 terminated 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.

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, MCHTERM3
On second hard machine check error, or machine check handler is not active or hardware recovery is not active process as in DMKMCH, MCHSYSIL.

DMKMCH, MCHWAIT
For TOD damage, load PSW, enter wait state.

DMKMCH, MCHRESTA
If the TOD is not damaged, the address of the TOD is saved for accounting purpose and-

DMKDMPRS
Dumps and initiates system restart.

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, it 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, CCHWAIT
The SEREP code (X'0F') is placed in the interruption code of the machine check new PSW. The I/O old PSW, CSW, and CAW are restored. Checkpoint is set up by moving 'CPCP' into 'CPID'. The TOD clock is saved and a wait state PSW is loaded to place the system in a disabled wait state.

DMKCCH, SCNEND
Unless termination is established, return to DMKIOS for recovery.

Recording the Errors of the Virtual User Via SVC 76

DMKERO
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, EXITCCO
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 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 0 in users 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-

DMKPGTVG
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 does, 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. CCPARMs supplies the starting address and size parameters for this write operation.

DMKSNC, SVNCPIM
Save 3704/3705 control program image on NCPVOL device. CCPARMS also provides the parameters for this similar operation.

DMKSNC, SAVEFINI
Store cc=0 on no errors and return to caller.

SPOOL FILE CHECKPOINT AND RECOVERY

Initialization

DMKCKSIN
Entry from CP initializer, DMKCPI to initialize the checkpoint cylinders. Per DMKSYSCH, get a virtual page for the checkpoint cylinder and set up the device code in the system residence device. In addition set up local data areas such as pages per cylinder and checkpoint cylinders.

DMKCKS, CKSIN1
Loop through each SFBLOK in the system and checkpoint it in a slot on the checkpoint cylinder. Then loop through each remaining slot and mark it empty.

DMKCKS, CKSINS
Place the map delimiter of the last non-empty slot in the map.

DMKPTRUL
Unlock the map page.

DMKCKS, CKSIN5
Return to caller.

Dynamic Checkpoint of Spool Files and Spool Devices

DMKCKSPL
Entry from any routine that adds, deletes, changes, the status of closed spool files. Lock the routine, or waits until it becomes unlocked. Bring the 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
Unlock the routine. Unlock the page map and exit to caller.

Recontruction of Checkpointed Closed Spool Files

DMKCKSWM
Entry via DMKCPI during VM/370 reinitialization process whenever the records for closed spool data need to be reconstructed. Get a virtual page for the map of the checkpoint cylinder and set up the device code of the system residence volume. In addition, set up local data areas.

DMKCKS, CKSWM2B
For slots having real device entries, set or reset the RDEVDISA and RDEVDRAN and move in the checkpointed device classes into RVDEVCLAS.

DMKCKS, CKSWM2G
For slots containing spool hold queue block, chain this to the SHQ chain.

DMKCKS, CKSWM3
Get storage for SFBLOK space and set flags depending on its last checkpoint activity.

DMKCKS, CKSWM4
If the file SFBLOK was active, chain it to the appropriate printer, reader, or punch chain.

DMKCKS, CKSWM5
Allocate the DASD buffers of the spool file by reading each buffer to determine the next one and then allocate this page.

DMKCKS, CKSWM6E
For the dump spool file, the buffers are allocated sequentially from the beginning to the end.

DMKCKS, CKSWM9
Set up the map delimiter for the end of non-empty slot; then set up a new spool file identity (spoolid) higher than existing numbers. Return to DMKWRM.

RSCS PROGRAM ORGANIZATION

In this section, Figures 56 through 61 show how the RSCS routines interact with each other functionally. Figure 56 shows all of the RSCS components at an overview level. Figure 57 through 61 show the parts of the individual components.

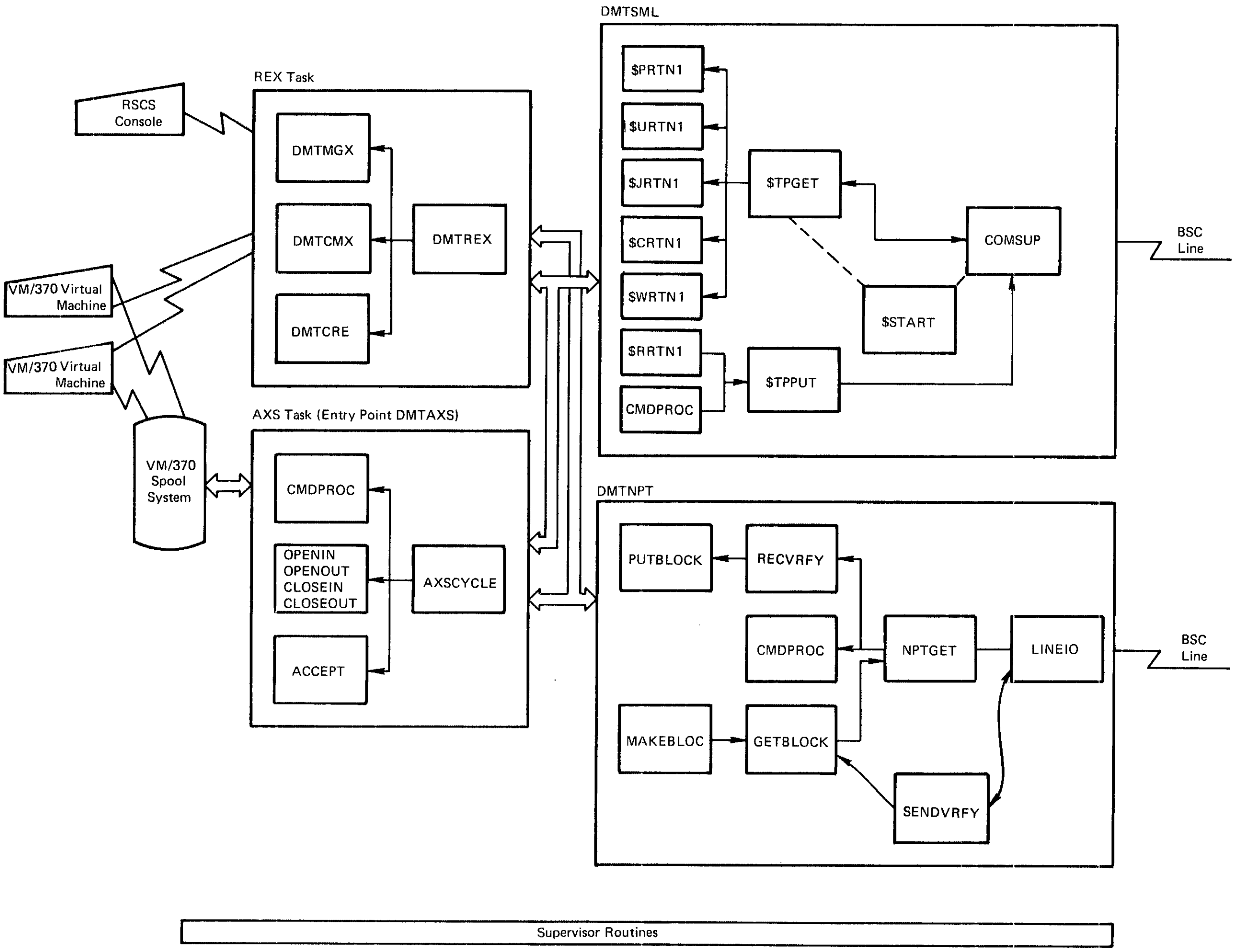


Figure 56. Overview of RSCS Program Organization

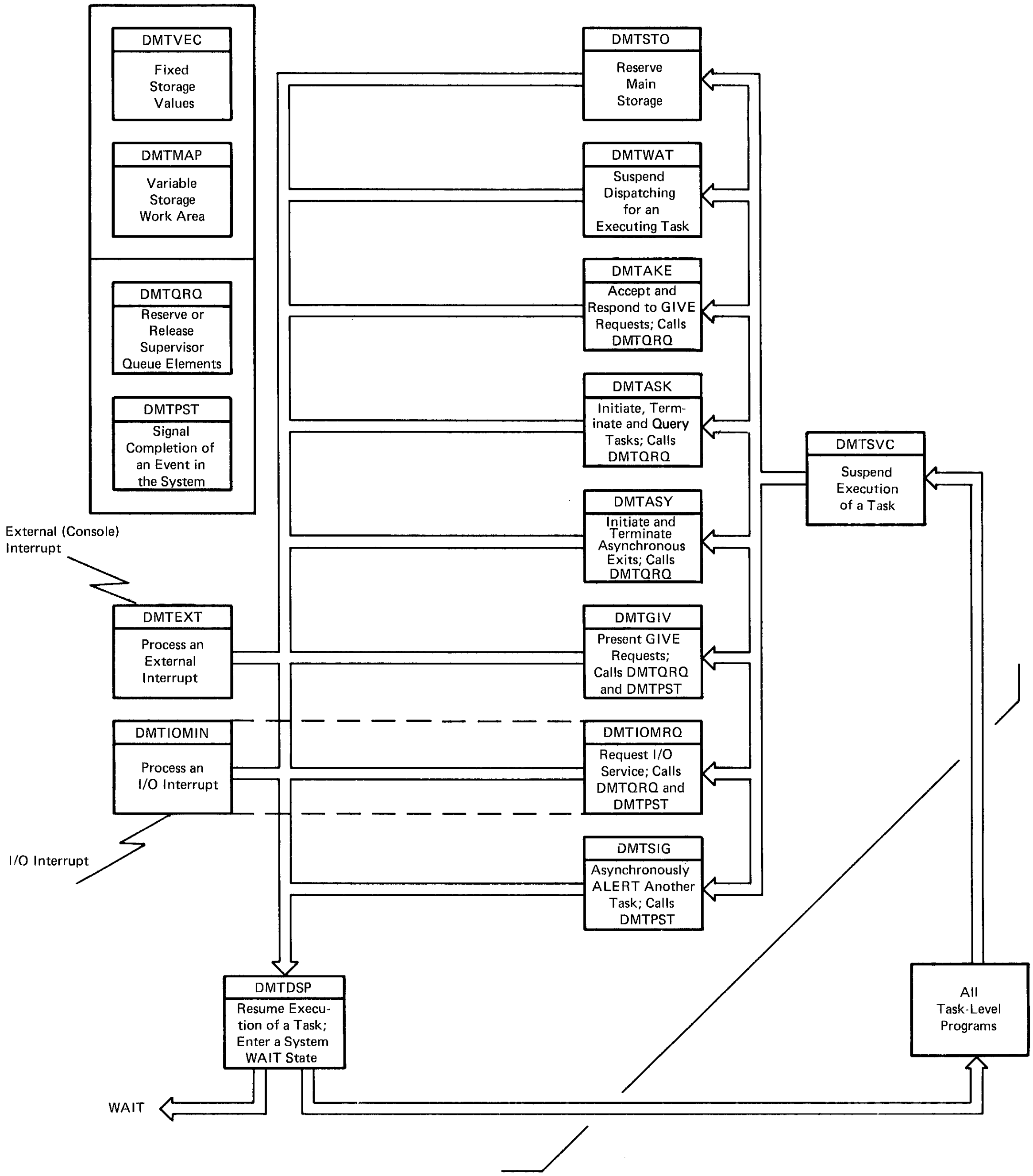


Figure 57. Program Organization for the Multitasking Supervisor

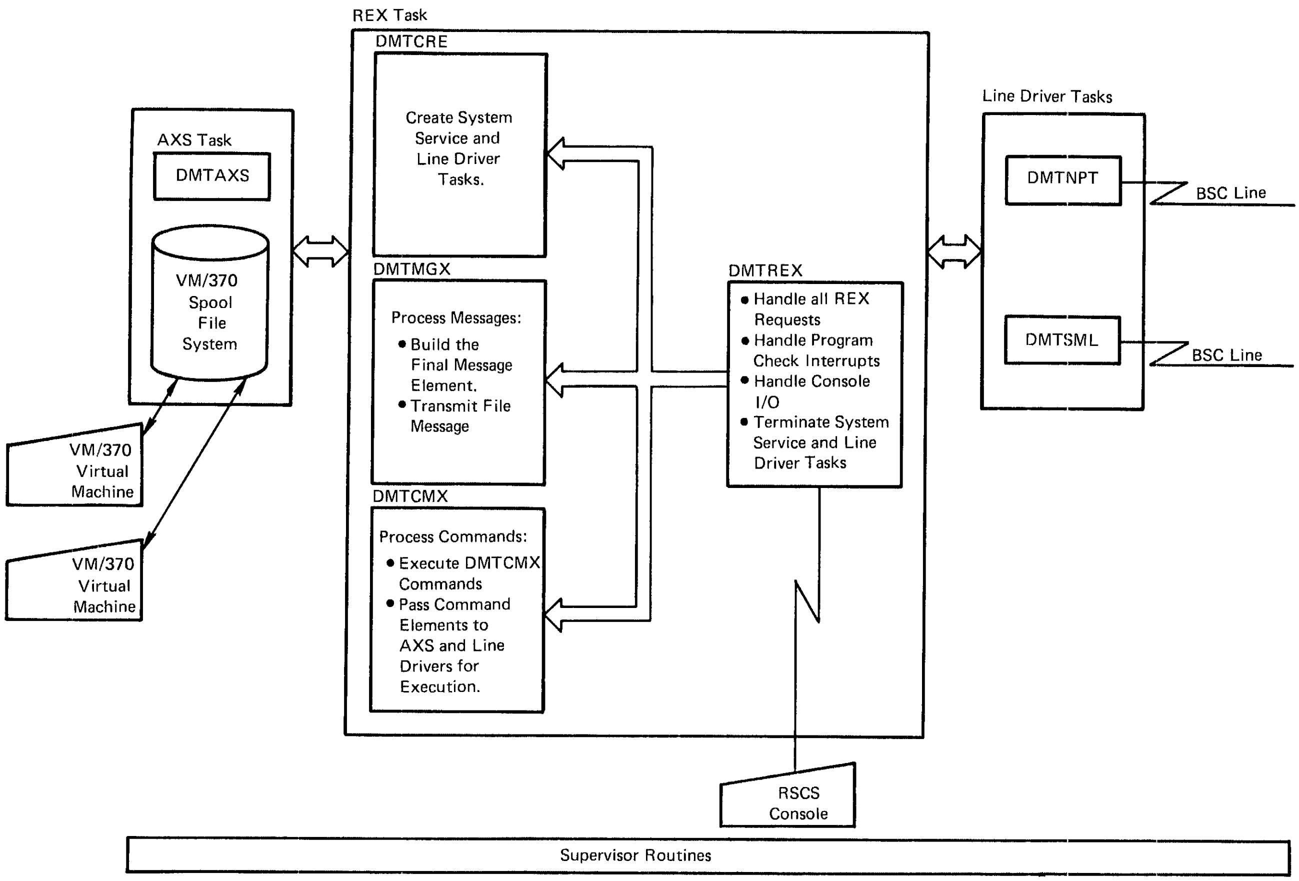


Figure 58. Program Organization for REX System Service Tasks

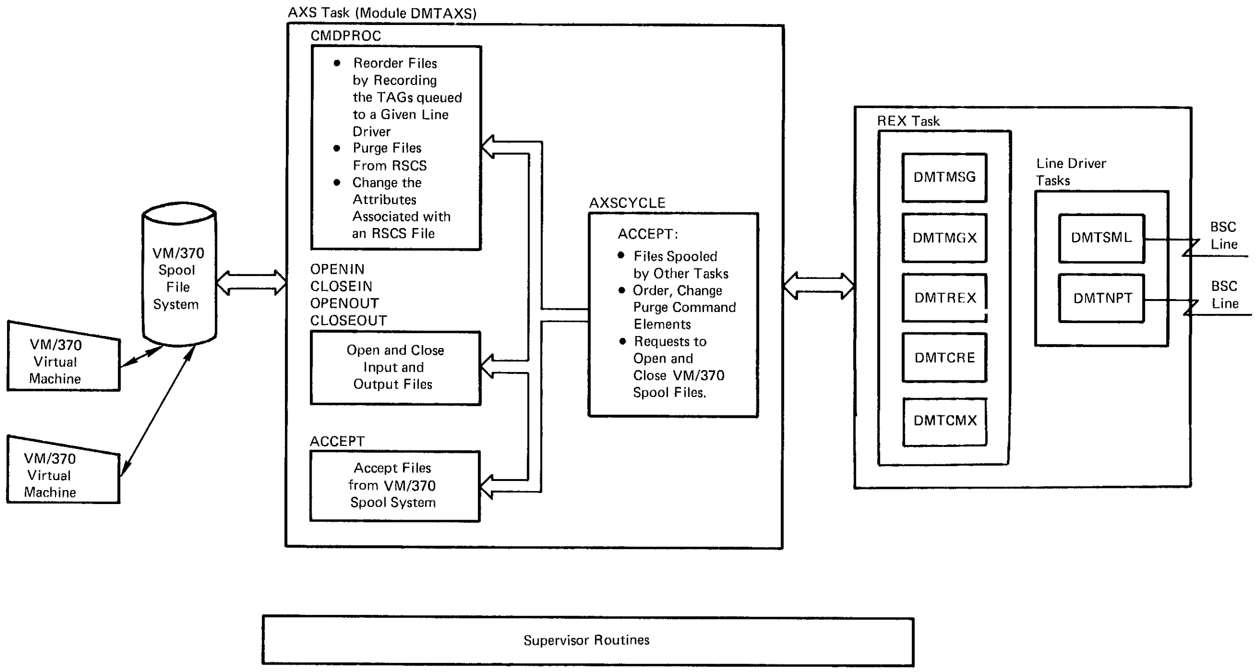


Figure 59. Program Organization for the AXS System Service Task

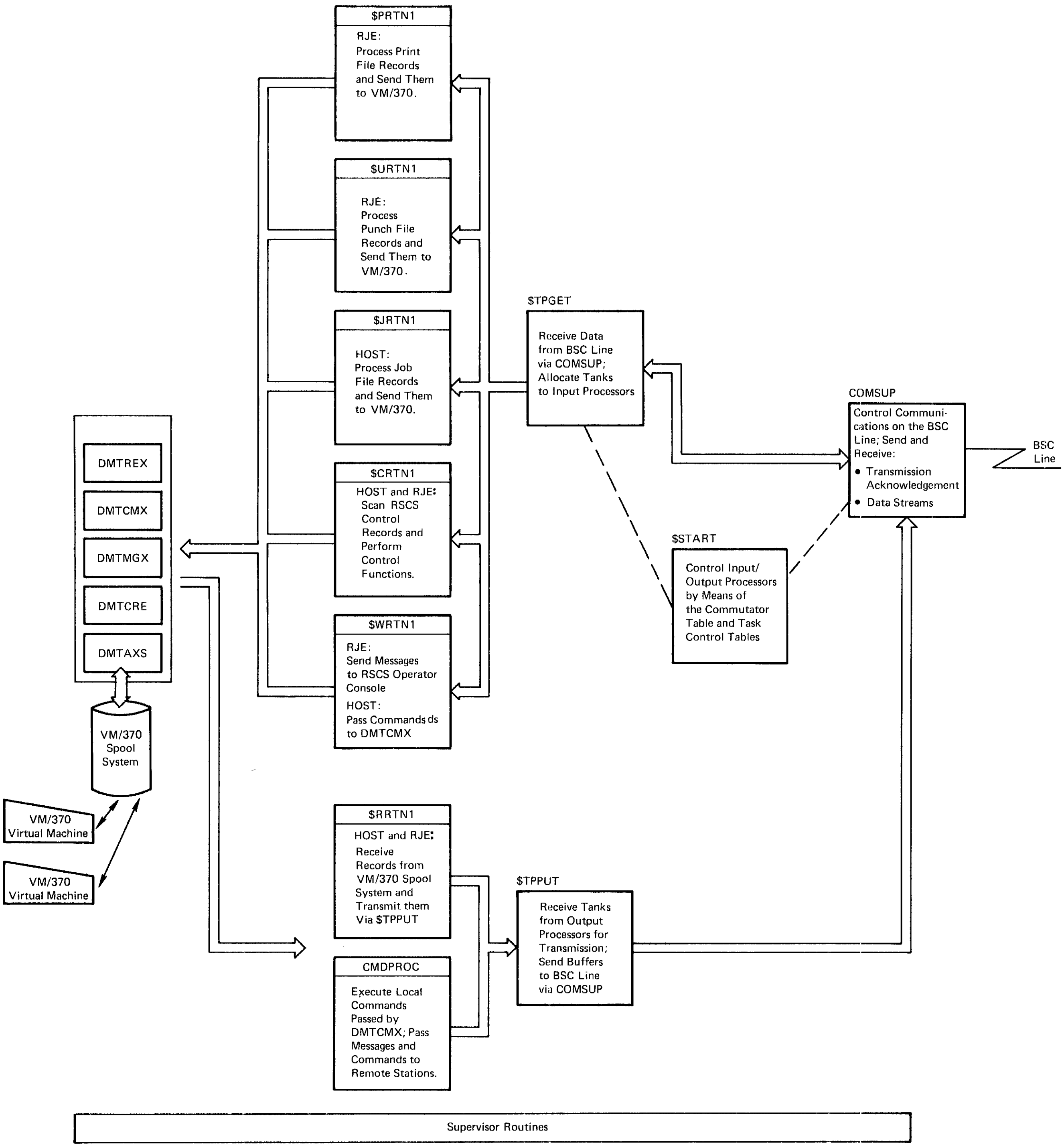


Figure 60. Program Organization for the SML Line Driver Task

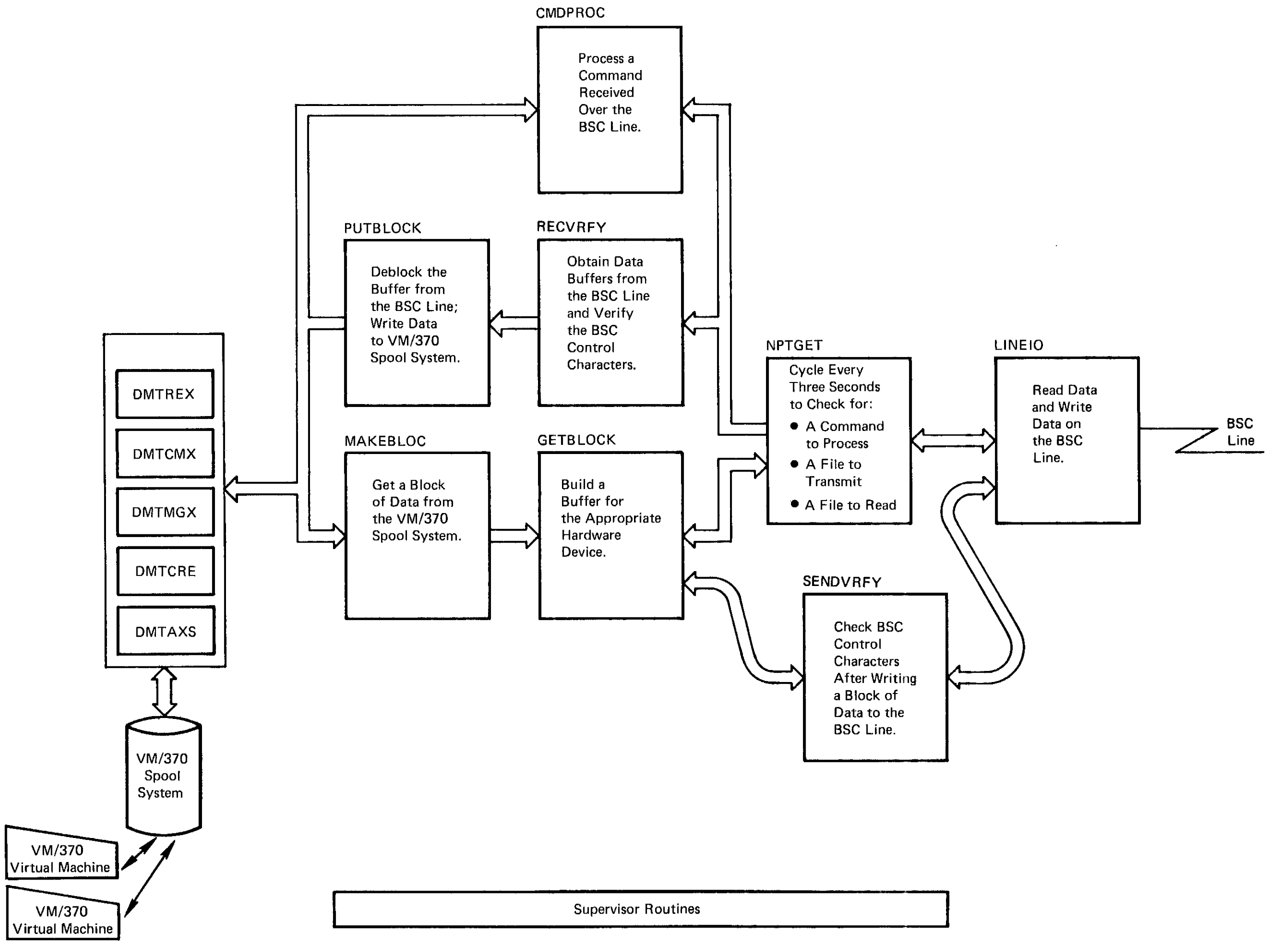


Figure 61. Program Organization for the NPT Line Driver Task

"Section 3. Directories" contains the cross-references for locating modules and labels within three VM/370 components. Section 3 also contains module descriptions for these components.

- CMS MODULE ENTRY POINT DIRECTORY

  Use this directory when you want to find the entry point and its function for any given module.

- CMS MODULE-TO-LABEL CROSS REFERENCE

  Use this directory when you want to know, for any given module, the names of any external references it may make to data areas, registers, or entry points in other modules.

- CMS LABEL-TO-MODULE CROSS REFERENCE

  Use this directory when you want to know which modules refer to any given label. This directory also, by means of the count field, indicates the number of times that the label was referenced.

- CP MODULE ENTRY POINT DIRECTORY

  Use this directory when you want to find the entry point and its function for any given module.

- CP MODULE-TO-LABEL CROSS REFERENCE

  Use this directory when you want to know, for any given module, the names of any external references it may make to data areas, registers, or entry points in other modules.

- CP LABEL-TO-MODULE CROSS REFERENCE

  Use this directory when you want to know which modules refer to any given label. This directory also, by means of the count field, indicates the number of times that the label was referenced.

- RSCS MODULE DIRECTORY

  Use this directory to determine what modules are branched to, from any given module, and the labels where the branchs occur.

- RSCS MODULE ENTRY POINT DIRECTORY

  Use this directory when you when to find the entry point and its function for any given module.

- RSCS MODULE-TO-LABEL CROSS REFERENCE

  Use this directory when you want to know, for any given module, the names of any external references it may make to data areas, registers, or entry points in other modules.

- RSCS LABEL-TO-MODULE CROSS REFERENCE

  Use this directory when you want to know which modules refer to any given label. This directory also, by means of the count field, indicates the number of times that the label was referenced.

Module Name	Entry Points	Function
DMSABN	DMSABN	Intercepts an abnormal termination (ABEND) and provides recovery from the ABEND. Entered by a DMKABN TYPCALL=BALR macro call.
	DMSABNKX	Entered by a KXCHK macro to halt execution after HX has been entered after signaling attention.
	DMSABNGO	Entered by any routine that sets up ABNPSW and ABNREGS in the work area beforehand.
	DMSABNSV	Entered as the result of a DMSABN TYPCALL=SVC macro call.
	DMSABNRT	Returns entry point from DEBUG.
DMSACC	ACCESS	Accesses data in the ADT and related information (such as AFT's and chain links) in virtual storage.
DMSACF	READFST	Reads all file status table blocks into storage for a read/write disk. Reads in file management tables for a read - only disk. For an O/S disk, control returns to to the caller after a successful return from DMSACM.
DMSACM	READMFD	Reads the ADT, QMSK, QQMSK, and first chain link into virtual storage from the master file directory on disk.
DMSALU	RELUFD	For a specified disk, releases all tables kept in free storage and clears appropriate information in the active disk table (ADT).
DMSAMS	DMSAMS	Provides an interface to DOS Access Method Utility programs (IDCAMS). Provided for support of CMS/VSAM.
DMSARD	DMSARD	Provides storage for the ASM3705 assembler auxiliary directory. DMSARD contains no executable code. It must be loaded with DMSARX and the GENDIRT command must then be issued to fill in the auxiliary directory entries. GENMOD must then be issued to create the ASSEMBLE module.

Module Name	Entry Points	Function
DMSARE	DMSARE	Releases storage used for tables pertaining to a given disk when that disk is no longer needed.
DMSARN	DMSARN	This is the ASM3705 command processor. It provides the interface between user and the 370x Assembler.
	ASMHAND	This is the SYSUT2 processing routine called from DMSSOB and used during the assembly whenever any I/O activity pertains to the SYSUT2 file.
DMSARX	DMSARX	Provide an interface for the ASM3705 command to the 3705 assembler program.
DMSASD	DMSASD	Provides storage for the assembler auxiliary directory. DMSASD contains no executable code. It must be loaded with DMSASM and the GENDIRT command must then be issued to fill in the auxiliary directory entries. The GENMOD command must then be issued to create the assemble module.
DMSASM	DMSASM	Processes the ASSEMBLE command. Provides the interface between the user and the system assembler.
	ASMPROC	This is the SYSUT1 processing routine (called from DMSSOB).
DMSASN	DMSASN	Associates logical units with a physical hardware device. (Interface for the ASSGN command used by CMS/DOS and CMS/VSAM.)
DMSAUD	DMSAUD	Reserves space on disk for writing a copy of disk and and file management tables on disk and then updates the master file directory.
	DMSAUDUP	Closes all CMS files, thereby updating the master file Directory for any disks that had an output file open.
DMSBAB	DMSBAB	Give control to an abnormal termination routine once linkage to such a routine has been established by STXIT AB macro.

Module Name	Entry Points	Function
DMSBOP	DMSBOP	Opens CMS/DOS files associated with the following DTF (Define The File) tables: DTFCN, DTFCD, DTFPR, DTFMT, DTFDI, DTFCP, DTFSD. Once the files are opened and initialized, I/O operations can be performed using the file.
DMSBRD	DMSBRD (RDBUF)	Reads one or more successive items from a specified file.
DMSBSC	BASIC	Processes the BASIC command. The BASIC command invokes the CALL-OS BASIC language processor to compile and execute the specified file of BASIC source code.
DMSBTB	DMSBTB	This is the CMS batch bootstrap routine. It loads the batch processor routine (DMSBTP) and user exit routine (if they exist) into free storage.
DMSBTP	DMSBTP	Main entry; reads from the virtual card reader each time CMS tries to execute a console read.
	DMSBTPAB	Entry point for abnormal conditions during user job: • Job exectuion ABEND (from DMSABN) • Job limit exceeded (from DMSITE, DMSCIO, DMSPIO) • Disabled CMS command (from the command)
	DMSBTPLM	Non-executable user job limit table referenced by DMSITE, DMSPIO, and DMSCIO.
DMSBWR	DMSBWR	Writes one or more successive items into a specified disk file.
DMSCAT	DMSCAT	Stacks a line of console input that DMSCRD reads later when it is called.
DMSCIO	DMSCIOR	Reads one card record.
	DMSCIOP	Punches one card record.
	DMSCIOSI	Punch caller's buffer.

Module Name	Entry Points	Function
DMSCIT	DMSCIT	Processes the interruptions for all CMS terminal I/O operations and starts the next I/O operation upon completion of the current I/O operation.
	DMSCITA	Processes terminal interruptions.
	DMSCITB	Starts next terminal I/O operation.
	DMSCITDB	Frees I/O buffers from stacks.
DMSCLS	DMSCLS	Closes CMS/DOS files associated with the following DTF (Define The File) tables: DMTCN, DTFCD, DTFPR, DTFMT, DTFDI, DTFCP, and DTFSD. For reader, printer, or punch files, a CP CLOSE command is issued. For disk files, DMSFNS is called to close the file. For a disk work file, DMSERS is called to erase the file, unless DELETFL=NO is specified.
DMSCMP	COMPARE	Compares the records contained in two disk files.
DMSCPF	DMSCPF	Passes a command line to CP for execution.
DMSCPY	DMSCPY	Processes the COPYFILE command to copy disk files.
DMSCRD	DMSCRD	Reads an input line and makes it available to the caller.
DMSCWR	DMSCWR	Writes an output line to the console.
DMSCWT	DMSCWT	Causes the calling program to wait until all terminal I/O operations have been completed.
DMSDBD	DMSDBD	Enables a user to dump his virtual storage from within an executing program.
DMSDBG	DMSDBG	Enables the user to debug his program from the terminal.
	DMSDBGP	Entry point for program interruptions.
	DMSDBG	Entry point for all other interruptions.
DMSDIO	DMSDIOR	Reads one or more 800-byte records (blocks) from disk, or reads one 200-byte record (sub-block) from disk.
	DMSDIOW	Writes one or more 800-byte records (blocks) on disk, or writes one 200-byte record (subblock) on disk.

Module Name	Entry Points	Function
DMSDLB	DMSDLB	Interface for the DOS DLBL command; allows the user to specify I/O devices extents, and certain file attributes for use by a program at execution time. DLBL can also be used to modify or delete previously defined disk file descriptions.
DMSDLK	DMSDLK	Interface for the DOS user command. Link-edit the relocatable output of the language processors. Once link-edited, these core image phases are added to the end of the specified DOSLIB.
DMKDMP	DMKDMP	Simulates the DOS/VS $$BDUMP and $$BPDCMP functions. For both functions, a CP DUMP command is issued, directing the dump to an offline printer.
DMSDOS	DMSDOS	Provides DOS SVC support. Interprets DOS SVC codes and passes control to appropriate routines for execution (for example, OPEN, CLOSE, FETCH, EXCP).
DMSDSK	DMSDSK	Dumps a disk file to cards or loads files from card to disk.
DMSDSL	DMSDSL	Provides capability to delete members (phases) of a DOSLIB library; also, to compress a DOSLIB library; also, to list the members (phases) of a DOSLIB library.
DMSDSV	DMSDSV	Lists the directories of DOS private or system packs.
DMSEDC	DMSEDC	Arranges compound (overstruck) characters into an ordered form and disregards tab characters as special characters.
DMSEDF	DMSEDF	Provides the Editor with the proper settings (CASE, TAB, FORMAT, SERIAL, etc.) by filetype. Contains non-executable code for reference by DMSEDI.
DMSEDI	DMSEDI	Modifies the contents of an existing file or creates a new file for editing.

Module Name	Entry Points	Function
DMSEDX	DMSEDX	Performs initialization for the CMS Editor.
DMSERR	DMSERR	Builds a message to be written at the virtual console by DMSCWR.
DMSERS	DMSERS	Deletes a file or related group of files from read/write disks.
DMSEXC	DMSEXC	Bootstrap loader for disk version of EXEC.
DMSEXT	DMSEXT	Processes the EXEC command.
DMSFCH	DMSFCH	Bring a specified phase into storage from a system or private core image library or from a CMS DOSLIB library. DMSFCH is invoked via SVC 1, 2, or 4 or via the FETCH command.
DMSFET	DMSFET	Provides an interface for the FETCH command; also, provides the capability to start execution of a specified phase.
DMSFLD	DMSFLD	Interprets OS JCL DD parameters for use by CMS.
DMSFNC	DMSFNC	Nucleus resident command name table.
	DMSFNCSV	Standard SVC table.
DMSFNS	DMSFNSA	Closes one or more input or output disk files.
	DMSFNSE	Closes a particular file without updating the directory or removing it from the active file table.
	DMSFNST	Temporarily closes all output files for a given disk.
DMSFOR	DMSFOR	Physically initializes a disk space for the CMS data management routines. For an existing disk, any information on the disk may be destroyed. The label may be changed and the number of cylinders allowed may be changed.

Module Name	Entry Points	Function
DMSFRE	DMSFREB	Called as a result of the DMSFREE and DMSFRET macro calls. Allocates or releases a block of storage depending upon the code in NUCON location CODE203.
	DMSFREES	Called as a result of the SVCFREE macro call. The size of the block is loaded from the PLIST and a DMSFREE macro is executed. Upon return, the address of the allocated block is stored into the PLIST.
	DMSFRETS	Called as a result of the SVCFRET macro call. The size and address of the block to be released are loaded from the PLIST and a DMSFRET macro is executed.
	DMSFREEX	Called as a result of a BALR to the address in the NUCON location AFREE. Executes the DMSFREE macro.
	DMSFRETX	Called as a result of a BALR to the address in the NUCON location AFRET. Executes the DMSFRET macro.
	DMSFRES	Called as a result of executing the DMSFRES macro. DMSFRES processes the following service routines: CKOFF, INIT1, INIT2, CHECKS, UREC, and CALOC.
DMSGIO	DMSGIO	Creates the DIAGNOSE and CCWs for an I/O operation to a display terminal from a virtual machine.
DMSGLB	DMSGLB	Defines the macro libraries to be searched during assembler processing. Defines text libraries to be searched by the loader for any unresolved external references.
DMSGND	DMSGND	Generates auxiliary system status table.
DMSGRN	DMSGRN	Edits STAGE1 output (STAGE2 input), builds 3705 assembler files, link-edits text files and an EXEC macro file.
DMSHDI	DMSHDI (HNDINT)	Sets the CMS interruption handling functions to transfer control to a given location for an I/O device other than those normally handled by CMS, or clears previously initialized I/O interruption handling.

Module Name	Entry Points	Function
DMSHDS	DMSHDS	Initializes the SVCINT SVC interruption handler to transfer control to a given location for a specific SVC number (other than 202) or to clear such previous handling.
DMSINA	DMSINA	Handles either user-defined synonyms or abbreviations or system-defined synonyms for command names.
DMSINDEX	DMSINDEX	Index of CMS listings in the microfiche deck.
DMSINI	DMSINIR DMSINIW	Reads a nucleus into main storage. Writes a nucleus onto a DASD device.
DMSINM	DMSINM (GETCLK) (CMSTIMER)	Obtains the time from the CP timer.
DMSINS	DMSINS	Controls initialization of the CMS nucleus.
DMSINS	DMSINS	Controls initialization of the CMS nucleus.
DMSINT	DMSINT DMSINTAB SUBSET	Reads CMS commands from the terminal and executes them. Entry is from DMSINS. Entry from DMSABN. CMS subset entry.
DMSIOW	DMSIOW, WAIT, DMSIOWR, WAITRTN	Places the virtual CPU in the wait state until the completion of an I/O operation on one or more devices.
DMSITE	DMSITE, EXTINT, DMSITET, TRAP,	Processes external interruptions.
DMSITI	DMSITI, IOINT,	This module is entered when an I/O operation causes the I/O new PSW to be loaded. This module handles all I/O interruptions, passes control to the interruption processing routine, and returns control to the interrupted program.

Module Name	Entry Points	Function
DMSITP	DMSITP	Processes program interruptions and processes SPIE exits.
DMSITS	DMSITS	Avoids CP overhead due to SVC call.
	DMSITS1	Address pointed to by the CMS SVC new PSW. This point is entered whenever an SVC interruption occurs.
	DMSITSCR	Return point to which a program called by a CMS SVC returns when it is finished processing.
	DMSITSOR	Return point to which a program called by an OS SVC returns when it is finished processing.
	DMSITSK	Called by an SVC by the DMSKEY macro.
	DMSITSXS	Called by an SVC from the DMSEXS macro.
	DMSITSR	This is the DMSITS recovery and reinitialization routine, called by DMSABN. DMSABN is the ABEND recovery routine.
DMSLAD	DMSLAD, ADTLKP	Finds the active disk table block whose mode matches the one supplied by the caller.
	DMSLADN, ADTNXT,	Finds the first or the next ADT block in the active disk table.
	DMSLADW	Finds the read or write disk according to input parameters.
	DMSLADAD	Modifies the file status table chain to include an auxiliary directory, or clears the auxiliary directory from the chain.
DMSLAF	DMSLAF, ACTLKP	Finds the active file table block whose filename, filetype, and filemode match the one supplied by the caller.
	DMSLAFNX, ACTNXT,	Finds the next or first AFT block in the active file table.
	DMSLAFFE ACTFREE	Finds an empty block in the active file table or adds a new block from free storage to the active file table, if necessary, and places a file status entry (if given) into the AFT block.
	DMSLAFFT ACTFRET	Removes an AFT block from the active file table and returns it to free storage if necessary.
DMSLBM	DMSLBM	Generates a macro library, adds macros to an existing library, and lists the dictionary of an existing macro library.

Module Name	Entry Points	Function
DMSLBT	DMSLBT, TXTLIB,	Creates a text library, adds text files to an existing text library, creates a disk file that lists the control section and entry point names in a text library or types, at the terminal, the control section and entry point names in a text library.
DMSLDR	DMSLDRA	Begins execution of a group of programs loaded into real storage. Definition of all undefined programs is established at location zero. Entered from the START command or internally from DMSLDRB LDT routine if START is specified.
	DMSLDRB	Processes TEXT files that may contain the following cards: SLC, ICS, ESD, TXT, REP, RLD, END, LDT, LIBRARY, and ENTRY. Entered from DMSLDP when the load function is requested.
	DMSLDRC	Does the processing required by various loader routines when an invalid card is detected in a text file.
	DMSLDRD	Does the processing required when a fatal I/O error is detected in a text file.
DMSLDS	DMSLDS	Lists information about specified data sets residing on an OS disk. Processes the LISTDS command.
DMSLFS	DMSLFS, TYPSRCH	Finds a specified 40-byte FST entry within the FST blocks for read-only or read/write disks.
DMSLGT	DMSLGTA	Entered from DMSLDRB if not a dynamic load. Frees all the TXTLIB blocks on the TXTLIB chain.
	DMSLGTB	Reads TXTLIB directories into a chain of free storage directory blocks. Entered from DMSLDRB.
DMSLIB	DMSLIB	Searches TEXT libraries for undefined symbols and closes the libraries.
DMSLIO	DMSLIO	Creates the load map on disk and types it at the terminal. Performs disk and typewriter output for DMSLDR.
DMSLKD	DMSLKD	Provides an interface between CMS and the VS1 linkage editor.
DMSLLU	DMSLLU	Lists the assignments of logical units.

Module Name	Entry Points	Function
DMSLOA	DMSLOA	Processes the LOAD and INCLUDE commands to invoke the relocating loader.
DMSLSB	DMSLSBA	Hexadecimal to binary conversion routine.
	DMSLSBB	Adds a symbcl to the string of locations waiting for an undefined symbol to be defined.
	DMSLBC	Removes the undefined bit from the REFTBL entry and replaces the ADCON with the relocated value.
	DMSLBD	Processes LDR options.
DMSLST	DMSLSTA	Processes the LISTFILE command. Prints information about the specified files.
DMSLSY	DMSLSY	Generates a unique character string of the form Z000001 for private code symbols.
DMSMDP	DMSMSP	Types the load map associated with the specified file on the terminal.
DMSMOD	DMSMOD	Processes the GENMOD command to create a file that is a core image copy; processes the LOADMOD command to load a file that is in core image form.
DMSMVE	DMSMVE	Transfers data between two specified OS ddnames, the ddnames may specify any devices or disk files supported by the CMS system.
DMSNCP	DMSNCP	Reads a 3705 control program module (Emulator Program or Network Control Program) in OS load module format and writes a page-format core image copy on a VM/370 system volume.
DMSNUC	DMSNUC	Contains CSECTS for nucleus work areas and permanent storage.
	NUCON	Nucleus constant area.
	SYSREF	Nucleus address table.
	DEVTAB	Device table.
	ADTSECT	Active disk table.
	AFTSECT	Active file table.
	EXTSECT	External interruption storage.
	IOSECT	I/O interruption storage.

Module Name	Entry Points	Function
DMSNUC (cont.)	PGMSECT	Program Interruption storage.
	SVCSECT	SVC interruption storage.
	DIOSECT	Disk I/O storage.
	FVS	File system storage.
	OPSECT	Parameter lists.
	CVTSECT	Simulated OS CVT.
	DBGSECT	Debug storage.
	TSOBLKS	TSO control blocks.
DMSOLD		Performs initialization and processing for each loading operation by processing text files that contain the following cards: SLC, ICS, ESD, TXT, REP, RLD, END, LDT, LIBRARY, and ENTRY.
	DMSOLD	Entered from DMSSLN when load requested.
	DMSIDRC	Entered when an invalid card is detected in a text file.
	DMSLDRD	Entered when a fatal error occurs during loading.
DMSOPL	DMSCPL	Reads the appropriate system directory records and headers and determines if the specified libraries contain any active members. Returns the disk address of the specified system library and indicates whether or not there are active members to be accessed on the disk.
DMSOPT	DMSOPT	Sets DOS options in the System Communications Region as specified by the OPTION command.
DMSOR1	DMSOR1	Relocates all DFT (Define The File) Table address constants to executable storage addresses. (Called by $$BOPENR via SVC 2.)
DMSOR2	DMSOR2	Relocates all DTF (Define The File) Table address constants to executable storage addresses. (Called by DMSOR1.)
DMSOR3	DMSOR3	Relocates all DTF (Define The File) Table address constants to executable storage addresses. (Called by DMSOR2.)

Module Name	Entry Points	Function
DMSOVR	DMSOVR	Analyzes the SVCTRACE command parameter list and loads the DMSOVS tracing routine.
DMSOVS	DMSOVS	Provides trace information requested by the SVCTRACE command.
DMSPIO	DMSPIO	Prints one line.
	DMSPIOCC	Puts CCWs and data into the caller's buffer.
	DMSPIOSI	Prints the caller's buffer, issues an SIO to the virtual printer, and analyzes the resulting status.
DMSPNT	DMSPNT	Places the address of a file status table entry in the active file table (if necessary), and sets the read pointer or write pointer for that file to a given item number within the file.
DMSPRT	DMSPRT	Prints CMS files.
DMSPRV	DMSPRV	Copies procedures from the DOS/VS system procedure library to a specified output device.
DMSPUN	DMSPUN	Punches CMS files to the virtual card punch.
DMSQRY	DMSQRY	Processes the QUERY command. Displays at the user's terminal, the status of various CMS functions and tables.
DMSRDC	READCARD	Reads cards and assigns the indicated filename.
DMSRNE	DMSRNE	Provides an interface for the CMS Editor RENUM subcommand, which renumbers files with filetypes of VSBASIC and FREEFORT.
DMSRNM	DMSRNM	Processes the RENAME command. Changes the fileid of the specified file.
DMSROS	DMSROS ROSACC	Accesses OS disks.
	DMSROS+4 ROSSTT	Verifies the existence of OS disks.

Module Name	Entry Points	Function
DMSROS (cont.)	DMSROS+8 ROSRPS	Reads OS disks.
	DMSROS+12 ROSFIND	Finds a member in an OS PDS.
	DMSROS+16 ROSNTPTB	Performs NOTE, POINT, and BSP functions.
DMSRRV	DMSRRV	Provides the capability to copy (to an output device) modules residing on DOS system or private relocatable libraries.
DMSSAB	DMSSAB	Processes OS ABEND macros.
DMSSBD	DMSSBD	Accesses data set records directly by item number. It converts record identifications given by OS BDAM macros into item numbers and uses these item numbers to access records.
DMSSBS		Processes OS BSAM READ and WRITE macros.
	DMSSBSRT	Entry for error return from call to DMSSBD.
DMSSCN	DMSSCN	Transforms the input line from a series of arguments to a series of 8-byte parameters.
DMSSCR	DMSSCR	Loads display buffers and issues a macro resulting in a CP DIAGNOSE to write to the display terminal.
DMSSCT	DMSSCTNP	Processes OS POINT, NOTE, CHECK, and FIND (type C) macros.
	DMSSCTCK	Processes OS CHECK macro.
	DMSSCTCE	Handles QSAM I/O errors for DMSSQS and PDS and keys errors for DMSSOP.
DMSSEB	DMSSEB	Calls device I/O routines to do I/O and sets up ECB and IOB return codes.
DMSSEG	DMSSEG	Contains a table of VCONS for CMS saved segment entries.
DMSSET	DMSSET	Processes the SET command.

Module Name	Entry Points	Function
DMSSLN	DMSSLN	Handles OS contents management requests issued under CMS (LINK, LOAD, XCTL, DELETE, ATTACH, EXIT).
DMSSMN	DMSSMN	Processes OS FREEMAIN and GETMAIN macros and CMS calls DMSSMNSB and DMSSMNST.
DMSSOP	DMSSOP	Processes OS OPEN and CLOSE macros.
DMSSQS	DMSSQS	Analyzes record formats and sets up the buffers for GET, PUT, and PUTX requests.
DMSSRT	DMSSRT	Arranges records within a file in descending sequential order.
DMSSRV	DMSSRV	Provides capability to copy books from a system or private source statement library to a specified output device.
DMSSSK	DMSSSK	Sets storage protect key for a specified saved system.
DMSSTG		Processes CMS calls to DMSSTGST and DMSSTGSB (STRINIT) and storage service routines.
	DMSSTGSB DMSSTGST	STRINIT.
	DMSSTGCL	OS exit reset routine.
	DMSSTGSV	Service routine to change nucleus variables.
	DMSSTGAT	Initializes storage and sets up an anchor table.
DMSSTT	DMSSTT	Locates the file status table entry for a given file and, if found, provides the caller with the address of the entry.
DMSSVN	DMSSVN	Processes the OS WAIT and POST macros.
DMSSVT	DMSSVT	Processes OS macros: XDAP, TIME, SPIE, RESTORE, BLDL, FIND, STOW, DEVTYPE, TRKBAL, WTO, WTOR, EXTRACT, IDENTIFY, CHAP, TTIMER, STIMER, DEQ, SNAP, ENQ, FREEDBUF, STAE, DETACH, CHKPT, RDJFCB, SYNAD, BACKSPACE, and STAX.

Module Name	Entry Points	Function
DMSSYN	SYNONYM	Processes the SYNONYM command. Sets up user-defined command names and abbreviations for CMS commands.
DMSTIO	DMSTIO	Reads or writes a tape record or controls tape positioning.
DMSTMA	DMSTMA	Reads an IEHMOVE unloaded PDS from tape and places it in a CMS MACLIB.
DMSTPD	DMSTPD	Reads a tape consisting of card image members of a PDS and creates CMS disk files for each member of the data set. The PDS option allows reading unblocked tapes produced by the OS IEBPTPCH utility or blocked tapes produced by the OS IEHMOVE utility. The UPDATE option provides the "./ ADD" function to blocked or unblocked tapes produced by the IEBUPDTE utility.
DMSTPE	DMSTPE	Processes the TAPE command to perform certain tape functions, such as: dump a CMS file, load a CMS file, set tape mode, scan, skip, rewind, run, FSF, FSR, BSF, BSR, ERG, and WTM.
DMSTQQ	DMSTQQ	Allocates a 200-byte first chain link (FCL) to a calling program.
	DMSTQQX	Makes a 200-byte disk area no longer needed by one program available for allocation to another program.
DMSTRK	DMSTRKA	Allocates an 800-byte disk area to a calling program.
	DMKSTRKX	Makes an 800-byte disk area that is no longer needed by one program available for allocation to another.
DMSTYP	TYPE	Processes the TYPE command. Types all or a specified part of a given file on the user's console.
DMSUPD	DMSUPD	Processes the UPDATE command. Updates source files according to specifications in update files. Multiple updates can be made, according to specifications in control files that designate the update files.

Module Name	Entry Points	Function
DMSVAN	DMSVAN	Contains table of Access Method Services nonshared (nonreentrant) modules.
DMSVAS	DMSVAS	Contains a table of Access Method Services shared (reentrant) modules.
DMSVIB	DMSVIB	Loads the CMS/VSAM saved system and pass control to the CMS/VSAM interface routine, DMSVIP.
DMSVIP	DMSVIP	Finds the CMS/DOS discontiguous shared segment (DCSS); issues all necessary DOS ASSGN statements for OS user; maps all OS VSAM macro requests to DOS specifications; equivalents, where necessary; traps all transfers of control between VSAM and the OS user and sets the appropriate operating environment flags.
DMSVPD	DMSVPD	Reads DOS, VSAM, and Access Method Services modules from a DOS PTF tape and writes the modules to the CMS user's A-disk.
DMSVSR	DMSVSR	Resets any flags or fields set by VSAM processing; purges the VSAM discontiguous shared segment.
DMSXCP	DMSXCP	Simulates the DOS EXCP function (DOS SVC 0) in the CMS/DOS environment. EXCP (Execute Channel Program) requests initiation of an I/O operation to a specific logical unit.
DMSZAP	DMSZAP	Processes the ZAP command. Provides a facility to maintain CMS LOADLIB members as written by the CMS command LKED.
DMSZAT	DMSZAT	Defines 8K-bytes of transient area.
DMSZIT	DMSZIT	Defines the end of the CMS nucleus.
DMSZNR	DMSZNR	Defines the end of NUCON (DMSNUC).
DMSZUS	DMSZUS	Defines the start of the user area.

Module	External References (Labels and Modules)											
DMSABN	ABATABND	ABNBIT	ABNERLST	ABNPAS13	ABNPSW	ABNREGS	ABNRR	ABWSECT	ADTFDA	ADTFFSTF	ADTFLG1	ADTFLG2	ADTFMIN
	ADTFQQF	ADTFROS	ADTHBCT	ADTM	ADTMFDA	ADTMFDN	ADTPQM3	ADTSECT	AFVS	AINTRTBL	AIOSECT	AOPSECT	AOUTRTBL
	ASUBFST	ASUBSECT	ASUBSTAT	AUSABRV	AUSRAREA	AUSRILST	AUSRITBL	BATFLAGS	BATLOAD	BATRUN	CMNDLINE	CONRDCNT	CONRDCOD
	CONREAD	CURRSAVE	DBGABN	DBGFLAGS	DMSABW	DMSCAT	DMSCITDB	DMSCRD	DMSCWT	DMSDBG	DMSEXCAB	DMSINTAB	DMSITSR
	DMSLADAD	DMSLADN	EGPRS	FCBFIRST	FCBNUM	FREELOWE	FVSECT	IONTABL	IOSECT	IPLPSW	KXFLAG	KXWANT	LDMSROS
	LOC	MACDIRC	MISFLAGS	NOPAGREL	NRMRET	NUCON	NUMFINRD	OLDPSW	OPSECT	OPTFLAGS	OSADTFST	OSFST	OSFSTLTH
	OSFSTNXT	PGMNPSW	PGMOPSW	RELPAGES	R0	R1	R12	R13	R14	R15	R3	R4	R5
	R6	R7	R8	SSAVE	SUBFLAG	SUBSECT	UFDBUSY						
DMSACC	ADTDTA	ADTFALUF	ADTFDA	ADTFDOS	ADTFFSTF	ADTFFSTV	ADTFLG1	ADTFLG2	ADTFLG3	ADTFMIN	ADTFRO	ADTFROS	ADTFRW
	ADTFSTC	ADTHBCT	ADTLHBA	ADTM	ADTMFDN	ADTMSK	ADTMX	ADTNUM	ADTPQM2	ADTPQM3	ADTRES	ADTSECT	ADTUSED
	ADT1ST	AFINIS	AFVS	CURRSAVE	DTAD	FVSECT	FW4	IADT	MISFLAGS	NUCON	R0	R1	R10
	R11	R12	R13	R14	R15	R2	R3	R4	R5	R6	R7	R8	R9
	UFDBUSY	WRBIT											
DMSACF	ADTADD	ADTCFST	ADTCHBA	ADTFALNM	ADTFALTY	ADTFALUF	ADTFDA	ADTFFSTF	ADTFLG1	ADTFLG2	ADTFLG3	ADTFMDRO	ADTFRO
	ADTFROS	ADTFRW	ADTFSTC	ADTFTYP	ADTHBCT	ADTLHBA	ADTM	ADTMFDA	ADTMFDN	ADTPQM2	ADTRES	ADTSECT	AFVS
	ARDTK	ATYPSRCH	DSKADR	DSKLOC	DSKLST	ERBIT	ERRCOD1	FSTIC	FSTRP	FSTSECT	FSTT	FSTWP	FVSECT
	FW4	F65535	JSR0	JSR1	NUCON	REGSAV0	REGSAV1	RWCNT	R0	R1	R10	R11	R12
	R13	R14	R15	R2	R3	R4	R5	R6	R7	R8	R9	UFDBUSY	
DMSACM	ADIOSECT	ADMSROS	ADTADD	ADTCYL	ADTDTA	ADTFLG1	ADTFLG2	ADTFLG3	ADTFMFD	ADTFQQF	ADTFRO	ADTFRW	ADTHBCT
	ADTID	ADTMFDN	ADTMSK	ADTMX	ADTNUM	ADTPQM1	ADTPQM2	ADTPQM3	ADTQQM	ADTRES	ADTROX	ADTSECT	ADTUSED
	AFVS	ARDTK	BATFLAGS	BATLOAD	CDMSROS	DIOSECT	DSKADR	DSKLOC	DSKLST	DTAD	DTADT	ERRCOD0	FFD
	FFE	FFF	FVSDSKA	FVSECT	FVSFSTIC	FVSFSTIL	F800	JSR0	LDMSROS	LOCCNT	NUCON	OSADTVTA	REGSAV0
	RWMFD	R0	R1	R10	R11	R12	R13	R14	R15	R2	R3	R4	R5
	R6	R7	R8	R9	SIGNAL	SWTCH	TBENT	UFDBUSY	UPBIT				
DMSALU	ADMSROS	ADTFDA	ADTFFSTF	ADTFLG1	ADTFLG2	ADTFLG3	ADTFMIN	ADTFQQF	ADTFRO	ADTFROS	ADTFRW	ADTFSTC	ADTFTYP
	ADTID	ADTM	ADTMFDN	ADTMSK	ADTMX	ADTPQM1	ADTPQM3	ADTQQM	ADTRES	ADTROX	ADTSECT	AFVS	CDMSROS
	FCBDSMD	FCBFIRST	FCBNEXT	FCBOSFST	FCBSECT	FLGSAVE	FVSECT	LDMSROS	LOC	NUCON	OSADTFST	OSFST	OSFSTLTH
	OSFSTNXT	REGSAV0	R0	R1	R10	R11	R12	R13	R14	R15	R2	R3	R4
	R5	R6	R7	R8	R9	STATEFST							

Module	External References (Labels and Modules)
DMSAMS	ADEVTAB ADTM ADTSECT AERASE ASCANN ASTATE ASTATEW ATABEND BGCOM CMSAMS COMNAME DOSDD DOSDEV DOSDSMD DOSDUM DOSEXTNO DOSEXTTB DOSNEXT DOSSECT DOSVOLNO DOSVOLTB DOSYSXXX DTAD DTAS FSTFV FSTIL FSTM FSTN FSTSECT LTK LUBPT MISFLAGS NUCON PIBPT PUBPT RELPAGES R0 R1 R10 R11 R12 R13 R14 R15 R2 R3 R4 R5 R6 R7 R8 R9 SYSNAMES VMSIZE
DMSARE	ADTDTA ADTFLG1 ADTFLG2 ADTFLG3 ADTFRO ADTFROS ADTFRW ADTFSTC ADTM ADTSECT AFINIS AUPDISK DTAD NUCON R0 R1 R10 R12 R14 R15 R2 R3 R4 R5 R6 R7 R8 R9
DMSARN	ADTFLG1 ADTFRW ADTM ADTMX ADTSECT AOPSECT ASTRINIT BATFLAGS BATRUN COMPSWT FCBBUFF FCBBYTE FCBCATML FCBCLOSE FCBDD FCBDEV FCBFORM FCBINIT FCBIOSW FCBITEM FCBPROC FCBPROCC FCBPROCO FCBREAD FCBSECT FSTL FSTM FSTSECT IOBCSW IOBIN IOBIOFLG MISFLAGS NUCON OSSFLAGS RELPAGES R0 R1 R10 R11 R12 R13 R14 R15 R2 R3 R4 R5 R6 R7 R8 R9
DMSARX	AADTLKW ADTFLG1 ADTFRW ADTM ADTMX ADTSECT CMNDLINE COMPSWT DEVICE DMSARD FCBBUFF FCBBYTE FCBCATML FCBCLOSE FCBDD FCBDEV FCBDSK FCBDSNAM FCBFORM FCBINIT FCBIOSW FCBITEM FCBPROCC FCBRDR FCBREAD FCBSECT FCBTAP FREELOWE FSTFV FSTIL FSTL FSTM FSTSECT IOBCSW ICBIN IOBIOFLG MAINHIGH MISFLAGS NUCON OPSECT OSIOTYPE OSSFLAGS RELPAGES R0 R1 R10 R11 R12 R13 R14 R15 R2 R3 R4 R5 R6 R7 R8 R9
DMSASM	AADTLKW ADTFLG1 ADTFRW ADTM ADTMX ADTSECT CMNDLINE COMPSWT DEVICE DMSASD FCBBUFF FCBBYTE FCBCATML FCBCLOSE FCBDD FCBDEV FCBDSK FCBDSNAM FCBFORM FCBINIT FCBIOSW FCBITEM FCBPROCC FCBRDR FCBREAD FCBSECT FCBTAP FREELOWE FSTFV FSTIL FSTL FSTM FSTSECT IOBCSW ICBIN IOBIOFLG MAINHIGH MISFLAGS NUCON OPSECT OSIOTYPE OSSFLAGS RELPAGES R0 R1 R10 R11 R12 R13 R14 R15 R2 R3 R4 R5 R6 R7 R8 R9
DMSASN	ADEVTAB ADTDTA ADTFDOS ADTFLG1 ADTFLG2 ADTFRO ADTFROS ADTFRW ADTSECT ASYSREF BGCOM DEVTAB DTAD DTADT NUCON PUBPT R0 R1 R10 R11 R12 R13 R14 R15 R2 R3 R4 R5 R6 R7 R8 R9 TAPE1 TAPE4
DMSAUD	ADTADD ADTDTA ADTFDA ADTFLG3 ADTHBCT ADTLAST ADTMFDA ADTMFDN ADTMSK ADTNUM ADTPQM1 ADTPQM2 ADTSECT AFVS ATRKLKP ATRKLKPX AWRTK DSKADR DSKLOC DSKLST DTADT FFD FFE FFF FINISLST FVSDSKA FVSECT F800 NUCON REGSAVO RWCNT RWFSTRG RWMFD R0 R1 R10 R11 R12 R13 R14 R15 R2 R3 R4 R5 R6 R7 R8 R9 UFDBUSY UPBIT
DMSBAB	BGCOM IJBABTAB NUCON PCPTR PIBADR PIBPT PIBSAVE PIK R0 R1 R10 R12 R13 R14 R15 R2 R3 R4 R5 R6 R8 R9 SVEARA SVEPSW SVEPSW2 SVEROF SVER00 SVER01 SVER09 SYSCOM

Module	External References (Labels and Modules)												
DMSBOP	ACBDDNM	ACBERFLG	ACBIN	ACBINFLG	ACBMACR1	ACBOLIGN	ACBOUT	ACBSTSKP	ADTFDOS	ADTFLG1	ADTFLG2	ADTFLG3	ADTFMFD
	ADTFRO	ADTFROS	ADTFRW	ADTSECT	AERASE	ASTATE	ASYSREF	BGCOM	CMSVSAM	DOSBLKSZ	DOSBUFF	DOSDD	DOSDEV
	DOSDSMD	DOSDUM	DOSEXT	DOSEXTCT	DOSFORM	DOSINIT	DOSNEXT	DOSOP	DOSOSFST	DOSSECT	DOSSYS	DOSUCAT	DOSUCNAM
	DOSYSXXX	FSTIC	FSTM	FSTSECT	IJBFLG04	IKQACB	LUBPT	NICLPT	NUCON	OSFST	OSFSTFM	OSFSTRFM	OSFSTXNO
	OSFSTXTN	PIBPT	PUBADR	PUBCUU	PUBDEVT	PUBPT	PUBTAPM1	PUBTAPM2	PUBTAP7	RMSROPEN	R0	R1	R10
	R11	R12	R14	R15	R2	R3	R4	R5	R6	R7	R8	R9	SYSCOM
	SYSNAMES	VMSIZE											
DMSBRD	AACTFREE	AACTLKP	AFTADT	AFTCLA	AFTCLB	AFTCLD	AFTCLN	AFTDBA	AFTDBD	AFTDBN	AFTFBA	AFTFCL	AFTFCLA
	AFTFLG	AFTFST	AFTFV	AFTIC	AFTID	AFTIL	AFTIN	AFTRD	AFTRP	AFTSECT	AFTWRT	ARDTK	AUSRAREA
	DISK$SEG	DMSLFS	FSTFV	FSTIC	FSTRP	FSTSECT	NUCON	REGSAV3	RWFSTRG	R0	R1	R10	R11
	R12	R13	R14	R15	R2	R3	R4	R5	R6	R7	R8	R9	STATEFST
	STATER0	VMSIZE											
DMSBSC	AADTLKW	ADTFLG1	ADTFRW	ADTM	ADTMX	ADTSECT	ASTRINIT	CURRSAVE	EGPRS	FREELOWE	FSTFV	FSTIL	FSTL
	FSTM	FSTSECT	MAINHIGH	MISFLAGS	NUCON	OLDPSW	OVIND	RELPAGES	R0	R1	R10	R11	R12
	R13	R14	R15	R2	R3	R4	R5	R6	R7	R8	R9	SSAVE	
DMSBTB	ABATABND	ABATLIMT	ABATPROC	AFVS	ALDRTBLS	AUSRAREA	BATDCMS	BATFLAGS	BATFLAG2	BATLOAD	BATNOEX	BATRUN	BATUSEX
	FVSECT	FVSFSTIC	FVSFSTIL	LOCCNT	NUCON	R0	R1	R12	R14	R15	R2	R3	R4
	R5	R8	TBENT										
DMSBTP	ABNBIT	ADMSCRD	AFVS	ASCANN	BATCPEX	BATDCMS	BATFLAGS	BATFLAG2	BATMOVE	BATNOEX	BATRERR	BATSTOP	BATTERM
	BATUSEX	BATXCPU	BATXLIM	BATXPRT	CMSSEG	FVSECT	IPLADDR	NUCON	NUMFINRD	R0	R1	R10	R11
	R12	R13	R14	R15	R2	R3	R4	R5	R6	R7	R8	R9	SYSNAME
	SYSNAMES	UFDBUSY											
DMSBWR	AACTFREE	AACTFRET	AACTLKP	ADTDTA	ADTFLG1	ADTFLG3	ADTFRW	ADTFSTC	ADTMX	ADTNACW	ADTSECT	AFTADT	AFTCLA
	AFTCLB	AFTCLD	AFTCLDX	AFTCLN	AFTCLX	AFTD	AFTDBA	AFTDBC	AFTDBD	AFTDBN	AFTFBA	AFTFCL	AFTFCLA
	AFTFCLX	AFTFLG	AFTFLG2	AFTFST	AFTFULD	AFTFV	AFTIC	AFTID	AFTIL	AFTIN	AFTM	AFTN	AFTNEW
	AFTOLDCL	AFTRD	AFTRP	AFTSECT	AFTWP	AFTWRT	AFVS	AQQTRK	AQQTRKX	ARDTK	ATFINIS	ATRKLKP	ATRKLKPX
	AUPDISK	AWRTK	DMSLAD	DMSLFSW	FSTFV	FSTIL	FSTSECT	FSTWP	FVSECT	NUCON	REGSAV3	RWFSTRG	R0
	R1	R10	R11	R12	R13	R14	R15	R2	R3	R4	R5	R6	R7
	R8	R9	UFDBUSY	VMSIZE	WRBIT								

Module	External References (Labels and Modules)
DMSCAT	CMNDLIST FSTFINRD MSGFLAGS NOTYPING NUCON NUMFINRD R0 R1 R4
DMSCIO	ABATABND ABATLIMT BATFLAGS BATLSECT BATNCEX BATPUNC BATPUNL BATRUN BATXLIM BATXPUN CAW CSW NUCON R0 R1 R10 R11 R12 R13 R14 R15 R2 R3 R4 R5 R6 R7 R8
DMSCIT	AFVS AIOSECT ASVCSECT ATTN BATFLAG2 BATSTOP CAW CE CMSTAXE CONCCWS CONSTACK CSW CURRIOOP DBGEXEC DBGEXINT DBGFLAGS DE FSTFINRD FVSECT IOOPSW KXFLAG KXWANT LSTFINRD MSGFLAGS NOTYPING NUCON NUMFINRD NUMPNDWR OSSFLAGS OVSHO OVSON OVSSO OVSTAT PENDREAD PENDWRIT R0 R1 R12 R13 R14 R15 R2 R3 R4 R5 R6 R7 R8 R9 SVCSECT TAIEIAD TAIEMSGL TAIERSAV TAXEADDR TAXEEXIT TAXEEXTS TAXEFREQ TAXEIOL TAXEIOWS TAXELNK TAXERTNA TAXESTAT TAXETAIE TAXETSOF TSOATCNL TSOFLAGS UE WAIT WAITSAVE
DMSCLS	AERASE AFINIS ASYSREF BGCOM DOSDD DOSDSNAM DOSNEXT DOSSECT DOSYSXXX LUBPT NICLPT NUCON PIBPT PUBADR PUBCUU PUBDEVT PUBPT PUBTAPM1 R0 R1 R10 R11 R12 R13 R14 R15 R2 R3 R4 R5 R6 R7 R8 R9
DMSCMP	ADTM ADTSECT AFINIS ARDBUF NUCCN
DMSCPF	ABATPROC BALRSAVE BATCPEX BATFLAGS BATLOAD BATRUN BATUSEX CMNDLINE NUCON R0 R1 R12 R14 R15 R2 R3 R4 R5 R6 R7 R8
DMSCPY	ADTCHBA ADTFLG1 ADTFRW ADTM ADTSECT FSTD FSTFACT FSTFB FSTFV FSTIC FSTIL FSTM FSTN FSTSECT FSTYR MISFLAGS NUCON RELPAGES
DMSCRD	ABATPROC AFVS AINTRTBL AOPSECT BATFLAGS BATLOAD BATRUN CONINBLK CONINBUF CSW DMSCAT DMSCITB FSTFINRD FVSECT KXFLAG KXWSVC LSTFINRD MISFLAGS MSGFLAGS NOTYPING NUCON NUMFINRD NUMPNDWR OPSECT PENDREAD QSWITCH R0 R1 R11 R12 R13 R14 R15 R2 R3 R4 R5 R6 R8 R9 TSOATCNL TSOFLAGS WAITLST
DMSCWR	AFVS AOPSECT AOUTRTBL CONSTACK CSW DMSCITA DMSCITB FVSECT KXFLAG KXWSVC MSGFLAGS NOTYPING NUCON NUMPNDWR OPSECT PENDREAD PENDWRIT REDERRID R0 R1 R10 R11 R12 R13 R14 R15 R2 R3 R4 R5 R6 R7 R8 WAITLST
DMSCWT	AFVS AOPSECT FVSECT KXFLAG KXWSVC NUCON NUMPNDWR OPSECT PENDREAD R0 R1 R10 R11 R12 R14 R15 R9 WAITLST
DMSDBD	ADEVTAB ARGS CAW CCWPRINT CPULOG DBDDMSG DBDEXIT DBGFLAGS DBGOUT DBGRECUR DBGSECT DBGSWTCH DEC DEVTAB LASTLINE LINE LINE1 LINE1A LINE1B LINE1C NUCON PRINTER1 R0 R1 R10 R11 R14 R15 R2 R3 R4 R5 R6 R7 R8 R9 SAVE1

Module	External References (Labels and Modules)												
DMSDBG	ABNPSW	ABNREGS	ABWSECT	ADMSCRD	AIOSECT	AKILLEX	AOPSECT	ARGMAX	ARGS	ARGSAV	ARGSCT	BALRSAVE	BEGAT
	BITS	BRKPNTBL	CAW	CONHXT	CONWR	CONWRL	CSW	CURRSAVE	DBGABN	DBGEXEC	DBGEXINT	DBGFLAGS	DBGOUT
	DBGPGMCK	DBGRECUR	DBGSAV1	DBGSAV2	DBGSECT	DBGSET	DBGSWTCH	DEC	DECDEC	DMPTITLE	DMSABNRT	DMSABW	DMSCWR
	DMSCWT	DMSDBD	DMSIOWR	DMSITP	DUMPLIST	EXAMLC	EXAMLG	EXTOPSW	FIRSTDMP	FPRLOG	F0	F6	GPRLOG
	HEX	HEXHEX	INPUT	INPUTSIZ	INPUT1	IOOPSW	IPLPSW	JFLAGS	LASTDMP	LOWSAVE	MVCNT	MVCNT2	NUCON
	OPSECT	ORG	OUTPT1	PGMOPSW	PRFPOFF	PROTFLAG	RETSAV	RSTNPSW	R0	R1	R10	R13	R14
	R15	R2	R3	R4	R5	R6	R7	R8	R9	SAVE1	SAVE2	SSAVE	STOPAT
	SYMTABLE	SYMTBG	TPFUSR	TSYM	TYPFLAG	VMSIZE	WAITLIST	WAITRD	WAITSAVE	WTRDCNT	XPSW		
DMSDIO	ADIOSECT	ADTDTA	ADTFLG1	ADTFRO	ADTFRW	ADTSECT	AFVS	ANUCEND	CAW	CCWX	CCW1	CCW1A	CCW2
	CSW	DEVTYP	DIAGNUM	DIAGRET	DIOBIT	DIOFLAG	DIOFREE	DIOSECT	DOUBLE	DTAD	DTADT	ERRCODE	FREER0
	FVSECT	IOCOMM	IOOLD	IOOPSW	LASTCYL	LASTHED	NUCON	QQDSK1	QQDSK2	QQTRK	RWCCW	R0	R1
	R10	R11	R12	R13	R14	R15	R2	R4	R5	R6	R7	R8	R9
	SAVEADT	SEEKADR	SENCCW	SENSB	TOOBIG	UFDBUSY	WRTKF	XRSAVE					
DMSDLB	ADTFDOS	ADTFLG2	ADTFROS	ADTSECT	ASYSREF	BGCOM	CURRSAVE	DOSBUFSP	DCSCBID	DOSDD	DOSDDCAT	DOSDEV	DOSDSK
	DOSDSMD	DOSDSNAM	DOSDSTYP	DOSDUM	DOSEND	DOSENSIZ	DOSEXTNO	DOSEXTTB	DOSINIT	DOSJCAT	DOSNEXT	DOSOSDSN	DOSOSFST
	DOSPERM	DOSSECT	DOSUCAT	DOSUCNAM	DOSVOLNO	DOSVOLTB	DOSYSXXX	EGPRO	LUBPT	NICLPT	NUCON	PUBPT	R0
	R1	R10	R11	R12	R13	R14	R15	R2	R3	R4	R5	R6	R7
	R8	R9	SSAVE										
DMSDLK	AADTLKP	AADTLKW	ADTFLG1	ADTFRW	ADTM	ADTSECT	AERASE	AFINIS	ARDBUF	ASTATE	AWRBUF	BGCOM	COMNAME
	CSW	DOSDD	DOSDEV	DOSDSK	DOSOP	DOSOSFST	DOSSECT	FREELOWE	FSCBBUFF	FSCBD	FSCBFM	FSCBFN	FSCBFV
	FSCBITNO	FSTFB	FSTFRW	FSTFRWX	FSTFV	FSTIC	FSTIL	FSTM	FSTSECT	JOBDATE	LABLEN	NUCON	OSFST
	OSFSTDSK	OSFSTXTN	PUBADR	PUBCUU	PUBDEVT	PUBPT	SYSLINE						
DMSDMP	ASYSREF	BGCOM	EOCADR	NUCON	PPEND	R0	R1	R12	R2	R3	R4	R5	R6
	R7												
DMSDOS	ACMSRET	ANCHENDA	ANCHENTP	ANCHINST	ANCHLDPT	ANCHLENG	ANCHPHLN	ANCHPHNM	ANCHSECT	ANCHSTSW	AOSRET	ARFLG	ASYSREF
	BGCOM	CMSVSAM	COMNAME	CURRSAVE	DACTIVE	DIRC	DIRLL	DIRN	DIRNAME	DIRTT	DMSFCH	DMSXCP	FCBLENG
	FCHTAB	FREELOWE	IJBABTAB	IJBCCWT	IJBFTTAB	INTINFO	JCSW2	JCSW4	JOBDATE	LTK	MAINHIGH	MAINLIST	MAINSTRT
	NOTEXT	NUCON	NUCRSV3	PCPTR	PIBADR	PIBFLG	PIBPT	PIBSAVE	PIB2PTR	PIK	PNOTFND	PPBEG	PPEND
	R0	R1	R10	R11	R12	R13	R14	R15	R2	R3	R4	R5	R6
	R7	R8	R9	SVEARA	SVEPSW	SVEPSW2	SVER0F	SVER00	SVER09	SYSCOM	SYSNAMES	VMSIZE	
DMSDSK	ABATABND	ADTFTYP	ADTSECT	AERASE	AFINIS	AFVS	ARDBUF	ASTATE	ATYPSRCH	AUPDISK	AWRBUF	BATDCMS	BATFLAGS
	BATFLAG2	BATRUN	FINISLST	FSTDBC	FSTFV	FSTIC	FSTIL	FSTM	FSTN	FSTSECT	FSTT	FVSECT	FVSFSTM
	F65535	F800	NUCON	R0	R1	R13	R14	R15	R2	R3	R4	R5	R6
	R7	R8	R9	STATER1	UFDBUSY	UPBIT	WRBIT						
DMSDSL	ADTFLG1	ADTFRW	ADTM	ADTSECT	AERASE	ASTATE	DA	DIRNAME	DIRR	DIRTT	FCBIOSW2	FCBITEM	FCBMVPDS
	FCBSECT	FSTL	FSTSECT	FXD	NUCON	P0	PS	R0	R1	R10	R12	R14	R15
	R2	R3	R4	R5	R8								

Module	External References (Labels and Modules)												
DMSDSV	BGCOM	COMNAME	DOSDD	DOSSECT	FREELOWE	NUCON	PUBADR	PUBCUU	PUBPT				
DMSEDC	DUALNOS	EDCB	R0	R1	R10	R13	R14	R15	R2	R3	R4	R5	R6
	R7	R8	R9	SAVEAR									
DMSEDI	ADEVTAB	AERASE	AEXTEND	AFINIS	AFSTFNRD	AINCORE	ALCHAR1	ALCHAR2	ALTLIST	ARDBUF	AREA	ATTN	ATTNLEN
	AUTOCNT	AUTOCURR	AUTOREG	AWRBUF	BLOC	BYTE	CARDINCR	CARDNO	CASEREAD	CASESW	CHNGCNT	CHNGFLAG	CHNGMSG
	CHNGNUM	CMODE	CONSOLE	CORITEM	COUNT	CRBIT	DECIMAL	DEVTAB	DITCNT	DMSSCR	EDCB	EDCT	EDLIN
	EDRET	ENDBLOC	ENDTABS	FILEMS	FLAG	FLAG2	FMODE	FNAME	FPTR	FREELEN	FSIZE	FTYPE	FV
	GETFLAG	HALF	INCRNO	INVLD	IOID	IOLIST	IOMODE	ITEM	JAR	LINE	LMCURR	LMINCR	LMSTART
	MSGFLAGS	NEWMODE	NEWNAME	NEWTYPE	NOTYPING	NUCON	PADBUF	PADCHAR	PTR1	PTR2	PTR3	RANGE	REGSAV
	REGSAVX	REPCNT	RPLIST	R0	R1	R10	R13	R14	R15	R2	R3	R4	R5
	R6	R7	R8	R9	SAVCNT	SAVCWD	SCRFLGS	SCRFLG2	SEQNAME	SERSAV	SERTSEQ	SERTSW	SIGNAL
	SPARES	STACKAT	STACKATL	STRTNO	TABLIN	TABS	TEMPTAB	TIN	TOUT	TRUNCOL	TVERCOL1	TVERCOL2	TWITCH
	TYPFLG	VERCOL1	VERCOL2	VERLEN	XAREA	XXXCWD	XYCNT	XYFLAG	YAREA	ZONE1	ZONE2		
DMSEDX	ADEVTAB	AEDLIN	AEXTEND	AFINIS	AFLAGLCC	AFSTFNRD	ALINELOC	ALTMODE	ANUMLOC	ARDBUF	ASTATE	ASTATEW	BLANK1
	BLANK2	BLANK3	BLOC	CANCCW	CARDINCR	CASESW	CHNGMSG	CMSSEG	CONSOLE	CORITEM	DEVTAB	EDCB	EDCBEND
	EDCBLTH	EDLIN	EDRET	EDWORK	ENDBLOC	ENDTABS	FLAG	FLAGLOC	FLAG2	FMODE	FNAME	FREELEN	FSTD
	FSTFINRD	FSTFMODE	FSTRECCT	FSTRECFM	FTYPE	FV	INVLD	IOAD	IOID	IOLIST	IOMODE	ITEM	JAR
	LINE	LINELOC	LMSTART	LOCCNT	MAINAD	NUCON	NUMLOC	PADBUF	PADCHAR	PTR1	PTR2	PTR3	RECS
	REPCNT	R0	R1	R10	R12	R13	R14	R15	R2	R3	R4	R5	R6
	R7	R8	R9	SCRBUFAD	SEQNAME	SPARES	SUBACT	SUBFLAG	SYSNAMES	TABS	TIN	TRUNCOL	TWITCH
	VERCOL1	VERCOL2	VERLEN	ZONE1	ZONE2								
DMSERR	ABATABND	BATFLAGS	BATRUN	CALLEE	CAW	CONCCWS	CURRSAVE	DMSCWR	DMSCWT	DMSERT	ERBL	ERDSECT	ERF1BF
	ERF1BD	ERF1SBN	ERF1SB1	ERF1TX	ERF2CM	ERF2DI	ERF2DT	ERF2PR	ERF2SI	ERLET	ERMESS	ERNUM	ERPAS13
	ERPBFA	ERPCS	ERPF1	ERPF2	ERPHDR	ERPLET	ERPNUM	ERPSBA	ERPTXA	ERSAVE	ERSBD	ERSBF	ERSBL
	ERSECT	ERSFA	ERSFL	ERSFLST	ERSSZ	ERTEXT	ERTPL	ERTPLA	ERTPLL	ERTSIZE	ERT1	ERT2	NUCON
	OLDPSW	SSAVE											
DMSERS	AACTFRET	AACTLKP	AACTNXT	ADTADD	ADTCFST	ADTCHBA	ADTFLG1	ADTFRO	ADTFRW	ADTFSTC	ADTHBCT	ADTLFST	ADTLHBA
	ADTM	ADTRES	ADTSECT	AFTADT	AFTDBC	AFTFCL	AFTFLG	AFTPFST	AFTSECT	AFVS	AQQTRKX	ARDTK	ASTATEW
	ATFINIS	ATRKLKPX	AUPDISK	DMSLAD	DMSLADW	DMSLFSW	DSKADR	DSKLOC	DSKLST	ERBIT	ERRCOD1	ERSFLAG	FSTBKWD
	FSTDBC	FSTFCL	FSTFWDP	FSTM	FSTN	FSTSECT	FSTT	FVSECT	FVSERAS0	FVSERAS1	FVSERAS2	NUCON	REGSAV1
	R0	R1	R10	R11	R12	R13	R14	R15	R2	R3	R4	R5	R6
	R7	R8	R9	SIGNAL	STATEFST	STATER1	UFDBUSY						
DMSEXC	ADTM	ADTSECT	AEXEC	AFINIS	AFVS	AOPSECT	CMSSEG	DMSLFS	EXADD	EXECFLAG	EXECRUN	EXLEVEL	EXNUM
	FILEBUFF	FILEBYTE	FILEMODE	FSTD	FSTLRECL	MISFLAGS	NUCON	OPSECT	PLIST	R0	R1	R10	R11
	R12	R13	R14	R15	R2	R3	R4	R5	R6	R7	R8	R9	SYSNAMES

Module	External References (Labels and Modules)
DMSEXT	ADTFDOS ADTFLG2 ADTFMFD ADTFROS ADTM ADTSECT AFINIS AGETCLK AOPSECT APOINT ARDBUF ASCANO ASTATE CMNDLIST CURRDATE CURRTIME EXADD EXLEVEL FSTFINRD LASTCMND LASTEXEC MSGFLAGS NOTYPING NUCON OPSECT OSRESET OSSFLAGS PREVCMND PREVEXEC R0 R1 R10 R14 R15 R2 R3 R4 R5 R6 R7 R8 R9
DMSFCH	ASTATE ASYSREF AUSRAREA BGCOM COMNAME CSW FCBDD FCBDEV FCBDSK FCBDSNAM FCBINIT FCBOP FCBOSFST FCBSECT FREELOWE HIPHAS HIPROG IHADEB LOC LUBPT NUCON OSFST OSFSTDSK OSFSTXTN PO PS PUBPT R0 R1 R10 R11 R12 R13 R14 R15 R2 R3 R4 R5 R6 R7 R8 R9
DMSFET	ALDRTBLS AUSRAREA BGCOM COMNAME DACTIVE DIRN DIRNAME FCHAPHNM FCHLENG FCHOPT FCHTAB HIPHAS IJBFTTAB LASTIOC LOCCNT NOTEXT NUCON PNOTFND R0 R1 R12 R14 R15 R2 R3 R4 R5 R6 R7 STRTADDR SYSCCM TBENT
DMSFLD	ABATABND ASTATE BATDCMS BATFLAGS BATFLAG2 BATRUN CURRSAVE EGPR0 FCBBLKSZ FCBCASE FCBCATML FCBCON FCBDD FCBDEV FCBDSK FCBDSMD FCBDSNAM FCBDSORG FCBDSTYP FCBENSIZ FCBFIRST FCBINIT FCBIOSW FCBLRECL FCBMEMBR FCBMODE FCBNUM FCBOSDSN FCBPCH FCBPROC FCBPTR FCBRDR FCBRECFM FCBSECT FCBTAP FCBTAPID FCBXTENT JFCBIND2 JFCBUFNO JFCKEYLE JFCLIMCT JFCOPTCD LOC NUCON R0 R1 R10 R11 R12 R13 R14 R15 R2 R3 R4 R5 R6 R7 R8 R9 SSAVE
DMSFNC	DMSBWR DMSCAT DMSCIOSI DMSCITDB DMSCPF DMSCRD DMSCWR DMSCWT DMSDBG DMSERR DMSEXC DMSFET DMSFREES DMSFREEX DMSFRETS DMSITET DMSLADAD DMSLDRA DMSLOA DMSMOD DMSPIO DMSPIOCC DMSPIOSI DMSSTGAT DMSVSR
DMSFNS	AACTFRET AACTLKP ADIOSECT ADTADD AERASE AFVS AQQTRKX ARDTK ATRKLKPX ATYPSRCH AUPDISK AWRTK BALRSAVE DEVTYP DIOCSW DIOSECT DISK$SEG DMSLFSW DSKLOC DSKLST FINISLST FNBIT FVSECT NUCON REGSAV3 RWFSTRG R0 R1 R10 R11 R12 R13 R14 R15 R2 R5 R6 R7 R8 R9 SENSB STATEFST SUBFLAG SUBINIT UFDBUSY
DMSFOR	ADEVTAB ADTCYL ADTDTA ADTFALUF ADTFDA ADTFFSTF ADTFLG1 ADTFLG2 ADTFQQF ADTFRO ADTFRW ADTHBCT ADTID ADTLAST ADTLEFT ADTLHBA ADTM ADTMSK ADTNUM ADTPQM1 ADTPQM2 ADTPQM3 ADTQQM ADTRES ADTSECT ADTUSED ADT1ST ARDTK AUPDISK AWRTK DTAD NUCON R0 R1 R10 R11 R12 R13 R14 R15 R2 R3 R4 R5 R6 R7 R8 R9
DMSFRE	ABNPSW ABNREGS ABWSECT ACALL AFREETAB ASSTAT ASVCSECT AUSRAREA BLOCKLEN CALLER CODE CODE203 CURRSAVE DMSABNGO DMSABW DMSFRT DMSNUCU FLAGS FLCLN FLHC FLNU FLPA FRDSECT FREEFLG1 FREEFLG2 FREEHN FREEHU FREELN FREELOWE FREELOW1 FREELU FREESAVE FRF1B FRF1C FRF1E FRF1H FRF1L FRF1M FRF1N FRF1V FRF2CKE FRF2CKT FRF2CKX FRF2CL FRF2NOI FRF2SVP LOCCNT NUCON NUM POINTER PRFPOFF PROTFLAG SIZE SKEY SSAVE SVCAB SVCSECT SYSCODE TCODE TRNCODE USARCODE USERCODE USERKEY VMSIZE
DMSGIO	ADEVTAB BUFAD CANCCW CMDBLOK CSW CTL EDCB NUCON R0 R1 R10 R13 R14 R15 R2 R4 R5 WRCOUNT
DMSGLB	ASTATE MACLIBL NUCON R0 R1 R12 R13 R14 R15 R2 R3 R4 R5 R7 R8 TXTDIRC TXTLIBS

Module	External References (Labels and Modules)
DMSGND	ALDRTBLS ASTATE FSTD FSTDATEW NUCON R0 R1 R11 R12 R14 R15 R2 R3 R4 R5 R6 R9 TBENT
DMSGRN	R0 R1 R10 R11 R12 R13 R14 R15 R2 R3 R4 R5 R6 R7 R8 R9
DMSHDI	AIOSECT ANUCEND AUSRILST AUSRITBL IONTABL IOSECT NUCON R0 R1 R10 R12 R13 R14 R15 R2 R3 R4 R5 R6 R7 R8 R9 VMSIZE
DMSHDS	ANUCEND ASVCSECT JFIRST JLAST JNUMB NUCON R0 R1 R10 R12 R13 R14 R15 R2 R3 R4 R5 R6 R7 R8 R9 SVCSECT VMSIZE
DMSINA	AUSABRV BALRSAVE NOABBREV NOSTDSYN NUCON OPTFLAGS R0 R1 R14 R15 R2 R3 R4 R5 R6 R7 R8 R9
DMSINI	ADEVTAB CAW CC CE CHANO CONSOLE CSW DE DEVTAB DMSDBGP DMSINS DMSINSE DMSITS1 EXTNPSW INSTALID IONPSW IOOPSW IPLCCW1 IPLPSW MCKM MCKNPSW NOP NUCCN RDCONS RDDATA R0 R1 R10 R11 R12 R13 R14 R15 R2 R3 R4 R5 R6 R7 R8 R9 SDISK SEARCH SEEK SETSEC SILI SYSADDR SYSTEMID TIC WAIT WRDATA WRITE WRITE1 YDISK
DMSINM	ASUBSECT BALRSAVE CURRCPUT CURRDATE CURRVIRT NUCON R0 R1 R10 R14 R15 R2 R3 R4 R5 R8 SUBSECT TIMBUF
DMSINS	ACMSCVT ADTFDA ADTFFSTF ADTFFSTV ADTFLG1 ADTFLG3 ADTFSTC ADTSECT AEXTSECT ALDRTBLS AOPSECT ASSTAT ASTATE ASTATEXT ASYSREF AUSRAREA BATFLAGS BATFLAG2 BATIPLSS BATLOAD BATRUN BGCOM CAW CC CHANO CMNDLINE CMNDLIST CMSCVT CMSSEG CONRDCNT CONRDCOD CONREAD CURRDATE CVTMDL CVTMZOO CVTNUCB CVTOPTA CVTSECT DMSLAD DMSLCA DMSSCNN DTAD EXTSECT FREELOWE F0 GRAFDEV IONPSW IPLADDR IPLPSW LOCCNT MAINHIGH MCKM MISFLAGS MSGFLAGS NOVMREAD NUCON OPSECT OPTFLAGS PGMNPSW PRFTSYS PROTFLAG R0 R1 R10 R11 R12 R13 R14 R15 R2 R3 R4 R5 R6 R7 R9 SILI SYSNAME SYSNAMES SYSREF SYSTEMID TIMCHAR TIMER TIMINIT VMSIZE WAIT YYDDD
DMSINT	AACTLKP AEXTSECT AFTM AFTN AFTSECT AFTWP AFVS AIOSECT ACPSECT ASCBPTR ASUBFST ASUBRET ASUBSECT ASUBSTAT ASVCSECT AUSRAREA CMNDLINE CMSSEG CMSTIM CONRDCNT CONRDCOD CONREAD CONWRBUF CONWRCOD CONWRITE DMSCPF DMSLFS DMSSCNN ERRNUM EXTPSW EXTSECT FILENAME FILETYPE FINISLST FREELOWE FSTFINRD FVSECT IONTABL IOSECT JNUMB LASTCMND MISFLAGS MSGFLAGS NOABBREV NCIMPCP NOIMPEX NOPAGREL NORDYTIM NOTYPING NOVMREAD NUCON OPSECT OPTFLAGS OSRESET OSSFLAGS PLIST PREVCMND QSWITCH REDERRID RELPAGES RMSGBUF R0 R1 R10 R11 R12 R13 R14 R15 R2 R3 R4 R5 R6 R7 R9 SPIESAV STAESAV STARS STATEFST SUBACT SUBFLAG SUBSECT SVCSECT SWTCHSAV SYSNAMES TIMCHAR TIMER TIMINIT
DMSIOW	AEXTSECT CSW DEVICE EXTSECT IONPSW IOOPSW NUCON R0 R1 R10 R11 R14 R15 R2 R4 R5 R6 R7 R8 R9 TIMCHAR TIMER TIMINIT WAITSAVE

Module	External References (Labels and Modules)												
DMSITE	ABATABND	ABATLIMT	AEXTSECT	BALR	BATCPUC	BATCPUL	BATFLAGS	BATFLAG2	BATLOAD	BATLSECT	BATRUN	BATXCPU	BATXLIM
	CSW	DBGEXEC	DBGEXINT	DBGFLAGS	DBGSECT	DMSCWR	EXSAVE	EXSAVE1	EXTFLAG	EXTOPSW	EXTPSW	EXTRET	EXTSECT
	FVS	FVSECT	F0	F2	F4	F6	IONPSW	IOOPSW					
	PENDREAD	R0	R1	R10	R11	R12	R13	R14	R15	R2	R3	R7	SAVEXT
	SCAW	STIMEXIT	TIMCCW	TIMCHAR	TIMER	TIMINIT	TSOATCNL	TSOFLAGS	TYPLIST	UFDBUSY	XPSW		
DMSITI	ABNPSW	ABNREGS	ABWSECT	ADIOSECT	AFVS	AIOSECT	CSW	DEVICE	DIOSECT	DMSABNGO	DMSABW	FVSECT	HOLD
	IONTABL	IOOLD	IOOPSW	IOPSW	IOSAVE	IOSECT	KXFLAG	KXWANT	NEXTO	NUCON	OLDEST	R0	R10
	R11	R12	R13	R14	R15	R3	R4	R5	R6	R7	R8	R9	TSOATCNL
	TSOFLAGS	UFDBUSY	VSTRANGE	WAIT									
DMSITP	ABNPSW	ABNREGS	ABWSECT	AFVS	APGMSECT	ASYSREF	AUPIE	BGCOM	CALLEE	CURRSAVE	DMSABNGO	DMSABW	FVSECT
	IJBABTAB	INTINFO	LTK	NUCON	OPSW	PCPTR	PGMNPSW	PGMOPSW	PGMSECT	PIBADR	PIBPT	PIBSAVE	PICADDR
	PIE	PIK	PSAVE	R0	R1	R10	R11	R12	R13	R14	R15	R2	R3
	R4	R5	R6	R7	R8	R9	SCBPTR	SSAVE	SVEARA	SVEPSW	SVEPSW2	SVER00	SVER09
	SYSCOM	TPFUSR	TYPFLAG	UFDBUSY									
DMSITS	ABNPSW	ABNREGS	ABWSECT	AERR	AFVS	ASVCSECT	AWAIT	CALLEE	CALLER	CHKWRD1	CHKWRD2	CMSSEG	CODE
	CODE203	CURRSAVE	DMSABNGO	DMSABW	DMSCWT	DMSERR	DMSFNC	DMSFNC3	DMSMOD	EFPRS	EGPRS	EGPR0	EGPR11
	EGPR14	EGPR15	EGPR2	ERRET	FVSECT	F0	F6	ITSBIT	KEYMAX	KEYP	KEYS	KXFLAG	KXWANT
	KXWSVC	LASTTMOD	LENOVS	MCKM	MISFLAGS	NRMRET	NUCON	OLDPSW	OVSECT	PRFPOFF	PRFTSYS	PRFUSYS	PROTFLAG
	R0	R1	R10	R11	R12	R13	R14	R15	R2	R3	R4	R5	R6
	R7	R8	R9	SSAVE	SSAVENXT	SSAVEPRV	SSAVESZ	STRTADDR	SVCOPSW	SYSNAMES	TPFERT	TPFNS	TPFR01
	TPFSVO	TPFUSR	TSOATCNL	TSOFLAGS	TYPFLAG	UFDBUSY	USAVEPTR	USAVESZ					
DMSLAD	ADTFDA	ADTFFSTV	ADTFLG1	ADTFLG2	ADTFRO	ADTFROS	ADTFRW	ADTFVS	ADTHBCT	ADTLEFT	ADTM	ADTPSTM	ADTPTR
	ADTRES	ADTSECT	IADT	REGSAV0	R0	R1	R10	R12	R13	R14	R15	R2	R3
	R4	R5	R6	R7	R8	R9	SVLAD	SVLADW					
DMSLAF	ADTFLG1	ADTFRW	ADTM	ADTMX	ADTSECT	AFTADT	AFTFB	AFTFLG	AFTFSF	AFTFST	AFTLD	AFTM	AFTN
	AFTPFST	AFTPTR	AFTSECT	AFTT	AFTUSED	FSTL	FSTSECT	R0	R1	R11	R12	R13	R14
	R15	R2	R3	R4	R5								
DMSLBM	AADTLKW	ADTFLG1	ADTFRW	ADTM	ADTSECT	FREELOWE	FSTFV	FSTIC	FSTIL	FSTM	FSTSECT	MISFLAGS	NUCON
	RELPAGES	R0	R1	R10	R11	R14	R15	R2	R3	R4	R5	R6	R7
	R8	R9											
DMSLBT	AADTLKW	MISFLAGS	NUCON	RELPAGES	R0	R1	R10	R11	R12	R13	R14	R15	R2
	R3	R4	R5	R6	R7	R8	R9						

Module	External References (Labels and Modules)
DMSLDR	ACMSRET AERASE AFINIS ALDRTBLS APRILB APSV ARDBUF ASCANN ASTATE AUSRAREA BATFLAGS BATLOAD BRAD CALLEE CLOSELIB CMNDLIST COMMONEX CRDPTR CURRSAVE DMSLGTA DMSLGTB DMSLIB DMSLIO DMSLSBA DMSLSBB DMSLSBC DMSLSBD DMSLSY DYLD DYNAEND EGPR1 ENDCDADR ENTADR ENTNAME ESD1ST ESIDTB FINIS FLAGS FLAG1 FLAG2 FREELOWE FRSTSDID FSTXTADR GPRSAV LDRADDR LDRFLAGS LDRRTCD LDRST LOCCNT LOCCT LUNDEF MAINHIGH MEMBCUND NEED NOAUTO NODUP NOINV NOLIBE NOREP NOSLCADR NUCON NUMBYTE NXTSYM OSRESET OSSFLAGS OUTBUF OUTPUT PARMLIST PLISTSAV PREXIST PRHOLD PRVCNT PSW READBUF REG13SAV RESET RETREG RLDCONST R0 R1 R10 R11 R12 R13 R14 R15 R2 R3 R4 R5 R6 R7 R8 R9 SAV67 SPEC SSAVE START STRTADDR SYSUT1 TBENT TBLCT TBLREF TEMPST TMPLOC TPFUSR TXTDIRC TYPFLAG VMSIZE
DMSLDS	ADMSROS ADTCYL ADTFLG1 ADTFLG2 ADTFRO ADTFROS ADTFRW ADTID ADTM ADTSECT CSW FCBIOSW2 FCBMEMBR FCBMVPDS FCBOSDSN FCBSECT NUCON OSADTDSK OSADTVTA OSADTVTB PO R0 R1 R10 R11 R12 R13 R14 R15 R2 R3 R4 R5 R6 R7 R8 R9
DMSLFS	ADMSROS ADTCHBA ADTFDA ADTFFSTV ADTFLG1 ADTFLG2 ADTFLG3 ADTFRO ADTFROS ADTFRW ADTFTYP ADTHBCT ADTLFST ADTLHBA ADTM ADTMX ADTRES ADTSECT DISK$SEG DMSLAD DMSLADN DMSSTTR FVSECT NUCON REGSAV0 R0 R1 R10 R11 R12 R13 R14 R15 R2 R3 R4 R5 R6 R7 R8 R9 SVLFS
DMSLGT	APSV ARDBUF DMSLDRD FILE FMODE FNAME FTYPE LDRST NUCON OUTBUF RADD READBUF RFIX RITEM RLENG RNUM R0 R1 R10 R12 R13 R14 R15 R3 R4 R5 R6 R7 R8 R9 SPEC TXTDIRC TXTLIBS TYPE
DMSLIB	AFINIS APOINT APSV ASTATE CLOSELIB DMSLDRD DYMBRNM FILE FINIS FLAGS FLAG2 FMODE FNAME FTYPE LDRST NOAUTO NOLIBE NUCON NUMBYTE OSSFLAGS OUTBUF RADD READBUF RITEM RLENG RNUM R0 R1 R11 R12 R13 R14 R15 R5 R7 SETLIB SPEC TBLCT TBLREF TXTDIRC TXTLIBS TYPE
DMSLIO	AERASE AFINIS ALIASENT APSV AWRBUF DSKAD DSKLIN DYLD FILE FLAG1 FLAG2 FNAME LDRADDR LDRST NOERASE NOMAP NUCON OSSFLAGS OUTBUF OUTPUT PACK PARMLIST R0 R1 R10 R11 R13 R14 R15 R2 R3 R4 TYPE TYPEAD TYPLIN UNPACK
DMSLKD	AADTLKW ADTM ADTSECT FSTFV FSTIL FSTM FSTSECT MISFLAGS NUCON RELPAGES R0 R1 R10 R11 R12 R14 R15 R2 R3 R4 R5 R6 R7 R9
DMSLLU	ADTFLG1 ADTFRW ADTSECT AERASE AFINIS ASYSREF AWRBUF BGCOM LUBPT NICLPT NUCON PUBADR PUBCUU PUBDEVT PUBDSKM PUBPT R0 R1 R10 R11 R12 R14 R15 R2 R3 R4 R5 R6 R7 R8
DMSLOA	ALDRTBLS AUSRAREA DMSLDRB FSTXTADR LDRADDR LDRFLAGS LOCCNT NOAUTO NOERASE NOINV NOLIBE NOMAP NOREP NUCON PRHOLD R0 R1 R12 R14 R15 R2 R6 STRTADDR SYSREF TBENT TYPE
DMSLSB	APSV AUSRAREA BATFLAGS BATLOAD BRAD DMSLDRC DMSLDRD ENDCDADR ENTNAME FLAGS FLAG1 FLAG2 FREELOWE FRSTSDID FSTXTADR LASTTMOD LDRST LOCCT MAINHIGH NOAUTO NODUP NOINV NOLIBE NOMAP NOREP NUCON OUTBUF RESET RETT R0 R1 R10 R11 R12 R13 R14 R15 R2 R3 R4 R5 R6 R7 R8 R9 START STRTADDR TMPLOC TYPE

Module	External References (Labels and Modules)												
DMSLST	ADTFDA	ADTFLG1	ADTFRO	ADTFRW	ADTID	ADTM	ADTSECT	AERASE	NUCON	R0	R1	R10	R11
	R12	R13	R14	R15	R2	R3	R4	R5	R6	R7	R8	R9	
DMSLSY	DSYM	GET1	JSYM	NUCON	NXTSYM	R0	R1	R14	R15				
DMSMDP	ALDRTBLS	ASTATE	FSTIC	FSTSECT	NUCON	R0	R1	R14	R15	R2	R3	R4	TBENT
DMSMOD	AERASE	AFINIS	AFVS	ALDRTBLS	ARDBUF	ARDTK	ASTATE	ASTATEW	AUSRAREA	AWRBUF	DSKLOC	DSKLST	FREELOWE
	FRSTLOC	FVSECT	FVSFSTAD	FVSFSTCL	FVSFSTFV	FVSFSTIC	FVSFSTIL	F65535	LASTLMOD	LASTTMOD	LDRFLAGS	LOCCNT	NUCON
	PRFTSYS	PRFUSYS	PROTFLAG	REGSAV3	RWCNT	R0	R1	R10	R11	R12	R13	R14	R15
	R2	R3	R4	R5	R6	R7	R8	R9	STRTADDR	SUBFLAG	TBENT		
DMSMVE	ADTFLG1	ADTFRW	ADTSECT	BATFLAGS	BATMOVE	DA	DDNAM	FCBBLKSZ	FCBDD	FCBDEV	FCBDSK	FCBDSMD	FCBDSNAM
	FCBINIT	FCBIOSW2	FCBITEM	FCBLRECL	FCBMVPDS	FCBOP	FCBOPCB	FCBOSFST	FCBRECFM	FCBSECT	FCBTAP	FCBTAPID	FSTFV
	FSTIL	FSTSECT	IHADEB	NUCON	OSFST	OSFSTBLK	OSFSTLRL	OSFSTRFM	PS	R0	R1	R10	R12
	R13	R14	R15	R2	R3	R4	R5	R6	R7	R8	R9		
DMSNCP	FSTD	FSTFMODE	R0	R1	R10	R11	R12	R13	R14	R15	R2	R3	R4
	R5	R6	R8	R9									
DMSNUC	DMSINALT	DMSINA1S											
DMSOLD	AERASE	AFINIS	ALDRTBLS	APRILB	APSV	ARDBUF	ASCANN	ASTATE	AUSRAREA	AWRBUF	BATFLAGS	BATLOAD	BRAD
	CLOSELIB	CMNDLIST	COMMONEX	CRDPTR	DMSLGTA	DMSLGTB	DMSLIB	DMSLSBA	DMSLSBB	DMSLSBC	DMSLSBD	DMSLSY	DYLD
	DYNAEND	ENDCDADR	ENTADR	ENTNAME	ESD1ST	ESIDTB	FINIS	FLAGS	FLAG1	FLAG2	FREELOWE	FSTXTADR	GPRSAV
	LDRADDR	LDRFLAGS	LDRRTCD	LDRST	LOCCNT	LOCCT	LUNDEF	MEMBOUND					
	NOREP	NOSLCADR	NUCON	NUMBYTE	NXTSYM	OSRESET	OSSFLAGS	OUTBUF	OUTPUT	PARMLIST	PLISTSAV	PREXIST	PRHOLD
	PRVCNT	READBUF	REG13SAV	RESET	RETREG	RLDCONST	R0	R1	R10	R11	R12	R13	R14
	R15	R2	R3	R4	R5	R6	R7	R8	R9	SAV67	SPEC	STRTADDR	SYSUT1
	TBENT	TBLCT	TBLREF	TEMPST	TMPLOC	TXTDIRC	WORKFILE						
DMSOPL	ASYSREF	BGCOM	DOSDD	DOSNEXT	DOSSECT	DOSSYS	LUBPT	NUCON	R0	R1	R12	R15	R2
	R3	R4	R5	R6	R7	R8	R9						
DMSOPT	BGCOM	JCSW3	JCSW4	NUCON	R0	R1	R10	R11	R12	R14	R15	R2	SOB1
DMSOR1	NUCON	R0	R1	R12	R15	R2	R5	R6					
DMSOR2	R1	R12											
DMSOR3	R1	R12	R14										
DMSOVR	ADMSOVS	ASVCSECT	DMSOVS	LENOVS	NUCON	OVAPF	OVBPF	OVF1F	OVF1FS	OVF1GA	OVF1GE	OVF1GS	OVF1ON
	OVF1PA	OVF2CM	OVF2NR	OVF2OS	OVF2WA	OVSECT	OVSHO	OVSON	OVSSO	OVSTAT	R0	R1	R12
	R14	R15	R3	R4	R5	R6	R7	R8	SVCSECT				

Module	External References (Labels and Modules)												
DMSOVS	ASVCSECT	CURRSAVE	EFPRS	EGPRS	EGPR0	EGPR15	NUCON	OVAPF	OVBPF	OVF1ON	OVSAFT	OVSHO	OVSON
	OVSSO	OVSTAT	RFPRS	RGPRS	RGPR8	R0	R1	R12	R13	R14	R15	R3	R4
	R5	R6	R7	R8	SSAVE	SVCOUNT	SVCSECT	IPFSVO	TYPFLAG	VMSIZE	XCOUNT	XGPR0	XGPR1
	XGPR15												
DMSPIO	ABATABND	ABATLIMT	BATFLAGS	BATLSECT	BATNOEX	BATPRTC	BATPRTL	BATRUN	BATXLIM	BATXPRT	CAW	CSW	NUCON
	R1	R10	R11	R12	R13	R14	R15	R2	R3	R4	R5	R6	R7
	R8	R9											
DMSPNT	AACTFREE	AACTLKP	AFTIC	AFTRP	AFTSECT	AFTWP	DMSLFS	F65535	NUCON	REGSAV3	R0	R1	R11
	R14	R15	R2	R4	R5	R6							
DMSPRT	ADMSPIOC	AFINIS	ARDBUF	ASTATE	INSTALID	NUCON	R0	R1	R10	R11	R12	R13	R14
	R15	R2	R3	R4	R5	R6	R7	R8	R9				
DMSPRV	AERASE	AFINIS	ASYSREF	AWRBUF	BGCCM	LUBPT	NUCON	PUBADR	PUBCUU	PUBPT	R0	R1	R10
	R12	R14	R15	R2	R3								
DMSPUN	ADTID	ADTSECT	AFINIS	ARDBUF	ASTATE	FVSFSTAD	NUCON	R0	R1	R10	R11	R12	R13
	R14	R15	R2	R3	R4	R5	R6	R7	R8	R9	STATEFST		
DMSQRY	ADTCYL	ADTDTA	ADTFDOS	ADTFLG1	ADTFLG2	ADTFLG3	ADTFRO	ADTFROS	ADTFRW	ADTFSTC	ADTID	ADTM	ADTMX
	ADTNUM	ADTSECT	AEXTSECT	AFVS	AINTRTBL	ALDRTBLS	AOUTRTBL	ASYSREF	AUSABRV	CMSSEG	DTAD	DTADT	EXTSECT
	FCBDD	FCBDEV	FCBDSNAM	FCBDSTYP	FCBFIRST	FCBNUM	FCBSECT	FCBTAPID	FVSECT	MACLIBL	MISFLAGS	MSGFLAGS	NOABBREV
	NOIMPCP	NOIMPEX	NOPAGREL	NORDYTIM	NOSTDSYN	NUCON	OPTFLAGS	PRFPOFF	PROTFLAG	REDERRID	R0	R1	R10
	R11	R12	R13	R14	R15	R2	R3	R4	R5	R6	R7	R8	R9
	SYSNAMES	TIMCCW	TIMCHAR	TXTLIBS									
DMSRDC	ABATABND	AERASE	AFINIS	ASCANN	ASTATEW	AWRBUF	BATDCMS	BATFLAGS	BATFLAG2	BATRUN	NUCON	R0	R1
	R10	R11	R14	R15	R2	R3	R4	R5	R6	R7	R8	R9	
DMSRNE	AERASE	AFINIS	ARDBUF	AWRBUF	NUCON	R0	R1	R10	R12	R13	R14	R15	R2
	R3	R4	R5	R6	R7								
DMSRNM	AACTLKP	ADTCHBA	ADTFLG1	ADTFRO	ADTFRW	ADTFTYP	ADTM	ADTSECT					
	ATYPSRCH	AUPDISK	ERBIT	ERRCOD1	ERSFLAG	FSTM	FSTN	FSTSECT	FSTT	FVSECT	FVSERAS0	FVSERAS1	FVSERAS2
	NUCON	REGSAV1	R0	R1	R10	R11	R12	R13	R14	R15	R2	R3	R4
	R5	R6	R7	R8	R9	STATEFST	UFDBUSY						
DMSROS	ADTCYL	ADTDTA	ADTFDOS	ADTFLG1	ADTFLG2	ADTFLG3	ADTFROS	ADTM	ADTSECT	CSW	DTAD	FCBBLKSZ	FCBDSMD
	FCBDSNAM	FCBDSTYP	FCBFIRST	FCBIOSW2	FCBLRECL	FCBMVPDS	FCBNEXT	FCBOP	FCBOSDSN	FCBOSFST	FCBPROC	FCBRECFM	FCBSECT
	FILEBUFF	FILEBYTE	FILENAME	FILEREAD	LOC	NUCON	OPSECT	OSADTDSK	OSADTFST	OSADTVTA	OSADTVTB	OSFST	OSFSTALT
	OSFSTBLK	OSFSTCHR	OSFSTDEK	OSFSTDSK	OSFSTDSN	OSFSTEND	OSFSTEX4	OSFSTFLG	OSFSTFM	OSFSTFVF	OSFSTLRL	OSFSTLTH	OSFSTMVL
	OSFSTNTE	OSFSTNXT	OSFSTRFM	OSFSTRSW	OSFSTTRK	OSFSTTYP	OSFSTUMV	OSFSTXNO	OSFSTXTN	PO	PS	R0	R1
	R10	R11	R12	R14	R15	R2	R3	R4	R5	R6	R7	R8	R9
	UND	VAR											

Module	External References (Labels and Modules)												
DMSRRV	AERASE	AFINIS	ASTATE	ASYSREF	AWRBUF	BGCOM	DOSDD	DOSDEV	DOSDSK	DOSCP	DOSOSFST	DOSSECT	LUBPT
	NUCON	OSFST	OSFSTDSK	OSFSTXTN	PUBPT	R0	R1	R10	R11	R12	R14	R15	R2
	R3	R4	R5	R6	R7	R8	R9						
DMSSAB	APGMSECT	CURRSAVE	DEBDCBAD	FCBDD	FCBFIRST	FCBSECT	LINKLAST	LOC	NUCON	PGMOPSW	PGMSECT	RETRYBIT	R0
	R1	R10	R11	R12	R13	R14	R15	R2	R3	R4	R5	R6	R7
	R8	R9	SCBPTR	SCBSAV12	SCBWORK	STAEBIT	STAIBIT						
DMSSBD	DA	DATAEND	DECAREA	DECKYADR	DECLNGTH	DECRECPT	DECSDECB	DECTYPE	DMSSBS	DMSSBSRT	FCBBYTE	FCBITEM	FCBKEYS
	FCBOP	FCBSECT	FCBXTENT	IHADECB	IOBIN	IOBIOFLG	KEYCHNG	KEYCOUT	KEYLNGTH	KEYNAME	KEYOP	KEYSECT	KEYTBLAD
	KEYTBLNO	OPSECT	PS	R0	R1	R10	R11	R12	R14	R15	R2	R3	R4
	R5	R6	R7	R8	R9	TBLLNGTH	VAR						
DMSSBS	AOPSECT	DA	DECAREA	DECDCBAD	DECIOBPT	DECLNGTH	DECSDECB	DECTYPE	DMSSBD	DMSSEB	FCBBUFF	FCBBYTE	FCBCATML
	FCBCOUT	FCBDEV	FCBDSMD	FCBDSNAM	FCBINIT	FCBITEM	FCBMODE	FCBOP	FCBOS	FCBPDS	FCBREAD	FCBSECT	FCBTAP
	FCBXTENT	IHADEB	IHADECB	IOBBCSW	IOBBECBP	IOBBFLG	IOBIN	IOBIOFLG	IOBOUT	NUCON	OPSECT	OSIOTYPE	PO
	PREVIOUS	PS	R0	R1	R11	R12	R13	R14	R15	R2	R3	R4	R5
	R6	R8	TAPEDEV	TAPELIST	TAPEMASK	TAPEOPER	UND	VAR					
DMSSCN	BALRSAVE	CMNDLIST	NUCON	R0	R1	R12	R14	R15	R2	R3	R4	R5	R6
	R7	R8											
DMSSCR	BUFFLOC	DECLTH	DMSGIO	EDCB	EDMSK	FLAG	FLAGLOC	FLAG2	FMODE	FNAME	FTYPE	FV	GIOPLIST
	HOLDFLAG	ITEM	LINELOC	NUMLOC	PTR1	PTR2	R0	R1	R11	R12	R13	R14	R15
	R2	R3	R4	R5	R6	R7	R9	SAVCNT	SAVEAR	SCLNO	SCRBUFAD	SCRFLGS	SCRFLG2
	TABLIN	TRUNCOL	TWITCH	UTILFLAG	VERCOL1	VERLEN							
DMSSCT	ADMSROS	AOPSECT	CMSOP	DA	DECDCBAD	DECIOBPT	DECSDECB	FCBCATML	FCBCLOSE	FCBCOUT	FCBDEV	FCBDSNAM	FCBINIT
	FCBIOSW	FCBITEM	FCBOP	FCBOS	FCBOSFST	FCBPDS	FCBR13	FCBSECT	FCBTAP	FILENAME	IHADEB	IHADECB	IOBBFLG
	IOBCSW	IOBIOFLG	IOBOUT	MACDIRC	MACLIBL	NUCON	OPSECT	PS	R0	R1	R11	R12	R13
	R14	R15	R2	R3	R4	R5	R6	R7	R8	R9	SAVER14		
DMSSEB	ADMSROS	AOPSECT	BLK	CMNDLINE	CONRDCNT	CONRDCOD	CONREAD	CONWRBUF	CONWRCNT	CONWRCOD	CONWRITE	FCBBUFF	FCBBYTE
	FCBCASE	FCBCOUT	FCBDEV	FCBDSMD	FCBDSTYP	FCBINIT	FCBIO	FCBIOSW	FCBITEM	FCBMODE	FCBOP	FCBOPCB	FCBOS
	FCBPROC	FCBPRPU	FCBREAD	FCBRECL	FCBR13	FCBSECT	FCBTAPID	FXD	IHADECB	IOBBCSW	IOBBECBC	IOBBECBP	IOBIN
	IOBIOFLG	NUCON	OPSECT	PRINTLST	PS	PUNCHLST	RDBUFF	RDCCW	RDCOUNT	READLST	R0	R1	R11
	R13	R14	R15	R2	R3	R8	SAVER14	TAPEBUFF	TAPECOUT	TAPEDEV	TAPELIST	TAPEMASK	TAPEOPER
	TAPESIZE	TSOATCNL	TSOFLAGS	UND	VAR								
DMSSEG	DMSEDC	DMSEDI	DMSEXT	DMSGIO	DMSLGT	DMSLIB	DMSLSB	DMSLSY	DMSOLD	DMSSAB	DMSSBD	DMSSBS	DMSSCR
	DMSSCT	DMSSEB	DMSSLN	DMSSMN	DMSSCP	DMSSQS	DMSSVN	DMSSVT					

Module	External References (Labels and Modules)												
DMSSET	ABATABND	ADEVTAB	ADMSFRT	ADTDTA	ADTFDOS	ADTFLG2	ADTM	ADTSECT					
	ASTATE	ASYSREF	BATDCMS	BATFLAGS	BATFLAG2	BATNOEX	BATRUN	BGCOM	CMSDOS	CMSSEG	CMSVSAM	CPULOG	EXTSECT
	FRDSECT	FREELOWE	FREELOW1	JCSW3	JCSW4	LOCCNT	LTK	LUBPT	MAINHIGH	MISFLAGS	MSGFLAGS	NOABBREV	NOIMPCP
	NOIMPEX	NOPAGREL	NORDYMSG	NORDYTIM	NOVMREAD	NUCKEY	NUCON	NUM	OPTFLAGS	PIBPT	PPEND	PRFPOFF	PROTFLAG
	PUBPT	REDERRID	R0	R1	R10	R11	R12	R14	R15	R2	R3	R4	R5
	R6	R7	R8	R9	SOB1	SYSCODE	SYSNAMES	SYSREF	TIMCCW	TIMCHAR	TIMER	TIMINIT	TSOBLKS
	UPSI	UPTMID	UPTSWS	USERCODE	USERKEY	VMSIZE							
DMSSLN	ADTRANS	AFINIS	AFVS	ALDRTBLS	ALIASENT	APGMSECT	ARDBUF	ASTATE	ASVCSECT	AUSRAREA	COMPSWT	CURRSAVE	DMSOLD
	DMSSMNSB	DUMCOM	DYLD	DYLIBO	DYMBRNM	DYNAEND	FREELOWE	FRSTLOC	FVSECT	F65535	LASTLMOD	LASTTMOD	LDRFLAGS
	LINKLAST	LINKSTRT	LOCCNT	MODLIST	NUCON	OSRESET	OSSFLAGS	PGMSECT	PRFTSYS	PRFUSYS	PROTFLAG	SCBPTR	STRTADDR
	SUBACT	SUBFLAG	SVCSECT	TBENT									
DMSSMN	ATSOCPPL	AUSRAREA	BALRSAVE	BGCOM	CODE203	COMPSWT	CURRSAVE	EGPR1	EGPR15	EOCADR	FREELOWE	LOCCNT	MAINHIGH
	MAINLIST	MAINSTRT	MISFLAGS	NUCON	OSSFLAGS	OSSMNU	PPEND	RELPAGES	R0	R1	R10	R12	R13
	R14	R15	R2	R3	R4	R6	R7	R8	R9	SSAVE			
DMSSOP	AACTLKP	ACMSCVT	ADTFLG1	ADTFRO	ADTNACW	ADTSECT	AERASE	AFINIS	AFTADT	AFTFST	AFTIC	AFTIN	AFTPFST
	AFTSECT	AOPSECT	AOSRET	ASTATE	AUPDISK	BLK	CMSCVT	CMSNAME	CMSOP	CURRSAVE	DA	DEBDCBAD	DEBDEBID
	DEBOFLGS	DEBOPATB	DMSSBS	DMSSCTCE	DMSSCTCK	DMSSCTNP	DMSSQSGT	DMSSQSPT	DMSSQSUP	FCBBLKSZ	FCBBUFF	FCBBYTE	FCBCASE
	FCBCATML	FCBCLEAV	FCBCLOSE	FCBCON	FCBCCUT	FCBDCBCT	FCBDD	FCBDEV	FCBDSK	FCBDSMD	FCBDSNAM	FCBDSTYP	FCBFIRST
	FCBFORM	FCBINIT	FCBIOSW	FCBIOSW2	FCBITEM	FCBKEYS	FCBLRECL	FCBMEMBR	FCBMODE	FCBMVPDS	FCBOP	FCBOS	FCBOSFST
	FCBPDS	FCBPROC	FCBPROCC	FCBPROCO	FCBRDR	FCBRECFM	FCBRECL	FCBSECT	FILEBYTE	FILEMODE	FILENAME	FILEREAD	FILETYPE
	FXD	F6	IHADEB	IOBDCBPT	IOBEND	IOBIOFLG	IOBNXTAD	IOBSTART	JFCBIND2	JFCBMASK	JFCDSORG	JFCKEYLE	JFCLIMCT
	JFCOPTCD	LOC	MACDIRC	MACLIBL	NUCCN	OPSECT	OSFST	OSFSTBLK	OSFSTCHR	OSFSTLRL	OSFSTRFM	OSIOTYPE	PLIST
	PO	PREVIOUS	PS	QS	R0	R1	R10	R11	R12	R13	R14	R15	R2
	R3	R4	R5	R6	R7	R8	R9	SAVER1	SAVER15	TAPEDEV	TAPELIST	TAPEMASK	TAPEOPER
	UND	VAR											
DMSSQS	AOPSECT	BLK	DEBTCBAD	DMSSCTCE	DMSSCTCK	DMSSEB	FCBBUFF	FCBBYTE	FCBCLOSE	FCBCOUT	FCBDEV	FCBDSMD	FCBINIT
	FCBIORD	FCBIOSW	FCBIOWR	FCBITEM	FCBOP	FCBPVMB	FCBREAD	FCBSECT	FXD	IHADEB	IOBECB	IOBECBPT	IOBIN
	IOBIOFLG	IOBOUT	IOBSTART	IOBUPD	LOC	NUCON	OPSECT	OSIOTYPE	PREVIOUS	PS	R0	R1	R10
	R11	R12	R13	R14	R15	R2	R3	R4	R5	R6	R7	UND	VAR
DMSSRT	ASCANO	ASTRINIT	FREELOWE	MAINHIGH	MISFLAGS	NUCON	RELPAGES	R0	R1	R12	R14	R15	R2
	R3	R4	R5	R6									
DMSSRV	AERASE	AFINIS	ASTATE	ASYSREF	AWRBUF	BGCOM	DOSDD	DOSDEV	DOSDSK	DOSOP	DOSOSFST	DOSSECT	LUBPT
	NUCON	OSFST	OSFSTDSK	OSFSTXTN	PUBPT	R0	R1	R10	R12	R14	R15	R2	R3
	R4	R5	R9										
DMSSSK	NUCON	R0	R1	R12	R14	R15	R2	R3	R4	R5	R6	R8	R9
	VMSIZE												

Module	External References (Labels and Modules)												
DMSSTG	AEXTSECT	ALDRTBLS	ANCHENDA	ANCHSECT	ANCHSIZ	APGMSECT	ASTATEXT	ATSOCPPL	AUSRAREA	BALRSAVE	BGCOM	CODE203	COMPSWT
	CORESIZE	CURRSAVE	DMSLGTA	DMSSMNCF	DMSSMNCN	DMSSMNRP	DMSSMNTS	DYLD	DYLIBO	DYMBRNM	EGPR12	EGPR14	EGPR15
	EOCADR	EXTSECT	FREELOWE	IJBBOX	LINKLAST	LINKSTRT	LOCCNT	MACDIRC	MACLIBL	MAINHIGH	MAINLIST	MAINSTRT	MISFLAGS
	NUCON	OLDPSW	OPTNBYTE	OSSFLAGS	OSSMNU	PDSSECT	PGMSECT	PICADDR	PPEND	RELPAGES	R0	R1	R10
	R12	R13	R14	R15	R2	R3	R4	R5	R6	R7	R8	R9	SCBPTR
	SCBWORK	SSAVE	STIMEXIT	SYSCOM	TAXEADDR	USAVEPTR							
DMSSTT	AACTLKP	ADTFLG1	ADTFLG2	ADTFRO	ADTFROS	ADTFRW	ADTM	ADTMX	ADTSECT	AFTADT	AFTFLG	AFTFST	AFTRD
	AFTSECT	AFTWRT	DMSLAD	DMSLADW	DMSLFS	DMSLFSW	FSTFAP	FSTFAR	FSTFAW	FSTFB	FSTFRO	FSTFROX	FSTFRW
	FSTFRWX	FSTM	FSTSECT	FVSFSTAD	FVSFSTDT	FVSFSTM	FVSFSTN	NUCON	OSFST	OSFSTFLG	OSFSTFM	REGSAV3	R0
	R1	R10	R12	R13	R14	R15	R2	R3	R4	R5	R6	R9	STATEFST
	STATER0												
DMSSVN	AEXTSECT	AOPSECT	CONRDBUF	CONRDCNT	CONREAD	CONSTACK	CONWRBUF	CONWRCNT	CONWRITE	CURRSAVE	EXTSECT	FCBSECT	FSTFINRD
	LOC	LSTFINRD	NUCON	NUMFINRD	NUMFNDWR	OPSECT	OSSFLAGS	PENDREAD	PENDWRIT	PS	R0	R1	R10
	R12	R13	R14	R15	R2	R3	R4	R5	R6	R8	STIMEXIT	TIMCHAR	TIMER
	TIMINIT	TSOATCNL	TSOFLAGS										
DMSSVT	ADMPEXEC	ADMSROS	AERASE	AEXTSECT	AOPSECT	APGMSECT	APIE	ARDBUF	ASTATE	ATFINIS	AUPDISK	AWRBUF	CHNGBYTE
	CMNDLINE	CMSNAME	CMSOP	CMSTAXE	CONRDCNT	CONREAD	CONWRBUF	CONWRCNT	CONWRITE	CORESIZE	CURRDATE	CURRSAVE	DATAEND
	DECSDECB	DIAGTIME	DIRNAME	DIRPTR	DMSLGT	DMSLSB	DMSSAB	DMSSBDFR	DMSSBS	DMSSCT	DMSSLN	DMSSLN3	DMSSLN42
	DMSSLN6	DMSSLN7	DMSSLN8	DMSSLN9	DMSSMN	DMSSMN10	DMSSMN4	DMSSMN5	DMSSOP	DMSSOP19	DMSSOP20	DMSSOP22	DMSSOP23
	DMSSQS	DMSSVN	DMSSVN1	DMSSVN2	DMSSVN93	DMSSVN94	DOSDD	DOSNEXT	DOSSECT	DUMPLIST	EXTSECT	FCBBUFF	FCBBYTE
	FCBCATML	FCBCOUT	FCBDD	FCBDEV	FCBDSK	FCBDSNAM	FCBDSTYP	FCBFIRST	FCBFORM	FCBINIT	FCBIOSW2	FCBITEM	FCBKEYS
	FCBMVPDS	FCBOP	FCBOS	FCBOSFST	FCBPDS	FCBSECT	FCBTAB	FCBXTENT	FILEBUFF	FILEBYTE	FILECOUT	FILEITEM	FILEMODE
	FILENAME	FILETYPE	IHADEB	IHADECB	IHAJFCB	IOBIN	IOBIOFLG	JFCBMASK	JFCLRECL	KEYCHNG	KEYCOUT	KEYFORM	KEYLNGTH
	KEYNAME	KEYOP	KEYSECT	KEYTABLE	KEYTBLAD	KEYTBLNO	KEYTYPE	LINKSTRT	LOC	LOWSAVE	MACDIRC	MACLIBL	NEWBLKS
	NUCON	OPSECT	OSIOTYPE	OSRESET	OSSFLAGS	PDSBLKSI	PDSDIR	PDSSECT	PGMSECT	PLIST	PREVIOUS	PS	R0
	R1	R10	R11	R12	R13	R14	R15	R2	R3	R4	R5	R6	R7
	R8	R9	SCBPTR	STIMEXIT	TAXEADDR	TAXEDEF	TAXEEXIT	TAXELNK	TBLLNGTH	TEMPBYTE	TIMER	VAR	VMSIZE
	WAITLIST												
DMSSYN	AFINIS	ARDBUF	ASTATE	AUSABRV	NOSTDSYN	NUCON	OPTFLAGS	R0	R1	R11	R12	R14	R15
	R2	R3	R4	R5	R6	R7	R8						
DMSTIO	ADEVTAB	ATABEND	CC	CSW	DEVADDR	DEVMISC	DEVNAME	DEVSECT	DEVSIZE	NUCON	R0	R1	R11
	R12	R13	R14	R15	SILI								
DMSTMA	DMSLIB	R0	R1	R10	R11	R12	R14	R15	R2	R3	R4	R5	R6
	R7	R8	R9										
DMSTPD	CSW	NUCON	R0	R1	R10	R11	R12	R14	R15	R2	R3	R4	R5
	R6	R7	R8	R9									

Module	External References (Labels and Modules)												
DMSTPE	AACTLKP	ADEVTAB	AERASE	AFINIS	AFTFST	AFTSECT	AFVS	ASTATE	ATABEND	ATYPSRCH	AUPDISK	AWRBUF	DEVADDR
	DEVMISC	DEVNAME	DEVSECT	DEVSIZE	FSTD	FSTDBC	FSTFCL	FSTFV	FSTIC	FSTIL	FSTM	FSTN	FSTRP
	FSTSECT	FSTT	FSTWP	FVSECT	NUCON	R0	R1	R10	R11	R12	R13	R14	R15
	R2	R3	R4	R5	R6	R7	R8	R9	UFDBUSY	WRBIT			
DMSTQQ	ADTDTA	ADTFLG1	ADTFLG2	ADTFMFD	ADTFRW	ADTQQM	ADTSECT	AQQTRK	ATRKLKP	ATRKLKPX	DTADT	FVSECT	F65535
	NUCON	TRKLSAVE											
DMSTRK	ADTFLG1	ADTFLG2	ADTFMFD	ADTFRW	ADTMSK	ADTRES	ADTSECT	ADT1ST	R0	R1	R10	R11	R12
	R13	R14	R15	R2	R3	R4	R5	R6	R7	R8	R9		
DMSTYP	AFINIS	ARDBUF	ASTATE	MSGFLAGS	NOTYPING	NUCON	R0	R1	R10	R14	R15	R2	R3
	R4	R5	R6	R7	R8	R9							
DMSUPD	ADTFLG1	ADTFRO	ADTFRW	ADTM	ADTMX	ADTSECT	AERASE	AFINIS	ARDBUF	ASTATE	AWRBUF	FSTFV	FSTIL
	FSTM	FSTSECT	MISFLAGS	NUCON	RELPAGES	R0	R1	R10	R11	R12	R13	R14	R15
	R2	R3	R4	R5	R6	R7	R8	R9					
DMSVIB	ACMSCVT	BALRSAVE	CMSVSAM	NUCON	R0	R1	R12	R14	R15	R2	R3	R5	SYSNAMES
	VMSIZE												
DMSVIP	ACBAMBL	ACBAMO	ACBBFPL	ACBBUFND	ACBDDNM	ACBDOSID	ACBDTFID	ACBERFLG	ACBEXLST	ACBIBUF	ACBID	ACBIDD	ACBLEN
	ACBMACRF	ACBOCEXT	ACBOCTER	ACBOEMPT	ACBOFLGS	ACBOKBUF	ACBOPEN	ACBPRTCT					
	AOSRET	CURRSAVE	DOSDD	DOSDEV	DOSDSMD	DOSDUM	DOSEXTNO	DOSEXTTB	DOSNEXT	DOSSECT	DOSVOLNO	DOSVOLTB	DOSYSXXX
	EXENACTB	EXENADDR	EXLEODF	EXLEODL	EXLECDP	EXLJRN	EXLJRNL	EXLLEN	EXLLERF	EXLLERL	EXLLERP	EXLSYNF	EXLSYNL
	EXLSYNP	IKQACB	IKQEXLST	IKQRPL	NUCON	RPLACB	RPLAREA	RPLARG	RPLASY	RPLBUFL	RPLCHAIN	RPLECBPR	RPLEOFDS
	RPLFDBKC	RPLFLAG	RPLKEYL	RPLNUP	RPLCPT1	RPLOPT2	RPLRLEN	RPLRTNCD	RPLST	RPLSTRID	RPLUPD	RPLVLERR	R0
	R1	R10	R11	R12	R13	R14	R15	R2	R3	R4	R5	R6	R7
	R8	R9											
DMSVPD	R0	R1	R11	R12	R14	R15	R2	R3	R4	R5	R6	R7	
DMSVSR	ACMSCVT	BGCOM	CMSAMS	CMSCVT	CMSVSAM	NUCON	PIB2PTR	PIK	PPEND	R0	R1	R12	R13
	R14	R15	R2	R3	R4	R5	R6	R7	R8	SYSNAMES			
DMSXCP	ADTDTA	ADTFDOS	ADTFLG2	ADTFLG3	ADTFROS	ADTFRW	ADTID	ADTM	ADTSECT	AFINIS	ARDBUF	ASYSREF	AWRBUF
	BGCOM	CSW	DMSCCB	DOSBUFF	DOSBUFSP	DOSBYTE	DOSCBID	DOSCOUT	DOSDD	DOSDEV	DOSDSK	DOSDSMD	DOSDSNAM
	DOSDUM	DOSEXTCX	DOSEXTNO	DOSEXTTB	DOSINIT	DOSITEM	DOSNEXT	DOSOP	DOSOSDSN	DOSOSFST	DOSREAD	DOSSAVE	DOSSECT
	DOSSENSE	DOSTAPID	DOSUCNAM	DOSVOLNO	DOSVOLTB	DOSWORK	DOSYSXXX	LUBPT	NICLPT	NUCON	PUBADR	PUBCUU	PUBDEVT
	PUBDSKM	PUBPT	PUBTAPM1	R0	R1	R10	R11	R12	R13	R14	R15	R2	R3
	R4	R5	R6	R7	R8	R9							
DMSZAP	FSTFB	FSTFRW	FSTFV	FSTIC	FSTIL	FSTM	FSTSECT	IS	LOC	NUCON	R0	R1	R10
	R11	R12	R13	R14	R15	R2	R3	R4	R5	R6	R7	R8	R9

Label	Count	References									
AACTFREE	000004	DMSBRD	DMSBWR	DMSPNT							
AACTFRET	000004	DMSBWR	DMSERS	DMSFNS							
AACTLKP	000011	DMSBRD	DMSBWR	DMSERS	DMSFNS	DMSINT	DMSPNT	DMSRNM	DMSSOP	DMSSTT	DMSTPE
AACTNXT	000001	DMSERS									
AADTLKP	000001	DMSDLK									
AADTLKW	000012	DMSARX	DMSASM	DMSBSC	DMSDLK	DMSLBM	DMSLBT	DMSLKD			
ABATABND	000011	DMSABN	DMSBTB	DMSCIO	DMSDSK	DMSERR	DMSFLD	DMSITE	DMSPIO	DMSRDC	DMSSET
ABATLIMT	000004	DMSBTB	DMSCIO	DMSITE	DMSPIO						
ABATPRCC	000003	DMSBTB	DMSCPF	DMSCRD							
ABNBIT	000003	DMSABN	DMSBTP								
ABNERLST	000001	DMSABN									
ABNPAS13	000001	DMSABN									
ABNPSW	000026	DMSABN	DMSDBG	DMSFRE	DMSITI	DMSITP	DMSITS				
ABNREGS	000013	DMSABN	DMSDBG	DMSFRE	DMSITI	DMSITP	DMSITS				
ABNRR	000002	DMSABN									
ABWSECT	000008	DMSABN	DMSDBG	DMSFRE	DMSITI	DMSITP	DMSITS				
ACALL	000001	DMSFRE									
ACBAMBL	000001	DMSVIP									
ACBAMO	000005	DMSVIP									
ACBBFPL	000001	DMSVIP									
ACBBUFND	000001	DMSVIP									
ACBDDNM	000002	DMSBOP	DMSVIP								
ACBDOSID	000001	DMSVIP									
ACBDTFID	000001	DMSVIP									
ACBERFLG	000007	DMSBOP	DMSVIP								
ACBEXLST	000004	DMSVIP									
ACBIBUF	000001	DMSVIP									

Label	Count	References
ACBID	000006	DMSVIP
ACBIDD	000007	DMSVIP
ACBIN	000001	DMSBOP
ACBINFLG	000001	DMSBOP
ACBLEN	000001	DMSVIP
ACBMACRF	000001	DMSVIP
ACBMACR1	000002	DMSBOP
ACBOCEXT	000001	DMSVIP
ACBOCTER	000001	DMSVIP
ACBOEMPT	000001	DMSVIP
ACBOFLGS	000002	DMSVIP
ACBOKBUF	000001	DMSVIP
ACBOLIGN	000001	DMSBOP
ACBOPEN	000002	DMSVIP
ACBOUT	000001	DMSBOP
ACBPRTCT	000001	DMSVIP
ACBST	000001	DMSVIP
ACBSTRNO	000001	DMSVIP
ACBSTSKP	000001	DMSBOP
ACBSTYP	000001	DMSVIP
ACBUAPTR	000001	DMSVIP
ACMSCVT	000004	DMSINS DMSSOP DMSVIB DMSVSR
ACMSRET	000004	DMSDOS DMSLDR DMSVIP
ADEVTAB	000017	DMSAMS DMSASN DMSDBD DMSEDI DMSEDX DMSFOR DMSGIO DMSINI DMSSET DMSTIO DMSTPE
ADIOSECT	000005	DMSACM DMSDIO DMSFNS DMSITI
ADMPEXEC	000001	DMSSVT
ADMSCRD	000002	DMSBTP DMSDBG
ADMSFRT	000001	DMSSET
ADMSOVS	000008	DMSOVR
ADMSPICC	000001	DMSPRT
ADMSROS	000016	DMSACM DMSALU DMSLDS DMSLFS DMSSCT DMSSEB DMSSVT
ADTADD	000008	DMSACF DMSACM DMSAUD DMSERS DMSFNS
ADTCFST	000003	DMSACF DMSERS
ADTCHBA	000016	DMSACF DMSCPY DMSERS DMSLFS DMSRNM
ADTCYL	000007	DMSACM DMSFOR DMSLDS DMSQRY DMSROS
ADTDTA	000027	DMSACC DMSACM DMSARE DMSASN DMSAUD DMSBWR DMSDIO DMSFOR DMSQRY DMSROS DMSSET DMSTQQ DMSXCP
ADTFALNM	000003	DMSACF
ADTFALTY	000004	DMSACF
ADTFALUF	000004	DMSACC DMSACF DMSFOR
ADTFDA	000025	DMSABN DMSACC DMSACF DMSALU DMSAUD DMSFOR DMSINS DMSLAD DMSLFS DMSLST
ADTFDOS	000016	DMSACC DMSASN DMSBOP DMSDLB DMSEXT DMSQRY DMSROS DMSSET DMSXCP
ADTFFSTF	000008	DMSABN DMSACC DMSACF DMSALU DMSFOR DMSINS
ADTFFSTV	000007	DMSACC DMSINS DMSLAD DMSLFS

Label	Count	References											
ADTFLG1	000101	DMSABN	DMSACC	DMSACF	DMSACM	DMSALU	DMSARE	DMSARN	DMSARX	DMSASM	DMSASN	DMSBOP	DMSBSC
		DMSBWR	DMSCPY	DMSDIO	DMSDLK	DMSDSL	DMSERS	DMSFOR	DMSINS	DMSLAD	DMSLAF	DMSLBM	DMSLDS
		DMSLFS	DMSLLU	DMSLST	DMSMVE	DMSQRY	DMSRNM	DMSROS	DMSSOP	DMSSTT	DMSTQQ	DMSTRK	DMSUPD
ADTFLG2	000063	DMSABN	DMSACC	DMSACF	DMSACM	DMSALU	DMSARE	DMSASN	DMSBOP	DMSDLB	DMSEXT	DMSFOR	DMSLAD
		DMSLDS	DMSLFS	DMSQRY	DMSROS	DMSSET	DMSSTT	DMSTQQ	DMSTRK	DMSXCP			
ADTFLG3	000029	DMSACC	DMSACF	DMSACM	DMSALU	DMSARE	DMSAUD	DMSBOP	DMSBWR	DMSINS	DMSLFS	DMSQRY	DMSROS
		DMSXCP											
ADTFMDRO	000003	DMSACF											
ADTFMFD	000007	DMSACM	DMSBOP	DMSEXT	DMSTQQ	DMSTRK							
ADTFMIN	000004	DMSABN	DMSACC	DMSALU									
ADTFQQF	000005	DMSABN	DMSACM	DMSALU	DMSFOR								
ADTFRO	000031	DMSACC	DMSACF	DMSACM	DMSALU	DMSARE	DMSASN	DMSBOP	DMSDIO	DMSERS	DMSFOR	DMSLAD	DMSLDS
		DMSLFS	DMSLST	DMSQRY	DMSRNM	DMSSOP	DMSSTT	DMSUPD					
ADTFROS	000031	DMSABN	DMSACC	DMSACF	DMSALU	DMSARE	DMSASN	DMSBOP	DMSDLB	DMSEXT	DMSLAD	DMSLDS	DMSLFS
		DMSQRY	DMSROS	DMSSTT	DMSXCP								
ADTFRW	000069	DMSACC	DMSACF	DMSACM	DMSALU	DMSARE	DMSARN	DMSARX	DMSASM	DMSASN	DMSBOP	DMSBSC	DMSBWR
		DMSCPY	DMSDIO	DMSDLK	DMSDSL	DMSERS	DMSFOR	DMSLAD	DMSLAF	DMSLBM	DMSLDS	DMSLFS	DMSLLU
		DMSLST	DMSMVE	DMSQRY	DMSRNM	DMSSTT	DMSTQQ	DMSTRK	DMSUPD	DMSXCP			
ADTFSTC	000013	DMSACC	DMSACF	DMSALU	DMSARE	DMSBWR	DMSERS	DMSINS	DMSQRY				
ADTFTYP	000012	DMSACF	DMSALU	DMSDSK	DMSLFS	DMSRNM							
ADTFVS	000001	DMSLAD											
ADTHBCT	000016	DMSABN	DMSACC	DMSACF	DMSACM	DMSAUD	DMSERS	DMSFOR	DMSLAD	DMSLFS			
ADTID	000011	DMSACM	DMSALU	DMSFOR	DMSLDS	DMSLST	DMSPUN	DMSQRY	DMSXCP				
ADTLAST	000006	DMSAUD	DMSFOR										
ADTLEFT	000003	DMSFOR	DMSLAD										
ADTLFST	000002	DMSERS	DMSLFS										
ADTLHBA	000007	DMSACC	DMSACF	DMSERS	DMSFOR	DMSLFS							
ADTM	000079	DMSABN	DMSACC	DMSACF	DMSALU	DMSAMS	DMSARE	DMSARN	DMSARX	DMSASM	DMSBSC	DMSCMP	DMSCPY
		DMSDLK	DMSDSL	DMSERS	DMSEXC	DMSEXT	DMSFOR	DMSLAD	DMSLAF	DMSLBM	DMSLDS	DMSLFS	DMSLKD
		DMSLST	DMSQRY	DMSRNM	DMSROS	DMSSET	DMSSTT	DMSUPD	DMSXCP				
ADTMFDA	000004	DMSABN	DMSACF	DMSAUD									
ADTMFDN	000014	DMSABN	DMSACC	DMSACF	DMSACM	DMSALU	DMSAUD						
ADTMSK	000011	DMSACC	DMSACM	DMSALU	DMSAUD	DMSFOR	DMSTRK						
ADTMX	000032	DMSACC	DMSACM	DMSALU	DMSARN	DMSARX	DMSASM	DMSBSC	DMSBWR	DMSLAF	DMSLFS	DMSQRY	DMSSTT
		DMSUPD											
ADTNACW	000008	DMSBWR	DMSSOP										
ADTNUM	000012	DMSACC	DMSACM	DMSAUD	DMSFOR	DMSQRY							
ADTPQM1	000010	DMSACM	DMSALU	DMSAUD	DMSFOR								
ADTPQM2	000009	DMSACC	DMSACF	DMSACM	DMSAUD	DMSFOR							
ADTPQM3	000006	DMSABN	DMSACC	DMSACM	DMSALU	DMSFOR							
ADTPSTM	000004	DMSLAD											
ADTPTR	000002	DMSLAD											
ADTQQM	000006	DMSACM	DMSALU	DMSFOR	DMSTQQ								
ADTRANS	000011	DMSSLN											

Label	Count	References											
ADTRES	000014	DMSACC	DMSACF	DMSACM	DMSALU	DMSERS	DMSFOR	DMSLAD	DMSLFS	DMSTRK			
ADTROX	000003	DMSACM	DMSALU										
ADTSECT	000106	DMSABN	DMSACC	DMSACF	DMSACM	DMSALU	DMSAMS	DMSARE	DMSARN	DMSARX	DMSASM	DMSASN	DMSAUD
		DMSBOP	DMSBSC	DMSBWR	DMSCMP	DMSCPY	DMSDIO	DMSDLB	DMSDLK	DMSDSK	DMSDSL	DMSERS	DMSEXC
		DMSEXT	DMSFOR	DMSINS	DMSLAD	DMSLAF	DMSLBM	DMSLDS	DMSLFS	DMSLKD	DMSLLU	DMSLST	DMSMVE
		DMSPUN	DMSQRY	DMSRNM	DMSROS	DMSSET	DMSSOP	DMSSTT	DMSTQQ	DMSTRK	DMSUPD	DMSXCP	
ADTUSED	000010	DMSACC	DMSACM	DMSFOR									
ADT1ST	000007	DMSACC	DMSFOR	DMSTRK									
AEDLIN	000001	DMSEDX											
AERASE	000036	DMSAMS	DMSBOP	DMSCLS	DMSDLK	DMSDSK	DMSDSL	DMSEDI	DMSFNS	DMSLDR	DMSLIO	DMSLLU	DMSLST
		DMSMOD	DMSOLD	DMSPRV	DMSRDC	DMSRNE	DMSRRV	DMSSOP	DMSSRV	DMSSVT	DMSTPE	DMSUPD	
AERR	000001	DMSITS											
AEXEC	000002	DMSEXC											
AEXTEND	000007	DMSEDI	DMSEDX										
AEXTSECT	000014	DMSINS	DMSINT	DMSIOW	DMSITE	DMSQRY	DMSSET	DMSSTG	DMSSVN	DMSSVT			
AFINIS	000042	DMSACC	DMSARE	DMSCLS	DMSCMP	DMSDLK	DMSDSK	DMSEDI	DMSEDX	DMSEXC	DMSEXT	DMSLDR	DMSLIB
		DMSLIO	DMSLLU	DMSMOD	DMSCLD	DMSPRT	DMSPRV	DMSPUN	DMSRDC	DMSRNE	DMSRRV	DMSSLN	DMSSOP
		DMSSRV	DMSSYN	DMSTPE	DMSTYP	DMSUPD	DMSXCP						
AFLAGLOC	000001	DMSEDX											
AFREETAB	000006	DMSFRE	DMSSET										
AFSTFNRD	000004	DMSEDI	DMSEDX										
AFTADT	000023	DMSBRD	DMSBWR	DMSERS	DMSLAF	DMSRNM	DMSSOP	DMSSTT					
AFTCLA	000011	DMSBRD	DMSBWR										
AFTCLB	000010	DMSBRD	DMSBWR										
AFTCLD	000015	DMSBRD	DMSBWR										
AFTCLDX	000005	DMSBWR											
AFTCLN	000014	DMSBRD	DMSBWR										
AFTCLX	000006	DMSBWR											
AFTD	000002	DMSBWR											
AFTDBA	000019	DMSBRD	DMSBWR										
AFTDBC	000005	DMSBWR	DMSERS										
AFTDBD	000007	DMSBRD	DMSBWR										
AFTDBN	000009	DMSBRD	DMSBWR										
AFTFB	000001	DMSLAF											
AFTFBA	000005	DMSBRD	DMSBWR										
AFTFCL	000011	DMSBRD	DMSBWR	DMSERS									
AFTFCLA	000007	DMSBRD	DMSBWR										
AFTFCLX	000008	DMSBWR											
AFTFLG	000036	DMSBRD	DMSBWR	DMSERS	DMSLAF	DMSSTT							
AFTFLG2	000015	DMSBWR											
AFTFSF	000002	DMSLAF											
AFTFST	000009	DMSBRD	DMSBWR	DMSLAF	DMSSOP	DMSSTT	DMSTPE						
AFTFULD	000002	DMSBWR											
AFTFV	000006	DMSBRD	DMSBWR										

Label	Count	References											
AFTIC	000008	DMSBRD	DMSBWR	DMSPNT	DMSSOP								
AFTID	000010	DMSBRD	DMSBWR										
AFTIL	000005	DMSBRD	DMSBWR										
AFTIN	000014	DMSBRD	DMSBWR	DMSSOP									
AFTLD	000002	DMSLAF											
AFTM	000008	DMSBWR	DMSINT	DMSLAF									
AFTN	000005	DMSBWR	DMSINT	DMSLAF									
AFTNEW	000004	DMSBWR											
AFTOLDCL	000006	DMSBWR											
AFTPFST	000007	DMSERS	DMSLAF	DMSSOP									
AFTPTR	000012	DMSLAF											
AFTRD	000006	DMSBRD	DMSBWR	DMSSTT									
AFTRP	000008	DMSBRD	DMSBWR	DMSPNT									
AFTSECT	000024	DMSBRD	DMSBWR	DMSERS	DMSINT	DMSLAF	DMSPNT	DMSRNM	DMSSOP	DMSSTT	DMSTPE		
AFTT	000001	DMSLAF											
AFTUSED	000004	DMSLAF											
AFTWP	000010	DMSBWR	DMSINT	DMSPNT									
AFTWRT	000008	DMSBRD	DMSBWR	DMSSTT									
AFVS	000042	DMSABN	DMSACC	DMSACF	DMSACM	DMSALU	DMSAUD	DMSBTB	DMSBTP	DMSBWR	DMSCIT	DMSCRD	DMSCWR
		DMSCWT	DMSDIO	DMSDSK	DMSERS	DMSEXC	DMSFNS	DMSINT	DMSITI	DMSITP	DMSITS	DMSMOD	DMSQRY
		DMSRNM	DMSSLN	DMSTPE									
AGETCLK	000001	DMSEXT											
AINCORE	000005	DMSEDI											
AINTRTBL	000007	DMSABN	DMSCRD	DMSQRY	DMSSET								
AIOSECT	000008	DMSABN	DMSCIT	DMSDBG	DMSHDI	DMSINT	DMSITI						
AKILLEX	000001	DMSDBG											
ALCHAR1	000002	DMSEDI											
ALCHAR2	000002	DMSEDI											
ALDRTBLS	000022	DMSBTB	DMSFET	DMSGND	DMSINS	DMSLDR	DMSLOA	DMSMDP	DMSMOD	DMSOLD	DMSQRY	DMSSET	DMSSLN
		DMSSTG											
ALIASENT	000004	DMSLIO	DMSSLN										
ALINELCC	000001	DMSEDX											
ALTLIST	000006	DMSEDI											
ALTMODE	000006	DMSEDX											
ANCHENDA	000003	DMSDOS	DMSSTG										
ANCHENTP	000001	DMSDOS											
ANCHINST	000001	DMSDOS											
ANCHLDPT	000002	DMSDOS											
ANCHLENG	000002	DMSDOS											
ANCHPHLN	000001	DMSDOS											
ANCHPHNM	000005	DMSDOS											
ANCHSECT	000003	DMSDOS	DMSSTG										
ANCHSIZ	000003	DMSSTG											
ANCHSTSW	000001	DMSDOS											

Label	Count	References											
ANUCEND	000003	DMSDIO	DMSHDI	DMSHDS									
ANUMLOC	000001	DMSEDX											
AOPSECT	000026	DMSABN	DMSARN	DMSCRD	DMSCWR	DMSCWT	DMSDBG	DMSEXC	DMSEXT	DMSINS	DMSINT	DMSSBS	DMSSCT
		DMSSEB	DMSSOP	DMSSQS	DMSSVN	DMSSVT							
AOSRET	000003	DMSDOS	DMSSOP	DMSVIP									
AOUTRTBL	000006	DMSABN	DMSCWR	DMSQRY	DMSSET								
APGMSECT	000007	DMSITP	DMSSAB	DMSSLN	DMSSTG	DMSSVT							
APIE	000001	DMSSVT											
APOINT	000002	DMSEXT	DMSLIB										
APRILB	000006	DMSLDR	DMSOLD										
APSV	000035	DMSLDR	DMSLGT	DMSLIB	DMSLIC	DMSLSB	DMSOLD						
AQQTRK	000004	DMSBWR	DMSTQQ										
AQQTRKX	000005	DMSBWR	DMSERS	DMSFNS									
ARDBUF	000040	DMSCMP	DMSDLK	DMSDSK	DMSEDI	DMSEDX	DMSEXT	DMSLDR	DMSLGT	DMSMOD	DMSOLD	DMSPRT	DMSPUN
		DMSRNE	DMSSLN	DMSSVT	DMSSYN	DMSTYP	DMSUPD	DMSXCP					
ARDTK	000011	DMSACF	DMSACM	DMSBRD	DMSBWR	DMSERS	DMSFNS	DMSFOR	DMSMOD				
AREA	000027	DMSEDI											
ARFLG	000002	DMSDOS											
ARGMAX	000001	DMSDBG											
ARGS	000038	DMSDBD	DMSDBG										
ARGSAV	000008	DMSDBG											
ARGSCT	000016	DMSDBG											
ASCANN	000005	DMSAMS	DMSBTP	DMSLDR	DMSCLD	DMSRDC							
ASCANO	000002	DMSEXT	DMSSRT										
ASCBPTR	000002	DMSINT											
ASSTAT	000002	DMSFRE	DMSINS										
ASTATE	000034	DMSAMS	DMSBOP	DMSDLK	DMSDSK	DMSDSL	DMSEDX	DMSEXT	DMSFCH	DMSFLD	DMSGLB	DMSGND	DMSINS
		DMSLDR	DMSLIB	DMSMDP	DMSMOD	DMSOLD	DMSPRT	DMSPUN	DMSRRV	DMSSET	DMSSLN	DMSSOP	DMSSRV
		DMSSVT	DMSSYN	DMSTPE	DMSTYP	DMSUPD							
ASTATEW	000007	DMSAMS	DMSEDX	DMSERS	DMSMOD	DMSRDC	DMSRNM						
ASTATEXT	000002	DMSINS	DMSSTG										
ASTRINIT	000003	DMSARN	DMSBSC	DMSSRT									
ASUBFST	000003	DMSABN	DMSINT										
ASUBRET	000002	DMSINT											
ASUBSECT	000006	DMSABN	DMSINM	DMSINT									
ASUBSTAT	000003	DMSABN	DMSINT										
ASVCSECT	000017	DMSCIT	DMSFRE	DMSHDS	DMSINT	DMSITS	DMSOVR	DMSOVS	DMSSLN				
ASYSREF	000025	DMSASN	DMSBOP	DMSCLS	DMSDLB	DMSDMP	DMSDOS	DMSFCH	DMSINS	DMSITP	DMSLLU	DMSOPL	DMSPRV
		DMSQRY	DMSRRV	DMSSET	DMSSRV	DMSXCP							
ATABEND	000005	DMSAMS	DMSTIO	DMSTPE									
ATFINIS	000006	DMSBWR	DMSERS	DMSRNM	DMSSVT								
ATRKLKP	000004	DMSAUD	DMSBWR	DMSTQQ									
ATRKLKPX	000014	DMSAUD	DMSBWR	DMSERS	DMSFNS	DMSTQQ							

Label	Count	References
ATSOCPPL	000002	DMSSMN DMSSTG
ATTN	000012	DMSCIT DMSEDI
ATTNLEN	000007	DMSEDI
ATYPSRCH	000005	DMSACF DMSDSK DMSFNS DMSRNM DMSTPE
AUPDISK	000016	DMSARE DMSBWR DMSDSK DMSERS DMSFNS DMSFOR DMSRNM DMSSOP DMSSVT DMSTPE
AUPIE	000002	DMSITP
AUSABRV	000004	DMSABN DMSINA DMSQRY DMSSYN
AUSRAREA	000033	DMSABN DMSBRD DMSBTB DMSFCH DMSFET DMSFRE DMSINS DMSINT DMSLDR DMSLOA DMSLSB DMSMOD
		DMSOLD DMSSLN DMSSMN DMSSTG
AUSRILST	000008	DMSABN DMSHDI
AUSRITBL	000007	DMSABN DMSHDI
AUTOCNT	000005	DMSEDI
AUTOCURR	000003	DMSEDI
AUTOREG	000002	DMSEDI
AWAIT	000001	DMSITS
AWRBUF	000025	DMSDLK DMSDSK DMSEDI DMSLIO DMSLLU DMSMOD DMSOLD DMSPRV DMSRDC DMSRNE DMSRRV DMSSRV
		DMSSVT DMSTPE DMSUPD DMSXCP
AWRTK	000004	DMSAUD DMSBWR DMSFNS DMSFOR
BALR	000241	DMSITE
BALRSAVE	000027	DMSCPF DMSDBG DMSFNS DMSINA DMSINM DMSSCN DMSSMN DMSSTG DMSVIB
BATCPEX	000005	DMSBTP DMSCPF
BATCPUC	000002	DMSITE
BATCPUL	000001	DMSITE
BATDCMS	000008	DMSBTB DMSBTP DMSDSK DMSFLD DMSRDC DMSSET
BATFLAGS	000057	DMSABN DMSACM DMSARN DMSBTB DMSBTP DMSCIO DMSCPF DMSCRD DMSDSK DMSERR DMSFLD DMSINS
		DMSITE DMSLDR DMSLSB DMSMVE DMSOLD DMSPIO DMSRDC DMSSET
BATFLAG2	000015	DMSBTB DMSBTP DMSCIT DMSDSK DMSFLD DMSINS DMSITE DMSRDC DMSSET
BATIPLSS	000001	DMSINS
BATLOAD	000013	DMSABN DMSACM DMSBTB DMSCPF DMSCRD DMSINS DMSITE DMSLDR DMSLSB DMSOLD
BATLSECT	000003	DMSCIO DMSITE DMSPIO
BATMOVE	000006	DMSBTP DMSMVE
BATNOEX	000010	DMSBTB DMSBTP DMSCIO DMSPIO DMSSET
BATPRTC	000002	DMSPIO
BATPRTL	000001	DMSPIO
BATPUNC	000002	DMSCIO
BATPUNL	000001	DMSCIO
BATRERR	000003	DMSBTP
BATRUN	000021	DMSABN DMSARN DMSBTB DMSCIO DMSCPF DMSCRD DMSDSK DMSERR DMSFLD DMSINS DMSITE DMSPIO
		DMSRDC DMSSET
BATSTOP	000002	DMSBTP DMSCIT
BATTERM	000005	DMSBTP
BATUSEX	000004	DMSBTB DMSBTP DMSCPF
BATXCPU	000002	DMSBTP DMSITE

Label	Count	References											
BATXLIM	000005	DMSBTP	DMSCIO	DMSITE	DMSPIO								
BATXPRT	000002	DMSBTP	DMSPIO										
BATXPUN	000001	DMSCIO											
BEGAT	000003	DMSDBG											
BGCOM	000048	DMSAMS	DMSASN	DMSBAB	DMSBOP	DMSCLS	DMSDLB	DMSDLK	DMSDMP	DMSDOS	DMSDSV	DMSFCH	DMSFET
		DMSINS	DMSITP	DMSLLU	DMSOPL	DMSOPT	DMSPRV	DMSRRV	DMSSET	DMSSMN	DMSSRV	DMSSTG	DMSVSR
		DMSXCP											
BITS	000009	DMSDBG											
BLANK1	000001	DMSEDX											
BLANK2	000002	DMSEDX											
BLANK3	000001	DMSEDX											
BLK	000015	DMSSEB	DMSSOP	DMSSQS									
BLOC	000006	DMSEDI	DMSEDX										
BLOCKLEN	000010	DMSFRE											
BRAD	000021	DMSLDR	DMSLSB	DMSOLD									
BRKPNTBL	000003	DMSDBG											
BUFAD	000010	DMSGIO											
BUFFLOC	000001	DMSSCR											
BYTE	000004	DMSEDI											
CALLEE	000018	DMSERR	DMSITP	DMSITS	DMSLDR								
CALLER	000004	DMSFRE	DMSITS										
CANCCW	000002	DMSEDX	DMSGIO										
CARDINCR	000003	DMSEDI	DMSEDX										
CARDNO	000003	DMSEDI											
CASEREAD	000001	DMSEDI											
CASESW	000006	DMSEDI	DMSEDX										
CAW	000015	DMSCIO	DMSCIT	DMSDBD	DMSDBG	DMSDIO	DMSERR	DMSINI	DMSINS	DMSPIO			
CC	000305	DMSINI	DMSINS	DMSTIO									
CCWPRINT	000017	DMSDBD											
CCWX	000002	DMSDIO											
CCW1	000005	DMSDIO											
CCW1A	000004	DMSDIO											
CCW2	000003	DMSDIO											
CDMSROS	000006	DMSACM	DMSALU										
CE	000004	DMSCIT	DMSINI										
CHANO	000002	DMSINI	DMSINS										
CHKWRD1	000002	DMSITS											
CHKWRD2	000002	DMSITS											
CHNGBYTE	000010	DMSSVT											
CHNGCNT	000003	DMSEDI											
CHNGFLAG	000018	DMSEDI											
CHNGMSG	000003	DMSEDI	DMSEDX										
CHNGNUM	000005	DMSEDI											

Label	Count	References
CLOSELIB	000016	DMSLDR DMSLIB DMSOLD
CMDBLOK	000001	DMSGIO
CMNDLINE	000012	DMSABN DMSARX DMSASM DMSCPF DMSINS DMSINT DMSSEB DMSSVT
CMNDLIST	000024	DMSCAT DMSEXT DMSINS DMSLDR DMSOLD DMSSCN
CMODE	000017	DMSEDI
CMSAMS	000005	DMSAMS DMSVSR
CMSCVT	000003	DMSINS DMSSOP DMSVSR
CMSDOS	000002	DMSSET
CMSNAME	000002	DMSSOP DMSSVT
CMSOP	000016	DMSSCT DMSSOP DMSSVT
CMSSEG	000017	DMSBTP DMSEDX DMSEXC DMSINS DMSINT DMSITS DMSQRY DMSSET
CMSTAXE	000005	DMSCIT DMSSVT
CMSTIM	000007	DMSINT
CMSVSAM	000009	DMSBOP DMSDOS DMSSET DMSVIB DMSVSR
CODE	000016	DMSFRE DMSITS
CODE203	000012	DMSFRE DMSITS DMSSMN DMSSTG
COMMONEX	000004	DMSLDR DMSOLD
COMNAME	000013	DMSAMS DMSDLK DMSDOS DMSDSV DMSFCH DMSFET
COMPSWT	000014	DMSARN DMSARX DMSASM DMSSLN DMSSMN DMSSTG
CONCCWS	000008	DMSCIT DMSERR
CONHXT	000002	DMSDBG
CONINBLK	000004	DMSCRD
CONINBUF	000004	DMSCRD
CONRDBUF	000001	DMSSVN
CONRDCNT	000007	DMSABN DMSINS DMSINT DMSSEB DMSSVN DMSSVT
CONRDCOD	000005	DMSABN DMSINS DMSINT DMSSEB
CONREAD	000009	DMSABN DMSINS DMSINT DMSSEB DMSSVN DMSSVT
CONSOLE	000018	DMSEDI DMSEDX DMSINI
CONSTACK	000008	DMSCIT DMSCWR DMSSVN
CONWR	000005	DMSDBG
CONWRBUF	000005	DMSINT DMSSEB DMSSVN DMSSVT
CONWRCNT	000004	DMSSEB DMSSVN DMSSVT
CONWRCCD	000005	DMSINT DMSSEB
CONWRITE	000005	DMSINT DMSSEB DMSSVN DMSSVT
CONWRL	000001	DMSDBG
CORESIZE	000009	DMSSTG DMSSVT
CORITEM	000007	DMSEDI DMSEDX
COUNT	000085	DMSEDI
CPULOG	000005	DMSDBD DMSSET
CRBIT	000002	DMSEDI
CRDPTR	000006	DMSLDR DMSOLD
CSW	000054	DMSCIO DMSCIT DMSCRD DMSCWR DMSDBG DMSDIO DMSDLK DMSFCH DMSGIO DMSINI DMSIOW DMSITE DMSITI DMSLDS DMSPIO DMSROS DMSTIO DMSTPD DMSXCP

Label	Count	References
CTL	000004	DMSGIO
CURRCPUT	000001	DMSINM
CURRDATE	000005	DMSEXT DMSINM DMSINS DMSSVT
CURRICCP	000003	DMSCIT
CURRSAVE	000061	DMSABN DMSACC DMSBSC DMSDBG DMSDLB DMSDOS DMSERR DMSFLD DMSFRE DMSITP DMSITS DMSLDR DMSOVS DMSSAB DMSSLN DMSSMN DMSSOP DMSSTG DMSSVN DMSSVT DMSVIP
CURRTIME	000001	DMSEXT
CURRVIRT	000002	DMSINM
CVTMDL	000001	DMSINS
CVTMZ00	000001	DMSINS
CVTNUCB	000001	DMSINS
CVTOPTA	000001	DMSINS
CVTSECT	000001	DMSINS
DA	000020	DMSDSL DMSMVE DMSSBD DMSSBS DMSSCT DMSSOP
DACTIVE	000007	DMSDOS DMSFET
DATAEND	000015	DMSSBD DMSSVT
DBDDMSG	000003	DMSDBD
DBDEXIT	000003	DMSDBD
DBGABN	000005	DMSABN DMSDBG
DBGEXEC	000003	DMSCIT DMSDBG DMSITE
DBGEXINT	000006	DMSCIT DMSDBG DMSITE
DBGFLAGS	000036	DMSABN DMSCIT DMSDBD DMSDBG DMSITE
DBGOUT	000029	DMSDBD DMSDBG
DBGPGMCK	000004	DMSDBG
DBGRECUR	000017	DMSDBD DMSDBG
DBGSAV1	000002	DMSDBG
DBGSAV2	000001	DMSDBG
DBGSECT	000007	DMSDBD DMSDBG DMSITE
DBGSET	000003	DMSDBG
DBGSWTCH	000012	DMSDBD DMSDBG
DDNAM	000001	DMSMVE
DE	000006	DMSCIT DMSINI
DEBDCBAD	000002	DMSSAB DMSSOP
DEBDEBID	000001	DMSSOP
DEBOFLGS	000001	DMSSOP
DEBOPATB	000001	DMSSOP
DEBTCBAD	000004	DMSSQS
DEC	000068	DMSDBD DMSDBG
DECAREA	000007	DMSSBD DMSSBS
DECDCBAD	000002	DMSSBS DMSSCT
DECDEC	000031	DMSDBG
DECIMAL	000009	DMSEDI
DECIOBPT	000003	DMSSBS DMSSCT

Label	Count	References					
DECKYADR	000004	DMSSBD					
DECLNGTH	000004	DMSSBD	DMSSBS				
DECLTH	000002	DMSSCR					
DECRECPT	000002	DMSSBD					
DECSDECB	000021	DMSSBD	DMSSBS	DMSSCT	DMSSVT		
DECTYPE	000025	DMSSBD	DMSSBS				
DEVADDR	000041	DMSTIO	DMSTPE				
DEVICE	000004	DMSARX	DMSASM	DMSIOW	DMSITI		
DEVMISC	000005	DMSTIO	DMSTPE				
DEVNAME	000003	DMSTIO	DMSTPE				
DEVSECT	000003	DMSTIO	DMSTPE				
DEVSIZE	000003	DMSTIO	DMSTPE				
DEVTAB	000011	DMSASN	DMSDBD	DMSEDI	DMSEDX	DMSINI	
DEVTYP	000017	DMSDIO	DMSFNS				
DIAGNUM	000001	DMSDIO					
DIAGRET	000003	DMSDIO					
DIAGTIME	000001	DMSSVT					
DIOBIT	000002	DMSDIO					
DIOCSW	000001	DMSFNS					
DIOFLAG	000009	DMSDIO					
DIOFREE	000003	DMSDIO					
DIOSECT	000007	DMSACM	DMSDIO	DMSFNS	DMSITI		
DIRC	000016	DMSDOS					
DIRLL	000004	DMSDOS					
DIRN	000006	DMSDOS	DMSFET				
DIRNAME	000038	DMSDOS	DMSDSL	DMSFET	DMSSVT		
DIRPTR	000007	DMSSVT					
DIRR	000001	DMSDSL					
DIRTT	000005	DMSDOS	DMSDSL				
DISK$SEG	000006	DMSBRD	DMSFNS	DMSLFS			
DITCNT	000005	DMSEDI					
DMPTITLE	000003	DMSDBG					
DMSABNGO	000005	DMSFRE	DMSITI	DMSITP	DMSITS		
DMSABNRT	000001	DMSDBG					
DMSABW	000011	DMSABN	DMSDBG	DMSFRE	DMSITI	DMSITP	DMSITS
DMSARD	000001	DMSARX					
DMSASD	000001	DMSASM					
DMSBWR	000001	DMSFNC					
DMSCAT	000003	DMSABN	DMSCRD	DMSFNC			
DMSCCB	000002	DMSXCP					
DMSCIOSI	000001	DMSFNC					
DMSCITA	000001	DMSCWR					
DMSCITB	000002	DMSCRD	DMSCWR				

Label	Count	References				
DMSCITDB	000002	DMSABN	DMSFNC			
DMSCPF	000002	DMSFNC	DMSINT			
DMSCRD	000003	DMSABN	DMSFNC			
DMSCWR	000004	DMSDBG	DMSERR	DMSFNC	DMSITE	
DMSCWT	000005	DMSABN	DMSDBG	DMSERR	DMSFNC	DMSITS
DMSDBD	000001	DMSDBG				
DMSDBG	000002	DMSABN	DMSFNC			
DMSDBGP	000001	DMSINI				
DMSEDC	000001	DMSSEG				
DMSEDI	000001	DMSSEG				
DMSERR	000002	DMSFNC	DMSITS			
DMSERT	000002	DMSERR				
DMSEXC	000001	DMSFNC				
DMSEXCAB	000001	DMSABN				
DMSEXT	000001	DMSSEG				
DMSFCH	000003	DMSDOS				
DMSFET	000001	DMSFNC				
DMSFNC	000001	DMSITS				
DMSFNC3	000001	DMSITS				
DMSFREES	000001	DMSFNC				
DMSFREEX	000001	DMSFNC				
DMSFRETS	000001	DMSFNC				
DMSFRT	000002	DMSFRE				
DMSGIO	000002	DMSSCR	DMSSEG			
DMSINALT	000001	DMSNUC				
DMSINA1S	000001	DMSNUC				
DMSINS	000001	DMSINI				
DMSINSE	000001	DMSINI				
DMSINTAB	000001	DMSABN				
DMSIOWR	000001	DMSDBG				
DMSITET	000001	DMSFNC				
DMSITP	000001	DMSDBG				
DMSITSR	000001	DMSABN				
DMSITS1	000001	DMSINI				
DMSLAD	000005	DMSBWR	DMSERS	DMSINS	DMSLFS	DMSSTT
DMSLADAD	000002	DMSABN	DMSFNC			
DMSLADN	000003	DMSABN	DMSLFS			
DMSLADW	000002	DMSERS	DMSSTT			
DMSLDRA	000001	DMSFNC				
DMSLDRB	000001	DMSLOA				
DMSLDRC	000001	DMSLSB				
DMSLDRD	000003	DMSLGT	DMSLIB	DMSLSB		
DMSLFS	000005	DMSBRD	DMSEXC	DMSINT	DMSPNT	DMSSTT

Label	Count	References			
DMSLFSW	000005	DMSBWR	DMSERS	DMSFNS	DMSSTT
DMSLGT	000002	DMSSEG	DMSSVT		
DMSLGTA	000003	DMSLDR	DMSOLD	DMSSTG	
DMSLGTB	000002	DMSLDR	DMSOLD		
DMSLIB	000004	DMSLDR	DMSOLD	DMSSEG	DMSTMA
DMSLIO	000001	DMSLDR			
DMSLOA	000003	DMSFNC	DMSINS		
DMSLSB	000002	DMSSEG	DMSSVT		
DMSLSBA	000002	DMSLDR	DMSOLD		
DMSLSBB	000002	DMSLDR	DMSOLD		
DMSLSBC	000002	DMSLDR	DMSOLD		
DMSLSBD	000002	DMSLDR	DMSOLD		
DMSLSY	000003	DMSLDR	DMSOLD	DMSSEG	
DMSMOD	000003	DMSFNC	DMSITS		
DMSNUCU	000001	DMSFRE			
DMSOLD	000002	DMSSEG	DMSSLN		
DMSOVS	000001	DMSCVR			
DMSPIO	000001	DMSFNC			
DMSPIOCC	000001	DMSFNC			
DMSPIOSI	000001	DMSFNC			
DMSSAB	000003	DMSSEG	DMSSVT		
DMSSBD	000002	DMSSBS	DMSSEG		
DMSSBDFR	000001	DMSSVT			
DMSSBS	000004	DMSSBD	DMSSEG	DMSSOP	DMSSVT
DMSSBSRT	000001	DMSSBD			
DMSSCNN	000002	DMSINS	DMSINT		
DMSSCR	000002	DMSEDI	DMSSEG		
DMSSCT	000002	DMSSEG	DMSSVT		
DMSSCTCE	000002	DMSSOP	DMSSQS		
DMSSCTCK	000003	DMSSOP	DMSSQS		
DMSSCTNP	000001	DMSSOP			
DMSSEB	000005	DMSSBS	DMSSEG	DMSSQS	
DMSSLN	000002	DMSSEG	DMSSVT		
DMSSLN3	000001	DMSSVT			
DMSSLN42	000001	DMSSVT			
DMSSLN6	000001	DMSSVT			
DMSSLN7	000001	DMSSVT			
DMSSLN8	000001	DMSSVT			
DMSSLN9	000001	DMSSVT			
DMSSMN	000002	DMSSEG	DMSSVT		
DMSSMNCP	000001	DMSSTG			
DMSSMNCN	000001	DMSSTG			
DMSSMNRP	000001	DMSSTG			

Label	Count	References											
DMSSMNSB	000001	DMSSLN											
DMSSMNTS	000001	DMSSTG											
DMSSMN10	000001	DMSSVT											
DMSSMN4	000001	DMSSVT											
DMSSMN5	000001	DMSSVT											
DMSSOP	000002	DMSSEG	DMSSVT										
DMSSOP19	000001	DMSSVT											
DMSSOP20	000001	DMSSVT											
DMSSOP22	000001	DMSSVT											
DMSSOP23	000001	DMSSVT											
DMSSQS	000002	DMSSEG	DMSSVT										
DMSSQSGT	000001	DMSSOP											
DMSSQSPT	000001	DMSSOP											
DMSSQSUP	000001	DMSSOP											
DMSSTGAT	000001	DMSFNC											
DMSSTTR	000001	DMSLFS											
DMSSVN	000002	DMSSEG	DMSSVT										
DMSSVN1	000001	DMSSVT											
DMSSVN2	000001	DMSSVT											
DMSSVN93	000001	DMSSVT											
DMSSVN94	000001	DMSSVT											
DMSSVT	000001	DMSSEG											
DMSVSR	000001	DMSFNC											
DMSXCP	000001	DMSDOS											
DOSBLKSZ	000005	DMSBOP											
DOSBUFF	000011	DMSBOP	DMSXCP										
DOSBUFSP	000002	DMSDLB	DMSXCP										
DOSBYTE	000014	DMSXCP											
DOSCBID	000002	DMSDLB	DMSXCP										
DOSCOUT	000002	DMSXCP											
DOSDD	000023	DMSAMS	DMSBOP	DMSCLS	DMSDLB	DMSDLK	DMSDSV	DMSOPL	DMSRRV	DMSSRV	DMSSVT	DMSVIP	DMSXCP
DOSDDCAT	000006	DMSDLB											
DOSDEV	000017	DMSAMS	DMSBOP	DMSDLB	DMSDLK	DMSRRV	DMSSRV	DMSVIP	DMSXCP				
DOSDSK	000006	DMSDLB	DMSDLK	DMSRRV	DMSSRV	DMSXCP							
DOSDSMD	000027	DMSAMS	DMSBOP	DMSDLB	DMSVIP	DMSXCP							
DOSDSNAM	000008	DMSCLS	DMSDLB	DMSXCP									
DOSDSTYP	000001	DMSDLB											
DOSDUM	000012	DMSAMS	DMSBOP	DMSDLB	DMSVIP	DMSXCP							
DOSEND	000001	DMSDLB											
DOSENSIZ	000006	DMSDLB											
DOSEXT	000004	DMSBOP											
DOSEXTCT	000002	DMSBOP											
DOSEXTCX	000004	DMSXCP											

Label	Count	References											
DOSEXTNO	000009	DMSAMS	DMSDLB	DMSVIP	DMSXCP								
DOSEXTTB	000007	DMSAMS	DMSDLB	DMSVIP	DMSXCP								
DOSFORM	000006	DMSBOP											
DOSINIT	000013	DMSBOP	DMSDLB	DMSXCP									
DOSITEM	000007	DMSXCP											
DOSJCAT	000006	DMSDLB											
DOSNEXT	000011	DMSAMS	DMSBOP	DMSCLS	DMSDLB	DMSOPL	DMSSVT	DMSVIP	DMSXCP				
DOSOP	000034	DMSBOP	DMSDLK	DMSRRV	DMSSRV	DMSXCP							
DOSOSDSN	000007	DMSDLB	DMSXCP										
DOSOSFST	000009	DMSBOP	DMSDLB	DMSDLK	DMSRRV	DMSSRV	DMSXCP						
DOSPERM	000002	DMSDLB											
DOSREAD	000009	DMSXCP											
DOSSAVE	000006	DMSXCP											
DOSSECT	000028	DMSAMS	DMSBOP	DMSCLS	DMSDLB	DMSDLK	DMSDSV	DMSOPL	DMSRRV	DMSSRV	DMSSVT	DMSVIP	DMSXCP
DOSSENSE	000008	DMSXCP											
DOSSYS	000002	DMSBOP	DMSOPL										
DOSTAPID	000002	DMSXCP											
DOSUCAT	000006	DMSBOP	DMSDLB										
DOSUCNAM	000007	DMSBOP	DMSDLB	DMSXCP									
DOSVOLNO	000011	DMSAMS	DMSDLB	DMSVIP	DMSXCP								
DOSVOLTB	000007	DMSAMS	DMSDLB	DMSVIP	DMSXCP								
DOSWORK	000004	DMSXCP											
DOSYSXXX	000015	DMSAMS	DMSBOP	DMSCLS	DMSDLB	DMSVIP	DMSXCP						
DOUBLE	000021	DMSDIO											
DSKAD	000002	DMSLIO											
DSKADR	000006	DMSACF	DMSACM	DMSAUD	DMSERS								
DSKLIN	000066	DMSLIO											
DSKLOC	000010	DMSACF	DMSACM	DMSAUD	DMSERS	DMSFNS	DMSMOD						
DSKLST	000020	DMSACF	DMSACM	DMSAUD	DMSERS	DMSFNS	DMSMOD						
DSYM	000002	DMSLSY											
DTAD	000029	DMSACC	DMSACM	DMSAMS	DMSARE	DMSASN	DMSDIO	DMSFOR	DMSINS	DMSQRY	DMSROS		
DTADT	000022	DMSACM	DMSASN	DMSAUD	DMSDIO	DMSQRY	DMSTQQ						
DTAS	000003	DMSAMS											
DUALNOS	000008	DMSEDC											
DUMCOM	000004	DMSSLN											
DUMPLIST	000002	DMSDBG	DMSSVT										
DYLD	000012	DMSLDR	DMSLIO	DMSOLD	DMSSLN	DMSSTG							
DYLIBO	000004	DMSSLN	DMSSTG										
DYMBRNM	000005	DMSLIB	DMSSLN	DMSSTG									
DYNAEND	000004	DMSLDR	DMSOLD	DMSSLN									
EDCB	000005	DMSEDC	DMSEDI	DMSEDX	DMSGIO	DMSSCR							
EDCBEND	000001	DMSEDX											
EDCBLTH	000002	DMSEDX											

Label	Count	References			
EDCT	000026	DMSEDI			
EDLIN	000013	DMSEDI	DMSEDX		
EDMSK	000003	DMSSCR			
EDRET	000003	DMSEDI	DMSEDX		
EDWORK	000001	DMSEDX			
EFPRS	000004	DMSITS	DMSOVS		
EGPRS	000021	DMSABN	DMSBSC	DMSITS	DMSCVS
EGPR0	000062	DMSDLB	DMSFLD	DMSITS	DMSOVS
EGPR1	000037	DMSLDR	DMSSMN		
EGPR11	000002	DMSITS			
EGPR12	000003	DMSSTG			
EGPR14	000009	DMSITS	DMSSTG		
EGPR15	000034	DMSITS	DMSOVS	DMSSMN	DMSSTG
EGPR2	000006	DMSITS			
ENDBLOC	000003	DMSEDI	DMSEDX		
ENDCDADR	000006	DMSLDR	DMSLSB	DMSOLD	
ENDTABS	000004	DMSEDI	DMSEDX		
ENTADR	000008	DMSLDR	DMSOLD		
ENTNAME	000005	DMSLDR	DMSLSB	DMSOLD	
EOCADR	000006	DMSDMP	DMSSMN	DMSSTG	
ERBIT	000006	DMSACF	DMSERS	DMSRNM	
ERBL	000001	DMSERR			
ERDSECT	000002	DMSERR			
ERF1BF	000002	DMSERR			
ERF1HD	000003	DMSERR			
ERF1SBN	000005	DMSERR			
ERF1SB1	000003	DMSERR			
ERF1TX	000002	DMSERR			
ERF2CM	000004	DMSERR			
ERF2DI	000001	DMSERR			
ERF2DT	000001	DMSERR			
ERF2PR	000002	DMSERR			
ERF2SI	000001	DMSERR			
ERLET	000001	DMSERR			
ERMESS	000002	DMSERR			
ERNUM	000002	DMSERR			
ERPAS13	000001	DMSERR			
ERPBFA	000002	DMSERR			
ERPCS	000001	DMSERR			
ERPF1	000013	DMSERR			
ERPF2	000012	DMSERR			
ERPHDR	000001	DMSERR			
ERPLET	000001	DMSERR			

Label	Count	References		
ERPNUM	000001	DMSERR		
ERPSBA	000004	DMSERR		
ERPTXA	000003	DMSERR		
ERRCODE	000063	DMSDIO		
ERRCOD0	000009	DMSACM		
ERRCOD1	000017	DMSACF	DMSERS	DMSRNM
ERRET	000035	DMSITS		
ERRNUM	000002	DMSINT		
ERSAVE	000007	DMSERR		
ERSBD	000013	DMSERR		
ERSBF	000010	DMSERR		
ERSBL	000005	DMSERR		
ERSECT	000001	DMSERR		
ERSFA	000004	DMSERR		
ERSFL	000005	DMSERR		
ERSFLAG	000050	DMSERS	DMSRNM	
ERSFLST	000002	DMSERR		
ERSSZ	000002	DMSERR		
ERTEXT	000004	DMSERR		
ERTPL	000004	DMSERR		
ERTPLA	000006	DMSERR		
ERTPLL	000008	DMSERR		
ERTSIZE	000002	DMSERR		
ERT1	000008	DMSERR		
ERT2	000013	DMSERR		
ESD1ST	000007	DMSLDR	DMSOLD	
ESIDTB	000040	DMSLDR	DMSOLD	
EXADD	000008	DMSEXC	DMSEXT	
EXAMLC	000005	DMSDBG		
EXAMLG	000006	DMSDBG		
EXECFLAG	000003	DMSEXC		
EXECRUN	000004	DMSEXC		
EXENACTB	000009	DMSVIP		
EXENADDR	000002	DMSVIP		
EXLEODF	000004	DMSVIP		
EXLEODL	000001	DMSVIP		
EXLEODP	000001	DMSVIP		
EXLEVEL	000006	DMSEXC	DMSEXT	
EXLJRN	000002	DMSVIP		
EXLJRNL	000004	DMSVIP		
EXLLEN	000009	DMSVIP		
EXLLERF	000004	DMSVIP		
EXLLERL	000001	DMSVIP		

Label	Count	References
EXLLERP	000001	DMSVIP
EXLSYNF	000004	DMSVIP
EXLSYNL	000002	DMSVIP
EXLSYNP	000001	DMSVIP
EXNUM	000003	DMSEXC
EXSAVE	000005	DMSITE
EXSAVE1	000006	DMSITE
EXTFLAG	000006	DMSITE
EXTNPSW	000001	DMSINI
EXTOPSW	000017	DMSDBG DMSITE
EXTPSW	000005	DMSINT DMSITE
EXTRET	000002	DMSITE
EXTSECT	000014	DMSINS DMSINT DMSIOW DMSITE DMSQRY DMSSET DMSSTG DMSSVN DMSSVT
FCBBLKSZ	000005	DMSFLD DMSMVE DMSROS DMSSOP
FCBBUFF	000042	DMSARN DMSARX DMSASM DMSSBS DMSSEB DMSSOP DMSSQS DMSSVT
FCBBYTE	000052	DMSARN DMSARX DMSASM DMSSBD DMSSBS DMSSEB DMSSOP DMSSQS DMSSVT
FCBCASE	000004	DMSFLD DMSSEB DMSSOP
FCBCATML	000019	DMSARN DMSARX DMSASM DMSFLD DMSSBS DMSSCT DMSSOP DMSSVT
FCBCLEAV	000004	DMSSOP
FCBCLOSE	000011	DMSARN DMSARX DMSASM DMSSCT DMSSOP DMSSQS
FCBCON	000002	DMSFLD DMSSOP
FCBCOUT	000025	DMSSBS DMSSCT DMSSEB DMSSOP DMSSQS DMSSVT
FCBDCBCT	000004	DMSSOP
FCBDD	000024	DMSARN DMSARX DMSASM DMSFCH DMSFLD DMSMVE DMSQRY DMSSAB DMSSOP DMSSVT
FCBDEV	000040	DMSARN DMSARX DMSASM DMSFCH DMSFLD DMSMVE DMSQRY DMSSBS DMSSCT DMSSEB DMSSOP DMSSQS DMSSVT
FCBDSK	000009	DMSARX DMSASM DMSFCH DMSFLD DMSMVE DMSSOP DMSSVT
FCBDSMD	000027	DMSALU DMSFLD DMSMVE DMSROS DMSSBS DMSSEB DMSSOP DMSSQS
FCBDSNAM	000033	DMSARX DMSASM DMSFCH DMSFLD DMSMVE DMSQRY DMSROS DMSSBS DMSSCT DMSSOP DMSSVT
FCBDSORG	000005	DMSFLD
FCBDSTYP	000016	DMSFLD DMSQRY DMSROS DMSSEB DMSSOP DMSSVT
FCBENSIZ	000007	DMSFLD
FCBFIRST	000016	DMSABN DMSALU DMSFLD DMSQRY DMSROS DMSSAB DMSSOP DMSSVT
FCBFORM	000008	DMSARN DMSARX DMSASM DMSSOP DMSSVT
FCBINIT	000065	DMSARN DMSARX DMSASM DMSFCH DMSFLD DMSMVE DMSSBS DMSSCT DMSSEB DMSSOP DMSSQS DMSSVT
FCBIO	000001	DMSSEB
FCBIORD	000003	DMSSQS
FCBIOSW	000033	DMSARN DMSARX DMSASM DMSFLD DMSSCT DMSSEB DMSSOP DMSSQS
FCBIOSW2	000017	DMSDSL DMSLDS DMSMVE DMSROS DMSSOP DMSSVT
FCBIOWR	000003	DMSSQS
FCBITEM	000061	DMSARN DMSARX DMSASM DMSDSL DMSMVE DMSSBD DMSSBS DMSSCT DMSSEB DMSSOP DMSSQS DMSSVT
FCBKEYS	000009	DMSSBD DMSSOP DMSSVT
FCBLRECL	000006	DMSFLD DMSMVE DMSROS DMSSOP

Label	Count	References											
FCBMEMBR	000009	DMSFLD	DMSLDS	DMSSOP									
FCBMODE	000006	DMSFLD	DMSSBS	DMSSEB	DMSSOP								
FCBMVPDS	000016	DMSDSL	DMSLDS	DMSMVE	DMSROS	DMSSOP	DMSSVT						
FCBNEXT	000003	DMSALU	DMSROS										
FCBNUM	000013	DMSABN	DMSFLD	DMSQRY									
FCBOP	000116	DMSFCH	DMSMVE	DMSROS	DMSSBD	DMSSBS	DMSSCT	DMSSEB	DMSSOP	DMSSQS	DMSSVT		
FCBOPCB	000003	DMSMVE	DMSSEB										
FCBOS	000017	DMSSBS	DMSSCT	DMSSEB	DMSSOP	DMSSVT							
FCBOSDSN	000013	DMSFLD	DMSLDS	DMSROS									
FCBOSFST	000020	DMSALU	DMSFCH	DMSMVE	DMSROS	DMSSCT	DMSSOP	DMSSVT					
FCBPCH	000001	DMSFLD											
FCBPDS	000011	DMSSBS	DMSSCT	DMSSOP	DMSSVT								
FCBPROC	000009	DMSARN	DMSFLD	DMSROS	DMSSEB	DMSSOP							
FCBPROCC	000005	DMSARN	DMSARX	DMSASM	DMSSOP								
FCBPROCO	000003	DMSARN	DMSSOP										
FCBPRPU	000006	DMSSEB											
FCBPTR	000001	DMSFLD											
FCBPVMB	000003	DMSSQS											
FCBRDR	000004	DMSARX	DMSASM	DMSFLD	DMSSOP								
FCBREAD	000021	DMSARN	DMSARX	DMSASM	DMSSBS	DMSSEB	DMSSQS						
FCBRECFM	000005	DMSFLD	DMSMVE	DMSROS	DMSSOP								
FCBRECL	000005	DMSSEB	DMSSOP										
FCBR13	000002	DMSSCT	DMSSEB										
FCBSECT	000038	DMSALU	DMSARN	DMSARX	DMSASM	DMSDSL	DMSFCH	DMSFLD	DMSLDS	DMSMVE	DMSQRY	DMSROS	DMSSAB
		DMSSBD	DMSSBS	DMSSCT	DMSSEB	DMSSOP	DMSSQS	DMSSVN	DMSSVT				
FCBTAB	000001	DMSSVT											
FCBTAP	000007	DMSARX	DMSASM	DMSFLD	DMSMVE	DMSSBS	DMSSCT						
FCBTAPID	000006	DMSFLD	DMSMVE	DMSQRY	DMSSEB								
FCBXTENT	000010	DMSFLD	DMSSBD	DMSSBS	DMSSVT								
FCHAPHNM	000002	DMSFET											
FCHLENG	000003	DMSDOS	DMSFET										
FCHOPT	000002	DMSFET											
FCHTAB	000008	DMSDOS	DMSFET										
FFD	000005	DMSACM	DMSAUD										
FFE	000002	DMSACM	DMSAUD										
FFF	000004	DMSACM	DMSAUD										
FILE	000074	DMSLGT	DMSLIB	DMSLIO									
FILEBUFF	000024	DMSEXC	DMSROS	DMSSVT									
FILEBYTE	000009	DMSEXC	DMSROS	DMSSOP	DMSSVT								
FILECOUT	000002	DMSSVT											
FILEITEM	000007	DMSSVT											
FILEMODE	000013	DMSEXC	DMSSOP	DMSSVT									
FILEMS	000004	DMSEDI											

Label	Count	References
FILENAME	000044	DMSINT DMSROS DMSSCT DMSSOP DMSSVT
FILEREAD	000002	DMSROS DMSSOP
FILETYPE	000013	DMSINT DMSSOP DMSSVT
FINIS	000064	DMSLDR DMSLIB DMSOLD
FINISLST	000005	DMSAUD DMSDSK DMSFNS DMSINT
FIRSTDMP	000002	DMSDBG
FLAG	000129	DMSEDI DMSEDX DMSSCR
FLAGLOC	000004	DMSEDX DMSSCR
FLAGS	000122	DMSFRE DMSLDR DMSLIB DMSLSB DMSOLD
FLAG1	000057	DMSLDR DMSLIO DMSLSB DMSOLD
FLAG2	000122	DMSEDI DMSEDX DMSLDR DMSLIB DMSLIO DMSLSB DMSOLD DMSSCR
FLCLN	000003	DMSFRE
FLGSAVF	000002	DMSALU
FLHC	000008	DMSFRE
FLNU	000007	DMSFRE
FLPA	000008	DMSFRE
FMODE	000043	DMSEDI DMSEDX DMSLGT DMSLIB DMSSCR
FNAME	000053	DMSEDI DMSEDX DMSLGT DMSLIB DMSLIO DMSSCR
FNBIT	000003	DMSFNS
FPRLOG	000001	DMSDBG
FPTR	000008	DMSEDI
FRDSECT	000004	DMSFRE DMSSET
FREEFLG1	000028	DMSFRE
FREEFLG2	000028	DMSFRE
FREEHN	000006	DMSFRE
FREEHU	000006	DMSFRE
FREELEN	000006	DMSEDI DMSEDX
FREELN	000013	DMSFRE
FREELOWE	000052	DMSABN DMSARX DMSASM DMSBSC DMSDLK DMSDOS DMSDSV DMSFCH DMSFRE DMSINS DMSINT DMSLBM
		DMSLDR DMSLSB DMSMOD DMSOLD DMSSET DMSSLN DMSSMN DMSSRT DMSSTG
FREELOW1	000004	DMSFRE DMSSET
FREELU	000006	DMSFRE
FREERO	000003	DMSDIO
FREESAVE	000012	DMSFRE
FRF1B	000002	DMSFRE
FRF1C	000003	DMSFRE
FRF1E	000003	DMSFRE
FRF1H	000006	DMSFRE
FRF1L	000006	DMSFRE
FRF1M	000004	DMSFRE
FRF1N	000003	DMSFRE
FRF1V	000003	DMSFRE
FRF2CKE	000003	DMSFRE

Label	Count	References
FRF2CKT	000007	DMSFRE
FRF2CKX	000003	DMSFRE
FRF2CL	000004	DMSFRE
FRF2NOI	000010	DMSFRE
FRF2SVP	000003	DMSFRE
FRSTLOC	000008	DMSMOD DMSSLN
FRSTSDID	000002	DMSLDR DMSLSB
FSCBBUFF	000001	DMSDLK
FSCBD	000012	DMSDLK
FSCBFM	000003	DMSDLK
FSCBFN	000022	DMSDLK
FSCBFV	000001	DMSDLK
FSCBITNO	000013	DMSDLK
FSIZE	000009	DMSEDI
FSTBKWD	000001	DMSERS
FSTD	000011	DMSCPY DMSEDX DMSEXC DMSGND DMSNCP DMSTPE
FSTDATEW	000001	DMSGND
FSTDBC	000006	DMSDSK DMSERS DMSTPE
FSTFACT	000001	DMSCPY
FSTFAP	000001	DMSSTT
FSTFAR	000001	DMSSTT
FSTFAW	000001	DMSSTT
FSTFB	000008	DMSCPY DMSDLK DMSSTT DMSZAP
FSTFCL	000003	DMSERS DMSTPE
FSTFINRD	000012	DMSCAT DMSCIT DMSCRD DMSEDX DMSEXT DMSINT DMSSVN
FSTFMODE	000007	DMSEDX DMSNCP
FSTFRO	000001	DMSSTT
FSTFROX	000001	DMSSTT
FSTFRW	000003	DMSDLK DMSSTT DMSZAP
FSTFRWX	000002	DMSDLK DMSSTT
FSTFV	000023	DMSAMS DMSARX DMSASM DMSBRD DMSBSC DMSBWR DMSCPY DMSDLK DMSDSK DMSLBM DMSLKD DMSMVE
		DMSTPE DMSUPD DMSZAP
FSTFWDP	000002	DMSERS
FSTIC	000017	DMSACF DMSBOP DMSBRD DMSCPY DMSDLK DMSDSK DMSLBM DMSMDP DMSTPE DMSZAP
FSTIL	000023	DMSAMS DMSARX DMSASM DMSBSC DMSBWR DMSCPY DMSDLK DMSDSK DMSLBM DMSLKD DMSMVE DMSTPE
		DMSUPD DMSZAP
FSTL	000006	DMSARN DMSARX DMSASM DMSBSC DMSDSL DMSLAF
FSTLRECL	000001	DMSEXC
FSTM	000030	DMSAMS DMSARN DMSARX DMSASM DMSBOP DMSBSC DMSCPY DMSDLK DMSDSK DMSERS DMSLBM DMSLKD
		DMSRNM DMSSTT DMSTPE DMSUPD DMSZAP
FSTN	000014	DMSAMS DMSCPY DMSDSK DMSERS DMSRNM DMSTPE
FSTRECCT	000001	DMSEDX
FSTRECFM	000001	DMSEDX

Label	Count	References											
FSTRP	000004	DMSACF	DMSBRD	DMSTPE									
FSTSECT	000048	DMSACF	DMSAMS	DMSARN	DMSARX	DMSASM	DMSBOP	DMSBRD	DMSBSC	DMSBWR	DMSCPY	DMSDLK	DMSDSK
		DMSDSL	DMSERS	DMSLAF	DMSLBM	DMSLKD	DMSMDP	DMSMVE	DMSRNM	DMSSTT	DMSTPE	DMSUPD	DMSZAP
FSTT	000009	DMSACF	DMSDSK	DMSERS	DMSRNM	DMSTPE							
FSTWP	000009	DMSACF	DMSBWR	DMSTPE									
FSTXTADR	000007	DMSLDR	DMSLOA	DMSLSB	DMSOLD								
FSTYR	000006	DMSCPY											
FTYPE	000016	DMSEDI	DMSEDX	DMSLGT	DMSLIB	DMSSCR							
FV	000013	DMSEDI	DMSEDX	DMSSCR									
FVS	000002	DMSITE											
FVSDSKA	000002	DMSACM	DMSAUD										
FVSECT	000047	DMSABN	DMSACC	DMSACF	DMSACM	DMSALU	DMSAUD	DMSBTB	DMSBTP	DMSBWR	DMSCIT	DMSCRD	DMSCWR
		DMSCWT	DMSDIO	DMSDSK	DMSERS	DMSFNS	DMSINT	DMSITE	DMSITI	DMSITP	DMSITS	DMSLFS	DMSMOD
		DMSQRY	DMSRNM	DMSSLN	DMSTPE	DMSTQQ							
FVSERAS0	000013	DMSERS	DMSRNM										
FVSERAS1	000012	DMSERS	DMSRNM										
FVSERAS2	000004	DMSERS	DMSRNM										
FVSFSTAD	000004	DMSMOD	DMSPUN	DMSSTT									
FVSFSTCL	000001	DMSMOD											
FVSFSTDT	000002	DMSSTT											
FVSFSTFV	000001	DMSMOD											
FVSFSTIC	000003	DMSACM	DMSBTB	DMSMOD									
FVSFSTIL	000003	DMSACM	DMSBTB	DMSMOD									
FVSFSTM	000002	DMSDSK	DMSSTT										
FVSFSTN	000001	DMSSTT											
FW4	000002	DMSACC	DMSACF										
FXD	000021	DMSDSL	DMSSEB	DMSSOP	DMSSQS								
F0	000008	DMSDBG	DMSINS	DMSITE	DMSITS								
F2	000008	DMSITE											
F4	000010	DMSITE											
F6	000012	DMSDBG	DMSITE	DMSITS	DMSSOP								
F65535	000008	DMSACF	DMSDSK	DMSMOD	DMSPNT	DMSSLN	DMSTQQ						
F800	000004	DMSACM	DMSAUD	DMSDSK									
GETFLAG	000007	DMSEDI											
GET1	000002	DMSLSY											
GIOPLIST	000001	DMSSCR											
GPRLOG	000008	DMSDBG											
GPRSAV	000004	DMSLDR	DMSOLD										
GRAFDEV	000001	DMSINS											
HALF	000002	DMSEDI											
HEX	000043	DMSDBG											
HEXHEX	000010	DMSDBG											
HIPHAS	000005	DMSFCH	DMSFET										

Label	Count	References									
HIPROG	000002	DMSFCH									
HOLD	000012	DMSITI									
HOLDFLAG	000015	DMSSCR									
IADT	000002	DMSACC	DMSLAD								
IHADEB	000017	DMSFCH	DMSMVE	DMSSBS	DMSSCT	DMSSOP	DMSSQS	DMSSVT			
IHADECB	000006	DMSSBD	DMSSBS	DMSSCT	DMSSEB	DMSSVT					
IHAJFCB	000001	DMSSVT									
IJBABTAB	000005	DMSBAB	DMSDOS	DMSITP							
IJBBOX	000001	DMSSTG									
IJBCCWT	000001	DMSDOS									
IJBFLG04	000001	DMSBOP									
IJBFTTAB	000004	DMSDOS	DMSFET								
IKQACB	000007	DMSBOP	DMSVIP								
IKQEXLST	000003	DMSVIP									
IKQRPL	000006	DMSVIP									
INCRNO	000003	DMSEDI									
INPUT	000062	DMSDBG									
INPUTSIZ	000002	DMSDBG									
INPUT1	000002	DMSDBG									
INSTALID	000005	DMSINI	DMSPRT								
INTINFC	000004	DMSDOS	DMSITP								
INVLD	000003	DMSEDI	DMSEDX								
IOAD	000002	DMSEDX									
IOBBCSW	000003	DMSSBS	DMSSEB								
IOBBECBC	000002	DMSSEB									
IOBBECBP	000003	DMSSBS	DMSSEB								
IOBBFLG	000002	DMSSBS	DMSSCT								
IOBCSW	000005	DMSARN	DMSARX	DMSASM	DMSSCT						
IOBDCBPT	000001	DMSSOP									
IOBECB	000002	DMSSQS									
IOBECBPT	000003	DMSSQS									
IOBEND	000001	DMSSOP									
IOBIN	000031	DMSARN	DMSARX	DMSASM	DMSSBD	DMSSBS	DMSSEB	DMSSQS	DMSSVT		
IOBIOFLG	000044	DMSARN	DMSARX	DMSASM	DMSSBD	DMSSBS	DMSSCT	DMSSEB	DMSSOP	DMSSQS	DMSSVT
IOBNXTAD	000003	DMSSOP									
IOBOUT	000007	DMSSBS	DMSSCT	DMSSQS							
IOBSTART	000008	DMSSOP	DMSSQS								
IOBUPD	000004	DMSSQS									
IOCOMM	000007	DMSDIO									
IOID	000002	DMSEDI	DMSEDX								
IOLIST	000048	DMSEDI	DMSEDX								
IOMODE	000003	DMSEDI	DMSEDX								
IONPSW	000006	DMSINI	DMSINS	DMSIOW	DMSITE						

Label	Count	References						
IONTABL	000012	DMSABN	DMSHDI	DMSINT	DMSITI			
IOOLD	000002	DMSDIO	DMSITI					
IOOPSW	000025	DMSCIT	DMSDBG	DMSDIO	DMSINI	DMSIOW	DMSITE	DMSITI
IOPSW	000001	DMSITI						
IOSAVE	000005	DMSITI						
IOSECT	000004	DMSABN	DMSHDI	DMSINT	DMSITI			
IPLADDR	000003	DMSBTP	DMSINS					
IPLCCW1	000001	DMSINI						
IPLPSW	000009	DMSABN	DMSDBG	DMSINI	DMSINS			
IS	000003	DMSZAP						
ITEM	000055	DMSEDI	DMSEDX	DMSSCR				
ITSBIT	000007	DMSITS						
JAR	000003	DMSEDI	DMSEDX					
JCSW2	000001	DMSDOS						
JCSW3	000016	DMSOPT	DMSSET					
JCSW4	000005	DMSDOS	DMSOPT	DMSSET				
JFCBIND2	000002	DMSFLD	DMSSOP					
JFCBMASK	000022	DMSSOP	DMSSVT					
JFCBUFNO	000001	DMSFLD						
JFCDSORG	000002	DMSSOP						
JFCKEYLE	000003	DMSFLD	DMSSOP					
JFCLIMCT	000003	DMSFLD	DMSSOP					
JFCLRECL	000001	DMSSVT						
JFCOPTCD	000008	DMSFLD	DMSSOP					
JFIRST	000009	DMSHDS						
JFLAGS	000014	DMSDBG						
JLAST	000010	DMSHDS						
JNUMB	000012	DMSHDS	DMSINT					
JOBDATE	000003	DMSDLK	DMSDOS					
JR0	000002	DMSITE						
JR1	000001	DMSITE						
JSR0	000009	DMSACF	DMSACM					
JSR1	000002	DMSACF						
JSYM	000002	DMSLSY						
KEYCHNG	000006	DMSSBD	DMSSVT					
KEYCOUT	000004	DMSSBD	DMSSVT					
KEYFORM	000002	DMSSVT						
KEYLNGTH	000010	DMSSBD	DMSSVT					
KEYMAX	000001	DMSITS						
KEYNAME	000007	DMSSBD	DMSSVT					
KEYOP	000009	DMSSBD	DMSSVT					
KEYP	000008	DMSITS						
KEYS	000003	DMSITS						

Label	Count	References											
KEYSECT	000002	DMSSBD	DMSSVT										
KEYTABLE	000011	DMSSVT											
KEYTBLAD	000009	DMSSBD	DMSSVT										
KEYTBLNO	000016	DMSSBD	DMSSVT										
KEYTYPE	000002	DMSSVT											
KXFLAG	000011	DMSABN	DMSCIT	DMSCRD	DMSCWR	DMSCWT	DMSITI	DMSITS					
KXWANT	000004	DMSABN	DMSCIT	DMSITI	DMSITS								
KXWSVC	000005	DMSCRD	DMSCWR	DMSCWT	DMSITS								
LABLEN	000003	DMSDLK											
LASTCMND	000010	DMSEXT	DMSINT										
LASTCYL	000003	DMSDIO											
LASTDMP	000001	DMSDBG											
LASTEXEC	000002	DMSEXT											
LASTHED	000003	DMSDIO											
LASTLINE	000010	DMSDBD											
LASTLMOD	000002	DMSMOD	DMSSLN										
LASTLOC	000001	DMSFET											
LASTTMOD	000008	DMSITS	DMSLSB	DMSMOD	DMSSLN								
LDMSROS	000004	DMSABN	DMSACM	DMSALU									
LDRADDR	000014	DMSLDR	DMSLIO	DMSLOA	DMSCLD								
LDRFLAGS	000018	DMSLDR	DMSLOA	DMSMOD	DMSOLD	DMSSLN							
LDRRTCD	000002	DMSLDR	DMSOLD										
LDRST	000009	DMSLDR	DMSLGT	DMSLIB	DMSLIO	DMSLSB	DMSOLD						
LENOVS	000003	DMSITS	DMSOVR										
LINE	000040	DMSDBD	DMSEDI	DMSEDX									
LINELOC	000002	DMSEDX	DMSSCR										
LINE1	000002	DMSDBD											
LINE1A	000001	DMSDBD											
LINE1B	000001	DMSDBD											
LINE1C	000001	DMSDBD											
LINKLAST	000007	DMSSAB	DMSSLN	DMSSTG									
LINKSTRT	000009	DMSSLN	DMSSTG	DMSSVT									
LMCURR	000005	DMSEDI											
LMINCR	000005	DMSEDI											
LMSTART	000009	DMSEDI	DMSEDX										
LOC	000145	DMSABN	DMSALU	DMSFCH	DMSFLD	DMSROS	DMSSAB	DMSSOP	DMSSQS	DMSSVN	DMSSVT	DMSZAP	
LOCCNT	000034	DMSACM	DMSBTB	DMSEDX	DMSFET	DMSFRE	DMSINS	DMSLDR	DMSLOA	DMSMOD	DMSOLD	DMSSET	DMSSLN
		DMSSMN	DMSSTG										
LOCCT	000024	DMSLDR	DMSLSB	DMSOLD									
LOWSAVE	000003	DMSDBG	DMSSVT										
LSTFINBD	000005	DMSCIT	DMSCRD	DMSSVN									
LTK	000009	DMSAMS	DMSDOS	DMSITP	DMSSET								
LUBPT	000016	DMSAMS	DMSBOP	DMSCLS	DMSDLB	DMSFCH	DMSLLU	DMSOPL	DMSPRV	DMSRRV	DMSSET	DMSSRV	DMSXCP

Label	Count	References											
LUNDEF	000012	DMSLDR	DMSOLD										
MACDIRC	000011	DMSABN	DMSSCT	DMSSOP	DMSSTG	DMSSVT							
MACLIBL	000009	DMSGLB	DMSQRY	DMSSCT	DMSSOP	DMSSTG	DMSSVT						
MAINAD	000003	DMSEDX											
MAINHIGH	000039	DMSARX	DMSASM	DMSBSC	DMSDOS	DMSFRE	DMSINS	DMSLDR	DMSLSB	DMSSET	DMSSMN	DMSSRT	DMSSTG
MAINLIST	000010	DMSDOS	DMSSMN	DMSSTG									
MAINSTRT	000005	DMSDOS	DMSSMN	DMSSTG									
MAX	000014	DMSFRE											
MAXCODE	000002	DMSFRE											
MCKM	000014	DMSINI	DMSINS	DMSITS									
MCKNPSW	000001	DMSINI											
MEMBOUND	000008	DMSLDR	DMSOLD										
MISFLAGS	000033	DMSABN	DMSACC	DMSAMS	DMSARN	DMSARX	DMSASM	DMSBSC	DMSCPY	DMSCRD	DMSEXC	DMSINS	DMSINT
		DMSITS	DMSLBM	DMSLBT	DMSLKD	DMSQRY	DMSSET	DMSSMN	DMSSRT	DMSSTG	DMSUPD		
MODLIST	000002	DMSSLN											
MSGFLAGS	000024	DMSCAT	DMSCIT	DMSCRD	DMSCWR	DMSEDI	DMSEXT	DMSINS	DMSINT	DMSQRY	DMSSET	DMSTYP	
MVCNT	000001	DMSDBG											
MVCNT2	000001	DMSDBG											
NEED	000007	DMSLDR	DMSOLD										
NEWBLKS	000005	DMSSVT											
NEWMODE	000009	DMSEDI											
NEWNAME	000019	DMSEDI											
NEWTYPE	000005	DMSEDI											
NEXTO	000001	DMSITI											
NICLPT	000005	DMSBOP	DMSCLS	DMSDLB	DMSLLU	DMSXCP							
NOABBREV	000006	DMSINA	DMSINT	DMSQRY	DMSSET								
NOAUTO	000007	DMSLDR	DMSLIB	DMSLOA	DMSLSB	DMSOLD							
NODUP	000006	DMSLDR	DMSLSB	DMSOLD									
NOERASE	000008	DMSLIO	DMSLOA										
NOIMPCP	000007	DMSINT	DMSQRY	DMSSET									
NOIMPEX	000004	DMSINT	DMSQRY	DMSSET									
NOINV	000005	DMSLDR	DMSLOA	DMSLSB	DMSOLD								
NOLIBE	000007	DMSLDR	DMSLIB	DMSLOA	DMSLSB	DMSOLD							
NOMAP	000007	DMSLIO	DMSLOA	DMSLSB									
NOP	000014	DMSINI											
NOPAGREL	000005	DMSABN	DMSINT	DMSQRY	DMSSET								
NORDYMSG	000002	DMSSET											
NORDYTIM	000006	DMSINT	DMSQRY	DMSSET									
NOREP	000006	DMSLDR	DMSLOA	DMSLSB	DMSOLD								
NOSLCADR	000006	DMSLDR	DMSOLD										
NOSTDSYN	000005	DMSINA	DMSQRY	DMSSYN									
NOTEXT	000008	DMSDOS	DMSFET										
NOTYPING	000010	DMSCAT	DMSCIT	DMSCRD	DMSCWR	DMSEDI	DMSEXT	DMSINT	DMSTYP				

Label	Count	References
NOVMREAD	000003	DMSINS DMSINT DMSSET
NRMRET	000009	DMSABN DMSITS
NUCCODE	000004	DMSFRE
NUCKEY	000002	DMSFRE DMSSET
NUCON	000168	DMSABN DMSACC DMSACF DMSACM DMSALU DMSAMS DMSARE DMSARN DMSARX DMSASM DMSASN DMSAUD DMSBAB DMSBOP DMSBRD DMSBSC DMSBTB DMSBTP DMSBWR DMSCAT DMSCIO DMSCIT DMSCLS DMSCMP DMSCPF DMSCPY DMSCRD DMSCWR DMSCWT DMSDBD DMSDBG DMSDIO DMSDLB DMSDLK DMSDMP DMSDOS DMSDSK DMSDSL DMSDSV DMSEDI DMSEDX DMSERR DMSERS DMSEXC DMSEXT DMSFCH DMSFET DMSFLD DMSFNS DMSFOR DMSFRE DMSGIO DMSGLB DMSGND DMSHDI DMSHDS DMSINA DMSINI DMSINM DMSINS DMSINT DMSIOW DMSITE DMSITI DMSITP DMSITS DMSLBM DMSLBT DMSLDR DMSLDS DMSLFS DMSLGT DMSLIB DMSLIO DMSLKD DMSLLU DMSLOA DMSLSB DMSLST DMSLSY DMSMDP DMSMOD DMSMVE DMSOLD DMSOPL DMSOPT DMSOR1 DMSOVR DMSOVS DMSPIO DMSPNT DMSPRT DMSPRV DMSPUN DMSQRY DMSRDC DMSRNE DMSRNM DMSROS DMSRRV DMSSAB DMSSBS DMSSCN DMSSCT DMSSEB DMSSET DMSSLN DMSSMN DMSSOP DMSSQS DMSSRT DMSSRV DMSSSK DMSSTG DMSSTT DMSSVN DMSSVT DMSSYN DMSTIO DMSTPD DMSTPE DMSTQQ DMSTYP DMSUPD DMSVIB DMSVIP DMSVSR DMSXCP DMSZAP
NUCRSV3	000001	DMSDOS
NUM	000562	DMSFRE DMSSET
NUMBYTE	000005	DMSLDR DMSLIB DMSOLD
NUMFINRD	000014	DMSABN DMSBTP DMSCAT DMSCIT DMSCRD DMSSVN
NUMLOC	000002	DMSEDX DMSSCR
NUMPNDWR	000016	DMSCIT DMSCRD DMSCWR DMSCWT DMSITE DMSSVN
NXTSYM	000004	DMSLDR DMSLSY DMSOLD
OLDEST	000001	DMSITI
OLDPSW	000069	DMSABN DMSBSC DMSERR DMSITS DMSSTG
OPSECT	000028	DMSABN DMSARX DMSASM DMSCRD DMSCWR DMSCWT DMSDBG DMSEXC DMSEXT DMSINS DMSINT DMSROS DMSSBD DMSSBS DMSSCT DMSSEB DMSSOP DMSSQS DMSSVN DMSSVT
OPSW	000012	DMSITP
OPTFLAGS	000030	DMSABN DMSINA DMSINS DMSINT DMSQRY DMSSET DMSSYN
OPTNBYTE	000002	DMSSTG
CRG	000004	DMSDBG
OSADTDSK	000007	DMSLDS DMSROS
OSADTFST	000006	DMSABN DMSALU DMSROS
OSADTVTA	000008	DMSACM DMSLDS DMSROS
OSADTVTB	000008	DMSLDS DMSROS
OSFST	000013	DMSABN DMSALU DMSBOP DMSDLK DMSFCH DMSMVE DMSROS DMSRRV DMSSOP DMSSRV DMSSTT
OSFSTALT	000009	DMSROS
OSFSTBLK	000005	DMSMVE DMSROS DMSSOP
OSFSTCHR	000013	DMSROS DMSSOP
OSFSTDBK	000002	DMSROS
OSFSTDSK	000006	DMSDLK DMSFCH DMSROS DMSRRV DMSSRV
OSFSTDSN	000003	DMSROS
OSFSTEND	000007	DMSROS
OSFSTEX4	000006	DMSROS

Label	Count	References											
OSFSTFLG	000023	DMSROS	DMSSTT										
OSFSTFM	000007	DMSBOP	DMSROS	DMSSTT									
OSFSTFVF	000002	DMSROS											
OSFSTLRL	000005	DMSMVE	DMSROS	DMSSOP									
OSFSTLTH	000005	DMSABN	DMSALU	DMSROS									
OSFSTMVL	000001	DMSROS											
OSFSTNTE	000010	DMSROS											
OSFSTNXT	000005	DMSABN	DMSALU	DMSROS									
OSFSTRFM	000012	DMSBOP	DMSMVE	DMSROS	DMSSOP								
OSFSTRSW	000009	DMSROS											
OSFSTTRK	000008	DMSROS											
OSFSTTYP	000003	DMSROS											
OSFSTUMV	000001	DMSROS											
OSFSTXNO	000005	DMSBOP	DMSROS										
OSFSTXTN	000013	DMSBOP	DMSDLK	DMSFCH	DMSROS	DMSRRV	DMSSRV						
OSIOTYPE	000015	DMSARX	DMSASM	DMSSBS	DMSSOP	DMSSQS	DMSSVT						
OSRESET	000010	DMSEXT	DMSINT	DMSLDR	DMSOLD	DMSSLN	DMSSVT						
OSSFLAGS	000057	DMSARN	DMSARX	DMSASM	DMSCIT	DMSEXT	DMSINT	DMSITE	DMSLDR	DMSLIB	DMSLIO	DMSOLD	DMSSLN
		DMSSMN	DMSSTG	DMSSVN	DMSSVT								
OSSMNU	000007	DMSSMN	DMSSTG										
OUTBUF	000054	DMSLDR	DMSLGT	DMSLIB	DMSLIO	DMSLSB	DMSOLD						
OUTPT1	000009	DMSDBG											
OUTPUT	000031	DMSLDR	DMSLIO	DMSOLD									
OVAPF	000004	DMSOVR	DMSOVS										
OVBPF	000005	DMSOVR	DMSOVS										
OVF1F	000001	DMSOVR											
OVF1FS	000001	DMSOVR											
OVF1GA	000001	DMSOVR											
OVF1GB	000001	DMSOVR											
OVF1GS	000001	DMSOVR											
OVF1ON	000009	DMSOVR	DMSOVS										
OVF1PA	000001	DMSOVR											
CVF2CM	000002	DMSOVR											
OVF2NR	000002	DMSOVR											
OVF2OS	000002	DMSOVR											
OVF2WA	000002	DMSOVR											
OVIND	000001	DMSBSC											
OVSAFT	000004	DMSOVS											
OVSECT	000003	DMSITS	DMSOVR										
OVSHO	000004	DMSCIT	DMSOVR	DMSOVS									
OVSON	000009	DMSCIT	DMSOVR	DMSOVS									
OVSSO	000006	DMSCIT	DMSOVR	DMSOVS									
OVSTAT	000019	DMSCIT	DMSOVR	DMSOVS									

Label	Count	References											
PACK	000026	DMSLIO											
PADBUF	000017	DMSEDI	DMSEDX										
PADCHAR	000007	DMSEDI	DMSEDX										
PARMLIST	000013	DMSLDR	DMSLIO	DMSOLD									
PCPTR	000005	DMSBAB	DMSDOS	DMSITP									
PDSBLKSI	000008	DMSSVT											
PDSDIR	000003	DMSSVT											
PDSSECT	000002	DMSSTG	DMSSVT										
PENDREAD	000022	DMSCIT	DMSCRD	DMSCWR	DMSCWT	DMSITE	DMSSVN						
PENDWRIT	000011	DMSCIT	DMSCWR	DMSSVN									
PGMNPSW	000006	DMSABN	DMSINS	DMSITP									
PGMOPSW	000016	DMSABN	DMSDBG	DMSITP	DMSSAB								
PGMSECT	000006	DMSITP	DMSSAB	DMSSLN	DMSSTG	DMSSVT							
PIBADR	000011	DMSBAB	DMSDOS	DMSITP									
PIBFLG	000001	DMSDOS											
PIBPT	000023	DMSAMS	DMSBAB	DMSBOP	DMSCLS	DMSDOS	DMSITP	DMSSET					
PIBSAVE	000016	DMSBAB	DMSDOS	DMSITP									
PIB2PTR	000002	DMSDOS	DMSVSR										
PICADDR	000004	DMSITP	DMSSTG										
PIE	000002	DMSITP											
PIK	000017	DMSBAB	DMSDOS	DMSITP	DMSVSR								
PLIST	000115	DMSEXC	DMSINT	DMSSOP	DMSSVT								
PLISTSAV	000016	DMSLDR	DMSOLD										
PNOTFND	000008	DMSDOS	DMSFET										
PO	000013	DMSDSL	DMSFCH	DMSLDS	DMSROS	DMSSBS	DMSSOP						
POINTER	000026	DMSFRE											
PPBEG	000002	DMSDOS											
PPEND	000019	DMSDMP	DMSDOS	DMSSET	DMSSMN	DMSSTG	DMSVSR						
PREVCMND	000004	DMSEXT	DMSINT										
PREVEXEC	000001	DMSEXT											
PREVIOUS	000016	DMSSBS	DMSSOP	DMSSQS	DMSSVT								
PREXIST	000004	DMSLDR	DMSOLD										
PRFPOFF	000009	DMSDBG	DMSFRE	DMSITS	DMSQRY	DMSSET							
PRFTSYS	000005	DMSINS	DMSITS	DMSMOD	DMSSLN								
PRFUSYS	000003	DMSITS	DMSMOD	DMSSLN									
PRHOLD	000005	DMSLDR	DMSLOA	DMSOLD									
PRINTER1	000001	DMSDBD											
PRINTLST	000001	DMSSEB											
PROTFLAG	000017	DMSDBG	DMSFRE	DMSINS	DMSITS	DMSMOD	DMSQRY	DMSSET	DMSSLN				
PRVCNT	000010	DMSLDR	DMSOLD										
PS	000019	DMSDSL	DMSFCH	DMSMVE	DMSROS	DMSSBD	DMSSBS	DMSSCT	DMSSEB	DMSSOP	DMSSQS	DMSSVN	DMSSVT
PSAVE	000011	DMSITP											
PSW	000002	DMSLDR											

Label	Count	References											
PTR1	000015	DMSEDI	DMSEDX	DMSSCR									
PTR2	000036	DMSEDI	DMSEDX	DMSSCR									
PTR3	000008	DMSEDI	DMSEDX										
PUBADR	000017	DMSBOP	DMSCLS	DMSDLK	DMSDSV	DMSLLU	DMSPRV	DMSXCP					
PUBCUU	000013	DMSBOP	DMSCLS	DMSDLK	DMSDSV	DMSLLU	DMSPRV	DMSXCP					
PUBDEVT	000039	DMSBOP	DMSCLS	DMSDLK	DMSLLU	DMSXCP							
PUBDSKM	000002	DMSLLU	DMSXCP										
PUBPT	000017	DMSAMS	DMSASN	DMSBOP	DMSCLS	DMSDLB	DMSDLK	DMSDSV	DMSFCH	DMSLLU	DMSPRV	DMSRRV	DMSSET
		DMSSRV	DMSXCP										
PUBTAPM1	000005	DMSBOP	DMSCLS	DMSXCP									
PUBTAPM2	000016	DMSBOP											
PUBTAP7	000001	DMSBOP											
PUNCHLST	000001	DMSSEB											
QQDSK1	000001	DMSDIO											
QQDSK2	000007	DMSDIO											
QQTRK	000008	DMSDIO											
QS	000003	DMSSOP											
QSWITCH	000003	DMSCRD	DMSINT										
RADD	000005	DMSLGT	DMSLIB										
RANGE	000009	DMSEDI											
RDBUFF	000002	DMSSEB											
RDCCW	000001	DMSSEB											
RDCONS	000001	DMSINI											
RDCOUNT	000004	DMSSEB											
RDDATA	000027	DMSINI											
READBUF	000029	DMSLDR	DMSLGT	DMSLIB	DMSCLD								
READLST	000002	DMSSEB											
RECS	000002	DMSEDX											
REDERRID	000005	DMSCWR	DMSINT	DMSQRY	DMSSET								
REGSAV	000025	DMSEDI											
REGSAVX	000007	DMSEDI											
REGSAV0	000028	DMSACM	DMSACM	DMSALU	DMSAUD	DMSLAD	DMSLFS						
REGSAV1	000012	DMSACF	DMSERS	DMSRNM									
REGSAV3	000031	DMSBRD	DMSBWR	DMSFNS	DMSMOD	DMSPNT	DMSSTT						
REG13SAV	000003	DMSLDR	DMSOLD										
RELPAGES	000017	DMSABN	DMSAMS	DMSARN	DMSARX	DMSASM	DMSBSC	DMSCPY	DMSINT	DMSLBM	DMSLBT	DMSLKD	DMSSMN
		DMSSRT	DMSSTG	DMSUPD									
REPCNT	000010	DMSEDI	DMSEDX										
RESET	000087	DMSLDR	DMSLSB	DMSOLD									
RETREG	000009	DMSLDR	DMSOLD										
RETRYBIT	000002	DMSSAB											
RETSAV	000006	DMSDBG											
RETT	000005	DMSLSB											

Label	Count	References											
RFIX	000001	DMSLGT											
RFPRS	000001	DMSOVS											
RGPRS	000007	DMSOVS											
RGPR8	000001	DMSOVS											
RITEM	000004	DMSLGT	DMSLIB										
RLDCONST	000008	DMSLDR	DMSOLD										
RLENG	000002	DMSLGT	DMSLIB										
RMSGBUF	000011	DMSINT											
RMSROPEN	000001	DMSBOP											
RNUM	000002	DMSLGT	DMSLIB										
RPLACB	000003	DMSVIP											
RPLAREA	000001	DMSVIP											
RPLARG	000001	DMSVIP											
RPLASY	000002	DMSVIP											
RPLBUFL	000001	DMSVIP											
RPLCHAIN	000006	DMSVIP											
RPLECBPR	000004	DMSVIP											
RPLEOFDS	000001	DMSVIP											
RPLFDBKC	000003	DMSVIP											
RPLFLAG	000004	DMSVIP											
RPLIST	000005	DMSEDI											
RPLKEYL	000001	DMSVIP											
RPLNUP	000001	DMSVIP											
RPLOPT1	000004	DMSVIP											
RPLOPT2	000001	DMSVIP											
RPLRLEN	000001	DMSVIP											
RPLRTNCD	000006	DMSVIP											
RPLST	000002	DMSVIP											
RPLSTRID	000001	DMSVIP											
RPLUPD	000001	DMSVIP											
RPLVLERR	000001	DMSVIP											
RSTNPSW	000002	DMSDBG											
RWCCW	000003	DMSDIO											
RWCNT	000004	DMSACF	DMSAUD	DMSMOD									
RWFSTRG	000009	DMSAUD	DMSBRD	DMSBWR	DMSFNS								
RWMFD	000010	DMSACM	DMSAUD										
R0	002247	DMSABN	DMSACC	DMSACF	DMSACM	DMSALU	DMSAMS	DMSARE	DMSARN	DMSARX	DMSASM	DMSASN	DMSAUD
		DMSBAB	DMSBOP	DMSBRD	DMSBSC	DMSBTB	DMSBTP	DMSBWR	DMSCAT	DMSCIO	DMSCIT	DMSCLS	DMSCPF
		DMSCRD	DMSCWR	DMSCWT	DMSDBD	DMSDBG	DMSDIO	DMSDLB	DMSDMP	DMSDOS	DMSDSK	DMSDSL	DMSEDC
		DMSEDI	DMSEDX	DMSERS	DMSEXC	DMSEXT	DMSFCH	DMSFET	DMSFLD	DMSFNS	DMSFOR	DMSGIO	DMSGLB
		DMSGND	DMSGRN	DMSHDI	DMSHDS	DMSINA	DMSINI	DMSINM	DMSINS	DMSINT	DMSIOW	DMSITE	DMSITI
		DMSITP	DMSITS	DMSLAD	DMSLAF	DMSLBM	DMSLBT	DMSLDR	DMSLDS	DMSLFS	DMSLGT	DMSLIB	DMSLIO
		DMSLKD	DMSLLU	DMSLOA	DMSLSB	DMSLST	DMSLSY	DMSMDP	DMSMOD	DMSMVE	DMSNCP	DMSOLD	DMSOPL
		DMSOPT	DMSOR1	DMSOVR	DMSOVS	DMSPNT	DMSPRT	DMSPRV	DMSPUN	DMSQRY	DMSRDC	DMSRNE	DMSRNM
		DMSROS	DMSRRV	DMSSAB	DMSSBD	DMSSBS	DMSSCN	DMSSCR	DMSSCT	DMSSEB	DMSSET	DMSSMN	DMSSOP
		DMSSQS	DMSSRT	DMSSRV	DMSSSK	DMSSTG	DMSSTT	DMSSVN	DMSSVT	DMSSYN	DMSTIO	DMSTMA	DMSTPD
		DMSTPE	DMSTRK	DMSTYP	DMSUPD	DMSVIB	DMSVIP	DMSVPD	DMSVSR	DMSXCP	DMSZAP		

Label	Count	References											
R1	006064	DMSABN	DMSACC	DMSACF	DMSACM	DMSALU	DMSAMS	DMSARE	DMSARN	DMSARX	DMSASM	DMSASN	DMSAUD
		DMSBAB	DMSBOP	DMSBRD	DMSBSC	DMSBTB	DMSBTP	DMSBWR	DMSCAT	DMSCIO	DMSCIT	DMSCLS	DMSCPF
		DMSCRD	DMSCWR	DMSCWT	DMSDBD	DMSDBG	DMSDIO	DMSDLB	DMSDMP	DMSDOS	DMSDSK	DMSDSL	DMSEDC
		DMSEDI	DMSEDX	DMSERS	DMSEXC	DMSEXT	DMSFCH	DMSFET	DMSFLD	DMSFNS	DMSFOR	DMSGIO	DMSGLB
		DMSGND	DMSGRN	DMSHDI	DMSHDS	DMSINA	DMSINI	DMSINM	DMSINS	DMSINT	DMSIOW	DMSITE	DMSITP
		DMSITS	DMSLAD	DMSLAF	DMSLBM	DMSLBT	DMSLDR	DMSLDS	DMSLFS	DMSLGT	DMSLIB	DMSLIO	DMSLKD
		DMSLLU	DMSLOA	DMSLSB	DMSLST	DMSLSY	DMSMDP	DMSMOD	DMSMVE	DMSNCP	DMSOLD	DMSOPL	DMSOPT
		DMSOR1	DMSOR2	DMSOR3	DMSOVR	DMSOVS	DMSPIO	DMSPNT	DMSPRT	DMSPRV	DMSPUN	DMSQRY	DMSRDC
		DMSRNE	DMSRNM	DMSROS	DMSRRV	DMSSAB	DMSSBD	DMSSBS	DMSSCN	DMSSCR	DMSSCT	DMSSEB	DMSSET
		DMSSMN	DMSSOP	DMSSQS	DMSSRT	DMSSRV	DMSSSK	DMSSTG	DMSSTT	DMSSVN	DMSSVT	DMSSYN	DMSTIO
		DMSTMA	DMSTPD	DMSTPE	DMSTRK	DMSTYP	DMSUPD	DMSVIB	DMSVIP	DMSVPD	DMSVSR	DMSXCP	DMSZAP
R10	001736	DMSACC	DMSACF	DMSACM	DMSALU	DMSAMS	DMSARE	DMSARN	DMSARX	DMSASM	DMSASN	DMSAUD	DMSBAB
		DMSBOP	DMSBRD	DMSBSC	DMSBTP	DMSBWR	DMSCIO	DMSCLS	DMSCWR	DMSCWT	DMSDBD	DMSDBG	DMSDIO
		DMSDLB	DMSDOS	DMSDSL	DMSEDC	DMSEDI	DMSEDX	DMSERS	DMSEXC	DMSEXT	DMSFCH	DMSFLD	DMSFNS
		DMSFOR	DMSGIO	DMSGRN	DMSHDI	DMSHDS	DMSINI	DMSINM	DMSINS	DMSINT	DMSIOW	DMSITE	DMSITI
		DMSITP	DMSITS	DMSLAD	DMSLBM	DMSLBT	DMSLDR	DMSLDS	DMSLFS	DMSLGT	DMSLIO	DMSLKD	DMSLLU
		DMSLSB	DMSLST	DMSMOD	DMSMVE	DMSNCP	DMSOLD	DMSOPT	DMSPIO	DMSPRT	DMSPRV	DMSPUN	DMSQRY
		DMSRDC	DMSRNE	DMSRNM	DMSROS	DMSRRV	DMSSAB	DMSSBD	DMSSET	DMSSMN	DMSSOP	DMSSQS	DMSSRV
		DMSSTG	DMSSTT	DMSSVN	DMSSVT	DMSTMA	DMSTPD	DMSTPE	DMSTRK	DMSTYP	DMSUPD	DMSVIP	DMSXCP
		DMSZAP											
R11	000702	DMSACC	DMSACF	DMSACM	DMSALU	DMSAMS	DMSARN	DMSARX	DMSASM	DMSASN	DMSAUD	DMSBOP	DMSBRD
		DMSBSC	DMSBTP	DMSBWR	DMSCIO	DMSCLS	DMSCRD	DMSCWR	DMSCWT	DMSDBD	DMSDIO	DMSDLB	DMSDOS
		DMSERS	DMSEXC	DMSFCH	DMSFLD	DMSFNS	DMSFOR	DMSGND	DMSGRN	DMSINI	DMSINS	DMSINT	DMSIOW
		DMSITE	DMSITI	DMSITP	DMSITS	DMSLAF	DMSLBM	DMSLBT	DMSLDR	DMSLDS	DMSLFS	DMSLIB	DMSLIO
		DMSLKD	DMSLLU	DMSLSB	DMSLST	DMSMOD	DMSNCP	DMSOLD	DMSOPT	DMSPIO	DMSPNT	DMSPRT	DMSPUN
		DMSQRY	DMSRDC	DMSRNM	DMSROS	DMSRRV	DMSSAB	DMSSBD	DMSSBS	DMSSCR	DMSSCT	DMSSEB	DMSSET
		DMSSOP	DMSSQS	DMSSVT	DMSSYN	DMSTIO	DMSTMA	DMSTPD	DMSTPE	DMSTRK	DMSUPD	DMSVIP	DMSVPD
		DMSXCP	DMSZAP										
R12	000598	DMSABN	DMSACC	DMSACF	DMSACM	DMSALU	DMSAMS	DMSARE	DMSARN	DMSARX	DMSASM	DMSASN	DMSAUD
		DMSBAB	DMSBOP	DMSBRD	DMSBSC	DMSBTB	DMSBTP	DMSBWR	DMSCAT	DMSCIO	DMSCIT	DMSCLS	DMSCPF
		DMSCRD	DMSCWR	DMSCWT	DMSDIO	DMSDLB	DMSDMP	DMSDOS	DMSDSL	DMSEDX	DMSERS	DMSEXC	DMSFCH
		DMSFET	DMSFLD	DMSFNS	DMSFOR	DMSGLB	DMSGND	DMSGRN	DMSHDI	DMSHDS	DMSINI	DMSINS	DMSINT
		DMSITE	DMSITI	DMSITP	DMSITS	DMSLAD	DMSLAF	DMSLBT	DMSLDR	DMSLDS	DMSLFS	DMSLGT	DMSLIB
		DMSLKD	DMSLLU	DMSLOA	DMSLSB	DMSLST	DMSMOD	DMSMVE	DMSNCP	DMSOLD	DMSOPL	DMSOPT	DMSOR1
		DMSOR2	DMSOR3	DMSOVR	DMSOVS	DMSPIO	DMSPRT	DMSPRV	DMSPUN	DMSQRY	DMSRNE	DMSRNM	DMSROS
		DMSRRV	DMSSAB	DMSSBD	DMSSBS	DMSSCN	DMSSCR	DMSSCT	DMSSET	DMSSMN	DMSSOP	DMSSQS	DMSSRT
		DMSSRV	DMSSSK	DMSSTG	DMSSTT	DMSSVN	DMSSVT	DMSSYN	DMSTIO	DMSTMA	DMSTPD	DMSTPE	DMSTRK
		DMSUPD	DMSVIB	DMSVIP	DMSVPD	DMSVSR	DMSXCP	DMSZAP					
R13	000694	DMSABN	DMSACC	DMSACF	DMSACM	DMSALU	DMSAMS	DMSARN	DMSARX	DMSASM	DMSASN	DMSAUD	DMSBAB
		DMSBRD	DMSBSC	DMSBTP	DMSBWR	DMSCIO	DMSCIT	DMSCLS	DMSCRD	DMSCWR	DMSDBG	DMSDIO	DMSDLB
		DMSDOS	DMSDSK	DMSEDC	DMSEDI	DMSEDX	DMSERS	DMSEXC	DMSFCH	DMSFLD	DMSFNS	DMSFOR	DMSGIO
		DMSGLB	DMSGRN	DMSHDI	DMSHDS	DMSINI	DMSINS	DMSINT	DMSITE	DMSITI	DMSITP	DMSITS	DMSLAD
		DMSLAF	DMSLBT	DMSLDR	DMSLDS	DMSLFS	DMSLGT	DMSLIB	DMSLIO	DMSLSB	DMSLST	DMSMOD	DMSMVE
		DMSNCP	DMSOLD	DMSOVS	DMSPIO	DMSPRT	DMSPUN	DMSQRY	DMSRNE	DMSRNM	DMSSAB	DMSSBS	DMSSCR
		DMSSCT	DMSSEB	DMSSMN	DMSSOP	DMSSQS	DMSSTG	DMSSTT	DMSSVN	DMSSVT	DMSTIO	DMSTPE	DMSTRK
		DMSUPD	DMSVIP	DMSVSR	DMSXCP	DMSZAP							

Label	Count	References											
R14	002863	DMSABN	DMSACC	DMSACF	DMSACM	DMSALU	DMSAMS	DMSARE	DMSARN	DMSARX	DMSASM	DMSASN	DMSAUD
		DMSBAB	DMSBOP	DMSBRD	DMSBSC	DMSBTB	DMSBTP	DMSBWR	DMSCAT	DMSCIO	DMSCIT	DMSCLS	DMSCPF
		DMSCRD	DMSCWR	DMSCWT	DMSDBD	DMSDBG	DMSDIO	DMSDLB	DMSDOS	DMSDSK	DMSDSL	DMSEDC	DMSEDI
		DMSEDX	DMSERS	DMSEXC	DMSEXT	DMSFCH	DMSFET	DMSFLD	DMSFNS	DMSFOR	DMSGIO	DMSGLB	DMSGND
		DMSGRN	DMSHDI	DMSHDS	DMSINA	DMSINI	DMSINM	DMSINS	DMSINT	DMSIOW	DMSITE	DMSITI	DMSITP
		DMSITS	DMSLAD	DMSLAF	DMSLBM	DMSLBT	DMSLDR	DMSLDS	DMSLFS	DMSLGT	DMSLIB	DMSLIO	DMSLKD
		DMSLLU	DMSLOA	DMSLSB	DMSLST	DMSLSY	DMSMDP	DMSMOD	DMSMVE	DMSNCP	DMSOLD	DMSOPT	DMSOR3
		DMSOVR	DMSOVS	DMSPIO	DMSPNT	DMSPRT	DMSPRV	DMSPUN	DMSQRY	DMSRDC	DMSRNE	DMSRNM	DMSROS
		DMSRRV	DMSSAB	DMSSBD	DMSSBS	DMSSCN	DMSSCR	DMSSCT	DMSSEB	DMSSET	DMSSMN	DMSSOP	DMSSQS
		DMSSRT	DMSSRV	DMSSSK	DMSSTG	DMSSTT	DMSSVN	DMSSVT	DMSSYN	DMSTIO	DMSTMA	DMSTPD	DMSTPE
		DMSTRK	DMSTYP	DMSUPD	DMSVIB	DMSVIP	DMSVPD	DMSVSR	DMSXCP	DMSZAP			
R15	004744	DMSABN	DMSACC	DMSACF	DMSACM	DMSALU	DMSAMS	DMSARE	DMSARN	DMSARX	DMSASM	DMSASN	DMSAUD
		DMSBAB	DMSBOP	DMSBRD	DMSBSC	DMSBTB	DMSBTP	DMSBWR	DMSCAT	DMSCIO	DMSCIT	DMSCLS	DMSCPF
		DMSCRD	DMSCWR	DMSCWT	DMSDBD	DMSDBG	DMSDIO	DMSDLB	DMSDOS	DMSDSK	DMSDSL	DMSEDC	DMSEDI
		DMSEDX	DMSERS	DMSEXC	DMSEXT	DMSFCH	DMSFET	DMSFLD	DMSFNS	DMSFOR	DMSGIO	DMSGLB	DMSGND
		DMSGRN	DMSHDI	DMSHDS	DMSINA	DMSINI	DMSINM	DMSINS	DMSINT	DMSIOW	DMSITE	DMSITI	DMSITP
		DMSITS	DMSLAD	DMSLAF	DMSLBM	DMSLBT	DMSLDR	DMSLDS	DMSLFS	DMSLGT	DMSLIB	DMSLIO	DMSLKD
		DMSLLU	DMSLOA	DMSLSB	DMSLST	DMSLSY	DMSMDP	DMSMOD	DMSMVE	DMSNCP	DMSOLD	DMSOPL	DMSOPT
		DMSOR1	DMSOVR	DMSOVS	DMSPIO	DMSPNT	DMSPRT	DMSPRV	DMSPUN	DMSQRY	DMSRDC	DMSRNE	DMSRNM
		DMSROS	DMSRRV	DMSSAB	DMSSBD	DMSSBS	DMSSCN	DMSSCR	DMSSCT	DMSSEB	DMSSET	DMSSMN	DMSSOP
		DMSSQS	DMSSRT	DMSSRV	DMSSSK	DMSSTG	DMSSTT	DMSSVN	DMSSVT	DMSSYN	DMSTIO	DMSTMA	DMSTPD
		DMSTPE	DMSTRK	DMSTYP	DMSUPD	DMSVIB	DMSVIP	DMSVPD	DMSVSR	DMSXCP	DMSZAP		
R2	003449	DMSACC	DMSACF	DMSACM	DMSALU	DMSAMS	DMSARE	DMSARN	DMSARX	DMSASM	DMSASN	DMSAUD	DMSBAB
		DMSBOP	DMSBRD	DMSBSC	DMSBTB	DMSBTP	DMSBWR	DMSCAT	DMSCIO	DMSCIT	DMSCLS	DMSCPF	DMSCRD
		DMSCWR	DMSDBD	DMSDBG	DMSDIO	DMSDLB	DMSDMP	DMSDOS	DMSDSK	DMSDSL	DMSEDC	DMSEDI	DMSEDX
		DMSERS	DMSEXC	DMSEXT	DMSFCH	DMSFET	DMSFLD	DMSFNS	DMSFOR	DMSGIO	DMSGLB	DMSGND	DMSGRN
		DMSHDI	DMSHDS	DMSINA	DMSINI	DMSINM	DMSINS	DMSINT	DMSIOW	DMSITE	DMSITP	DMSITS	DMSLAD
		DMSLAF	DMSLBM	DMSLBT	DMSLDR	DMSLDS	DMSLFS	DMSLIO	DMSLKD	DMSLLU	DMSLOA	DMSLSB	DMSLST
		DMSMDP	DMSMOD	DMSMVE	DMSNCP	DMSOLD	DMSOPL	DMSOPT	DMSOR1	DMSPIO	DMSPNT	DMSPRT	DMSPRV
		DMSPUN	DMSQRY	DMSRDC	DMSRNE	DMSRNM	DMSROS	DMSRRV	DMSSAB	DMSSBD	DMSSBS	DMSSCN	DMSSCR
		DMSSCT	DMSSEB	DMSSET	DMSSMN	DMSSOP	DMSSQS	DMSSRT	DMSSRV	DMSSSK	DMSSTG	DMSSTT	DMSSVN
		DMSSVT	DMSSYN	DMSTMA	DMSTPD	DMSTPE	DMSTRK	DMSTYP	DMSUPD	DMSVIB	DMSVIP	DMSVPD	DMSVSR
		DMSXCP	DMSZAP										
R3	003494	DMSABN	DMSACC	DMSACF	DMSACM	DMSALU	DMSAMS	DMSARE	DMSARN	DMSARX	DMSASM	DMSASN	DMSAUD
		DMSBAB	DMSBOP	DMSBRD	DMSBSC	DMSBTB	DMSBTP	DMSBWR	DMSCAT	DMSCIO	DMSCIT	DMSCLS	DMSCPF
		DMSCRD	DMSCWR	DMSDBD	DMSDBG	DMSDLB	DMSDMP	DMSDOS	DMSDSK	DMSDSL	DMSEDC	DMSEDI	DMSEDX
		DMSERS	DMSEXC	DMSEXT	DMSFCH	DMSFET	DMSFLD	DMSFOR	DMSGLB	DMSGND	DMSGRN	DMSHDI	DMSHDS
		DMSINA	DMSINI	DMSINM	DMSINS	DMSINT	DMSITE	DMSITI	DMSITP	DMSITS	DMSLAD	DMSLAF	DMSLBM
		DMSLBT	DMSLDR	DMSLDS	DMSLFS	DMSLGT	DMSLIO	DMSLKD	DMSLLU	DMSLSB	DMSLST	DMSMDP	DMSMOD
		DMSMVE	DMSNCP	DMSOLD	DMSOPL	DMSOVR	DMSOVS	DMSPIO	DMSPRT	DMSPRV	DMSPUN	DMSQRY	DMSRDC
		DMSRNE	DMSRNM	DMSROS	DMSRRV	DMSSAB	DMSSBD	DMSSBS	DMSSCN	DMSSCR	DMSSCT	DMSSEB	DMSSET
		DMSSMN	DMSSOP	DMSSQS	DMSSRT	DMSSRV	DMSSSK	DMSSTG	DMSSTT	DMSSVN	DMSSVT	DMSSYN	DMSTMA
		DMSTPD	DMSTPE	DMSTRK	DMSTYP	DMSUPD	DMSVIB	DMSVIP	DMSVPD	DMSVSR	DMSXCP	DMSZAP	

Label	Count	References											
R4	002780	DMSABN	DMSACC	DMSACF	DMSACM	DMSALU	DMSAMS	DMSARE	DMSARN	DMSARX	DMSASM	DMSASN	DMSAUD
		DMSBAB	DMSBOP	DMSBRD	DMSBSC	DMSBTB	DMSBTP	DMSBWR	DMSCAT	DMSCIO	DMSCIT	DMSCLS	DMSCPF
		DMSCRD	DMSCWR	DMSDBD	DMSDBG	DMSDIO	DMSDLB	DMSDMP	DMSDOS	DMSDSK	DMSDSL	DMSEDC	DMSEDI
		DMSEDX	DMSERS	DMSEXC	DMSEXT	DMSFCH	DMSFET	DMSFLD	DMSFOR	DMSGIO	DMSGLB	DMSGND	DMSGRN
		DMSHDI	DMSHDS	DMSINA	DMSINI	DMSINM	DMSINS	DMSINT	DMSIOW	DMSITI	DMSITP	DMSITS	DMSLAD
		DMSLAF	DMSLBM	DMSLBT	DMSLDR	DMSLDS	DMSLFS	DMSLGT	DMSLIO	DMSLKD	DMSLLU	DMSLSB	DMSLST
		DMSMDP	DMSMOD	DMSMVE	DMSNCP	DMSOLD	DMSOPL	DMSOVR	DMSOVS	DMSPIO	DMSPNT	DMSPRT	DMSPUN
		DMSQRY	DMSRDC	DMSRNE	DMSRNM	DMSROS	DMSRRV	DMSSAB	DMSSBD	DMSSBS	DMSSCN	DMSSCR	DMSSCT
		DMSSET	DMSSMN	DMSSOP	DMSSQS	DMSSRT	DMSSRV	DMSSSK	DMSSTG	DMSSTT	DMSSVN	DMSSVT	DMSSYN
		DMSTMA	DMSTPD	DMSTPE	DMSTRK	DMSTYP	DMSUPD	DMSVIP	DMSVPD	DMSVSR	DMSXCP	DMSZAP	
R5	002930	DMSABN	DMSACC	DMSACF	DMSACM	DMSALU	DMSAMS	DMSARE	DMSARN	DMSARX	DMSASM	DMSASN	DMSAUD
		DMSBAB	DMSBOP	DMSBRD	DMSBSC	DMSBTB	DMSBTP	DMSBWR	DMSCIO	DMSCIT	DMSCLS	DMSCPF	DMSCRD
		DMSCWR	DMSDBD	DMSDBG	DMSDIO	DMSDLB	DMSDMP	DMSDOS	DMSDSK	DMSDSL	DMSEDC	DMSEDI	DMSEDX
		DMSERS	DMSEXC	DMSEXT	DMSFCH	DMSFET	DMSFLD	DMSFNS	DMSFOR	DMSGIO	DMSGLB	DMSGND	DMSGRN
		DMSHDI	DMSHDS	DMSINA	DMSINI	DMSINM	DMSINS	DMSINT	DMSIOW	DMSITI	DMSITP	DMSITS	DMSLAD
		DMSLAF	DMSLEM	DMSLBT	DMSLDR	DMSLDS	DMSLFS	DMSLGT	DMSLIB	DMSLKD	DMSLLU	DMSLSB	DMSLST
		DMSMOD	DMSMVE	DMSNCP	DMSCLD	DMSOPL	DMSOR1	DMSOVR	DMSOVS	DMSPIO	DMSPNT	DMSPRT	DMSPUN
		DMSQRY	DMSRDC	DMSRNE	DMSRNM	DMSROS	DMSRRV	DMSSAB	DMSSBD	DMSSBS	DMSSCN	DMSSCR	DMSSCT
		DMSSET	DMSSOP	DMSSQS	DMSSRT	DMSSRV	DMSSSK	DMSSTG	DMSSTT	DMSSVN	DMSSVT	DMSSYN	DMSTMA
		DMSTPD	DMSTPE	DMSTRK	DMSTYP	DMSUPD	DMSVIB	DMSVIP	DMSVPD	DMSVSR	DMSXCP	DMSZAP	
R6	002486	DMSABN	DMSACC	DMSACF	DMSACM	DMSALU	DMSAMS	DMSARE	DMSARN	DMSARX	DMSASM	DMSASN	DMSAUD
		DMSBAB	DMSBOP	DMSBRD	DMSBSC	DMSBTP	DMSBWR	DMSCIO	DMSCIT	DMSCLS	DMSCPF	DMSCRD	DMSCWR
		DMSDBD	DMSDBG	DMSDIO	DMSDLE	DMSDMP	DMSDOS	DMSDSK	DMSEDC	DMSEDI	DMSEDX	DMSERS	DMSEXC
		DMSEXT	DMSFCH	DMSFET	DMSFLD	DMSFNS	DMSFOR	DMSGND	DMSGRN	DMSHDI	DMSHDS	DMSINA	DMSINI
		DMSINS	DMSINT	DMSIOW	DMSITI	DMSITP	DMSITS	DMSLAD	DMSLBM	DMSLBT	DMSLDR	DMSLDS	DMSLFS
		DMSLGT	DMSLKD	DMSLLU	DMSLOA	DMSLSB	DMSLST	DMSMOD	DMSMVE	DMSNCP	DMSOLD	DMSOPL	DMSOR1
		DMSOVR	DMSOVS	DMSPIO	DMSPNT	DMSPRT	DMSPUN	DMSQRY	DMSRDC	DMSRNE	DMSRNM	DMSROS	DMSRRV
		DMSSAB	DMSSBD	DMSSBS	DMSSCN	DMSSCR	DMSSCT	DMSSET	DMSSMN	DMSSOP	DMSSQS	DMSSRT	DMSSSK
		DMSSTG	DMSSTT	DMSSVN	DMSSVT	DMSSYN	DMSTMA	DMSTPD	DMSTPE	DMSTRK	DMSTYP	DMSUPD	DMSVIP
		DMSVPD	DMSVSR	DMSXCP	DMSZAP								
R7	002318	DMSABN	DMSACC	DMSACF	DMSACM	DMSALU	DMSAMS	DMSARE	DMSARN	DMSARX	DMSASM	DMSASN	DMSAUD
		DMSBOP	DMSBRD	DMSBSC	DMSBTP	DMSBWR	DMSCIO	DMSCIT	DMSCLS	DMSCPF	DMSCWR	DMSDBD	DMSDBG
		DMSDIO	DMSDLE	DMSDMP	DMSDOS	DMSDSK	DMSEDC	DMSEDI	DMSEDX	DMSERS	DMSEXC	DMSEXT	DMSFCH
		DMSFET	DMSFLD	DMSFNS	DMSFOR	DMSGLB	DMSGRN	DMSHDI	DMSHDS	DMSINA	DMSINI	DMSINS	DMSINT
		DMSIOW	DMSITE	DMSITI	DMSITP	DMSITS	DMSLAD	DMSLBM	DMSLBT	DMSLDR	DMSLDS	DMSLFS	DMSLGT
		DMSLIB	DMSLKD	DMSLLU	DMSLSB	DMSLST	DMSMOD	DMSMVE	DMSOLD	DMSOPL	DMSOVR	DMSOVS	DMSPIO
		DMSPRT	DMSPUN	DMSQRY	DMSRDC	DMSRNE	DMSRNM	DMSROS	DMSRRV	DMSSAB	DMSSBD	DMSSCN	DMSSCR
		DMSSCT	DMSSET	DMSSMN	DMSSOP	DMSSQS	DMSSTG	DMSSVT	DMSSYN	DMSTMA	DMSTPD	DMSTPE	DMSTRK
		DMSTYP	DMSUPD	DMSVIP	DMSVPD	DMSVSR	DMSXCP	DMSZAP					

Label	Count	References											
R8	001983	DMSABN	DMSACC	DMSACF	DMSACM	DMSALU	DMSAMS	DMSARE	DMSARN	DMSARX	DMSASM	DMSASN	DMSAUD
		DMSBAB	DMSBOP	DMSBRD	DMSBSC	DMSBTB	DMSBTP	DMSBWR	DMSCIO	DMSCIT	DMSCLS	DMSCPF	DMSCRD
		DMSCWR	DMSDBD	DMSDBG	DMSDIO	DMSDLB	DMSDOS	DMSDSK	DMSDSL	DMSEDC	DMSEDI	DMSEDX	DMSERS
		DMSEXC	DMSEXT	DMSFCH	DMSFLD	DMSFNS	DMSFOR	DMSGLB	DMSGRN	DMSHDI	DMSHDS	DMSINA	DMSINI
		DMSINM	DMSIOW	DMSITI	DMSITP	DMSITS	DMSLAD	DMSLBM	DMSLBT	DMSLDR	DMSLDS	DMSLFS	DMSLGT
		DMSLLU	DMSLSB	DMSLST	DMSMOD	DMSMVE	DMSNCP	DMSOLD	DMSOPL	DMSOVR	DMSOVS	DMSPIO	DMSPRT
		DMSPUN	DMSQRY	DMSRDC	DMSRNM	DMSROS	DMSRRV	DMSSAB	DMSSBD	DMSSBS	DMSSCN	DMSSCT	DMSSEB
		DMSSET	DMSSMN	DMSSOP	DMSSSK	DMSSTG	DMSSVN	DMSSVT	DMSSYN	DMSTMA	DMSTPD	DMSTPE	DMSTRK
		DMSTYP	DMSUPD	DMSVIP	DMSVSR	DMSXCP	DMSZAP						
R9	001779	DMSACC	DMSACF	DMSACM	DMSALU	DMSAMS	DMSARE	DMSARN	DMSARX	DMSASM	DMSASN	DMSAUD	DMSBAB
		DMSBOP	DMSBRD	DMSBSC	DMSBTP	DMSBWR	DMSCIT	DMSCLS	DMSCRD	DMSCWT	DMSDBD	DMSDBG	DMSDIO
		DMSDLB	DMSDOS	DMSDSK	DMSEDC	DMSEDI	DMSEDX	DMSERS	DMSEXC	DMSEXT	DMSFCH	DMSFLD	DMSFNS
		DMSFOR	DMSGND	DMSGRN	DMSHDI	DMSHDS	DMSINA	DMSINI	DMSINS	DMSINT	DMSIOW	DMSITI	DMSITP
		DMSITS	DMSLAD	DMSLBM	DMSLBT	DMSLDR	DMSLDS	DMSLFS	DMSLGT	DMSLKD	DMSLSB	DMSLST	DMSMOD
		DMSMVE	DMSNCP	DMSOLD	DMSOPL	DMSPIO	DMSPRT	DMSPUN	DMSQRY	DMSRDC	DMSRNM	DMSROS	DMSRRV
		DMSSAB	DMSSBD	DMSSCR	DMSSCT	DMSSET	DMSSMN	DMSSOP	DMSSRV	DMSSSK	DMSSTG	DMSSTT	DMSSVT
		DMSTMA	DMSTPD	DMSTPE	DMSTRK	DMSTYP	DMSUPD	DMSVIP	DMSXCP	DMSZAP			
SAVCNT	000004	DMSEDI	DMSSCR										
SAVCWD	000021	DMSEDI											
SAVEADT	000002	DMSDIO											
SAVEAR	000010	DMSEDC	DMSSCR										
SAVER1	000042	DMSSOP											
SAVER14	000013	DMSSCT	DMSSEB										
SAVER15	000002	DMSSOP											
SAVEXT	000002	DMSITE											
SAVE1	000020	DMSDBD	DMSDBG										
SAVE2	000003	DMSDBG											
SAV67	000006	DMSLDR	DMSOLD										
SCAW	000003	DMSITE											
SCBPTR	000014	DMSITP	DMSSAB	DMSSLN	DMSSTG	DMSSVT							
SCBSAV12	000004	DMSSAB											
SCBWORK	000008	DMSSAB	DMSSTG										
SCLNO	000002	DMSSCR											
SCRBUFAD	000002	DMSEDX	DMSSCR										
SCRFLGS	000033	DMSEDI	DMSSCR										
SCRFLG2	000015	DMSEDI	DMSSCR										
SDISK	000004	DMSINI											
SEARCH	000035	DMSINI											
SEEK	000036	DMSINI											
SEEKADR	000001	DMSDIO											
SENCCW	000002	DMSDIO											
SENSB	000002	DMSDIO	DMSFNS										
SEQNAME	000004	DMSEDI	DMSEDX										

Label	Count	References											
SERSAV	000002	DMSEDI											
SERTSEQ	000003	DMSEDI											
SERTSW	000003	DMSEDI											
SETLIB	000002	DMSLIB											
SETSEC	000002	DMSINI											
SIGNAL	000053	DMSACM	DMSEDI	DMSERS									
SILI	000205	DMSINI	DMSINS	DMSTIO									
SIZE	000022	DMSFRE											
SKEY	000003	DMSFRE											
SOB1	000002	DMSOPT	DMSSET										
SPARES	000015	DMSEDI	DMSEDX										
SPEC	000189	DMSLDR	DMSLGT	DMSLIB	DMSOLD								
SPIESAV	000002	DMSINT											
SSAVE	000056	DMSABN	DMSBSC	DMSDBG	DMSDLB	DMSERR	DMSFLD	DMSFRE	DMSITP	DMSITS	DMSLDR	DMSOVS	DMSSMN
		DMSSTG											
SSAVENXT	000004	DMSITS											
SSAVEPRV	000008	DMSITS											
SSAVESZ	000003	DMSITS											
STACKAT	000001	DMSEDI											
STACKATL	000005	DMSEDI											
STAEBIT	000003	DMSSAB											
STAESAV	000002	DMSINT											
STAIBIT	000002	DMSSAB											
STARS	000001	DMSINT											
START	000022	DMSLDR	DMSLSB										
STATEFST	000022	DMSALU	DMSBRD	DMSERS	DMSFNS	DMSINT	DMSPUN	DMSRNM	DMSSTT				
STATER0	000002	DMSBRD	DMSSTT										
STATER1	000005	DMSDSK	DMSERS										
STIMEXIT	000009	DMSITE	DMSSTG	DMSSVN	DMSSVT								
STOPAT	000002	DMSDBG											
STRTADDR	000030	DMSFET	DMSITS	DMSLDR	DMSLOA	DMSLSB	DMSMOD	DMSOLD	DMSSLN				
STRTNO	000005	DMSEDI											
SUBACT	000003	DMSEDX	DMSINT	DMSSLN									
SUBFLAG	000024	DMSABN	DMSEDX	DMSFNS	DMSINT	DMSMOD	DMSSLN						
SUBINIT	000001	DMSFNS											
SUBSECT	000004	DMSABN	DMSINM	DMSINT									
SVCAB	000008	DMSFRE											
SVCOPSW	000020	DMSITS											
SVCOUNT	000003	DMSOVS											
SVCSECT	000010	DMSCIT	DMSFRE	DMSHDS	DMSINT	DMSOVR	DMSOVS	DMSSLN					
SVEARA	000008	DMSBAB	DMSDOS	DMSITP									
SVEPSW	000008	DMSBAB	DMSDOS	DMSITP									

Label	Count	References
SVEPSW2	000009	DMSBAB DMSDOS DMSITP
SVEROF	000005	DMSBAB DMSDOS
SVER00	000020	DMSBAB DMSDOS DMSITP
SVER01	000002	DMSBAB
SVER09	000011	DMSBAB DMSDOS DMSITP
SVLAD	000002	DMSLAD
SVLADW	000001	DMSLAD
SVLFS	000002	DMSLFS
SWTCH	000001	DMSACM
SWTCHSAV	000002	DMSINT
SYMTABLE	000003	DMSDBG
SYMTBG	000004	DMSDBG
SYSADDR	000003	DMSINI
SYSCODE	000005	DMSFRE DMSSET
SYSCOM	000018	DMSBAB DMSBOP DMSDOS DMSFET DMSITP DMSSTG
SYSLINE	000001	DMSDLK
SYSNAME	000006	DMSBTP DMSINS
SYSNAMES	000022	DMSAMS DMSBOP DMSBTP DMSDOS DMSEDX DMSEXC DMSINS DMSINT DMSITS DMSQRY DMSSET DMSVIB DMSVSR
SYSREF	000004	DMSINS DMSLOA DMSSET
SYSTEMID	000005	DMSINI DMSINS
SYSUT1	000024	DMSLDR DMSOLD
TABLIN	000016	DMSEDI DMSSCR
TABS	000017	DMSEDI DMSEDX
TAIEIAD	000002	DMSCIT
TAIEMSGL	000001	DMSCIT
TAIERSAV	000002	DMSCIT
TAPEBUFF	000001	DMSSEB
TAPECOUT	000002	DMSSEB
TAPEDEV	000003	DMSSBS DMSSEB DMSSOP
TAPELIST	000003	DMSSBS DMSSEB DMSSOP
TAPEMASK	000003	DMSSBS DMSSEB DMSSOP
TAPEOPER	000007	DMSSBS DMSSEB DMSSOP
TAPESIZE	000002	DMSSEB
TAPE1	000002	DMSASN
TAPE4	000002	DMSASN
TAXEADDR	000008	DMSCIT DMSSTG DMSSVT
TAXEDEF	000001	DMSSVT
TAXEEXIT	000002	DMSCIT DMSSVT
TAXEEXTS	000001	DMSCIT
TAXEFREQ	000004	DMSCIT
TAXEICL	000002	DMSCIT

Label	Count	References									
TAXEIOWS	000002	DMSCIT									
TAXELNK	000004	DMSCIT	DMSSVT								
TAXERTNA	000002	DMSCIT									
TAXESTAT	000003	DMSCIT									
TAXETAIE	000002	DMSCIT									
TAXETSCF	000002	DMSCIT									
TBENT	000024	DMSACM	DMSBTB	DMSFET	DMSGND	DMSLDR	DMSLOA	DMSMDP	DMSMOD	DMSOLD	DMSSLN
TBLCT	000017	DMSLDR	DMSLIB	DMSOLD							
TBLLNGTH	000005	DMSSBD	DMSSVT								
TBLREF	000016	DMSLDR	DMSLIB	DMSOLD							
TCODE	000001	DMSFRE									
TEMPBYTE	000003	DMSSVT									
TEMPST	000008	DMSLDR	DMSOLD								
TEMPTAB	000002	DMSEDI									
TIC	000053	DMSINI									
TIMBUF	000013	DMSINM									
TIMCCW	000005	DMSITE	DMSQRY	DMSSET							
TIMCHAR	000012	DMSINS	DMSINT	DMSIOW	DMSITE	DMSQRY	DMSSET	DMSSVN			
TIMER	000015	DMSINS	DMSINT	DMSIOW	DMSITE	DMSSET	DMSSVN	DMSSVT			
TIMINIT	000010	DMSINS	DMSINT	DMSIOW	DMSITE	DMSSET	DMSSVN				
TIN	000007	DMSEDI	DMSEDX								
TMPLOC	000008	DMSLDR	DMSLSB	DMSOLD							
TOOBIG	000003	DMSDIO									
TOUT	000008	DMSEDI									
TPFERT	000003	DMSITS									
TPFNS	000009	DMSITS									
TPFR01	000002	DMSITS									
TPFSVO	000005	DMSITS	DMSOVS								
TPFUSR	000011	DMSDBG	DMSITP	DMSITS	DMSLDR						
TRKLSAVE	000004	DMSTQQ									
TRNCODE	000001	DMSFRE									
TRUNCOL	000015	DMSEDI	DMSEDX	DMSSCR							
TSOATCNL	000018	DMSCIT	DMSCRD	DMSITE	DMSITI	DMSITS	DMSSEB	DMSSVN			
TSOBLKS	000001	DMSSET									
TSOFLAGS	000019	DMSCIT	DMSCRD	DMSITE	DMSITI	DMSITS	DMSSEB	DMSSVN			
TSYM	000005	DMSDBG									
TVERCOL1	000002	DMSEDI									
TVERCOL2	000001	DMSEDI									
TWITCH	000087	DMSEDI	DMSEDX	DMSSCR							
TXTDIRC	000008	DMSGLB	DMSLDR	DMSLGT	DMSLIB	DMSOLD					
TXTLIBS	000004	DMSGLB	DMSLGT	DMSLIB	DMSQRY						
TYPE	000092	DMSLGT	DMSLIB	DMSLIO	DMSLOA	DMSLSB					
TYPEAD	000001	DMSLIO									
TYPFLAG	000034	DMSDBG	DMSITP	DMSITS	DMSLDR	DMSOVS					

Label	Count	References
TYPFLG	000004	DMSEDI
TYPLIN	000040	DMSLIO
TYPLIST	000007	DMSITE
UE	000001	DMSCIT
UFDBUSY	000030	DMSABN DMSACC DMSACF DMSACM DMSAUD DMSBTP DMSBWR DMSDIO DMSDSK DMSERS DMSFNS DMSITE DMSITI DMSITP DMSITS DMSRNM DMSTPE
UND	000017	DMSROS DMSSBS DMSSEB DMSSOP DMSSQS
UNPACK	000010	DMSLIO
UPBIT	000005	DMSACM DMSAUD DMSDSK
UPSI	000004	DMSSET
UPTMID	000002	DMSSET
UPTSWS	000002	DMSSET
USARCODE	000002	DMSFRE
USAVEPTR	000023	DMSITS DMSSTG
USAVESZ	000002	DMSITS
USERCODE	000004	DMSFRE DMSSET
USERKEY	000010	DMSFRE DMSSET
UTILFLAG	000017	DMSSCR
VAR	000027	DMSROS DMSSBD DMSSBS DMSSEB DMSSOP DMSSQS DMSSVT
VERCOL1	000006	DMSEDI DMSEDX DMSSCR
VERCOL2	000003	DMSEDI DMSEDX
VERLEN	000006	DMSEDI DMSEDX DMSSCR
VMSIZE	000037	DMSAMS DMSBOP DMSBRD DMSBWR DMSDBG DMSDOS DMSFRE DMSHDI DMSHDS DMSINS DMSLDR DMSOVS DMSSET DMSSSK DMSSVT DMSVIB
VSTRANGE	000001	DMSITI
WAIT	000028	DMSCIT DMSINI DMSINS DMSITI
WAITLIST	000002	DMSDBG DMSSVT
WAITLST	000003	DMSCRD DMSCWR DMSCWT
WAITRD	000004	DMSDBG
WAITSAVE	000006	DMSCIT DMSDBG DMSIOW
WORKFILE	000005	DMSOLD
WRBIT	000008	DMSACC DMSBWR DMSDSK DMSTPE
WRCOUNT	000001	DMSGIO
WRDATA	000022	DMSINI
WRITE	000028	DMSINI
WRITE1	000007	DMSINI
WRTKF	000003	DMSDIO
WTRDCNT	000002	DMSDBG
XAREA	000001	DMSEDI
XCOUNT	000001	DMSOVS
XGPR0	000002	DMSOVS
XGPR1	000001	DMSOVS
XGPR15	000002	DMSOVS
XPSW	000013	DMSDBG DMSITE

Label	Count	References	
XRSAVE	000003	DMSDIO	
XXXCWD	000043	DMSEDI	
XYCNT	000008	DMSEDI	
XYFLAG	000003	DMSEDI	
YAREA	000001	DMSEDI	
YDISK	000002	DMSINI	
YYDDD	000003	DMSINS	
ZONE1	000011	DMSEDI	DMSEDX
ZONE2	000016	DMSEDI	DMSEDX

Module Name	Entry Points	Attributes, Function
DMKACO		Pageable.
	DMKACON	Provides additional accounting function at logon time (for installation use).
	DMKACODV	Builds an account card buffer for a VDEVBLOK.
	DMKACOFF	Creates account card buffer for a VMBLOK.
	DMKACOPU	Punches queued up accounting cards.
	DMKACOQU	Queues up account card buffers for output on a real device.
	DMKACOTM	Creates a connect and usage time message for a user.
DMKBLD		Pageable.
	DMKBLDEC	Allocates storage for a virtual ECBLOK and the two TRQBLOKs required for a virtual machine with the ECMODE option, and initializes these blocks.
	DMKBLDRL	Releases real segment, page, and swap tables to free storage.
	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 user's virtual machine.
	DMKBLDVM	Creates and partially initializes a VMBLOK for a virtual machine, identified by its terminal real device block.
DMKBOX		Pageable.
		Provides the VM/370 or user logo (header) for printed output.
	DMKBOXBX	Logo for initial screen display and header separator for printer spool files.
DMKBOX	DMKBOXHR	Installation header reference.
DMKBSC		Resident.
		Bisync line error processing.
	DMKBSCER	Examines the error condition resulting from a unit check or channel error that occurred while executing a CP generated bisync line channel program. If the error is uncorrectable, DMKMSW is called 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 retries.
DMKCCH		Resident.
		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.
	DMKCCHIS	Entry from DMKIOS when a channel check occurs when storing a CSW after a SIO.
	DMKCCHNT	Entry from DMKIOINT when a channel check occurs on an I/O interrupt.
	DMKCCHRT	Entry from DMKIOE to allow error messages to be printed.
DMKCCW		Resident.
	DMKCCWSB	Invokes an internal subroutine (CNTRLSUB) to obtain control bytes (seek data).

Module Name	Entry Points	Attributes, Function
DMKCCW (cont.)	DMKCCWTC	Searches previous (external) RCW chains and resolves the address of the RCW task if found.
	DMKCCWTR	Takes the list of virtual CCWs associated with the user's SIO and translates it into a real CCW list.
DMKCDB		Pageable. Processes DISPLAY, DCP, DUMP, and DMCP commands.
	DMKCDBDC	Executes the DISPLAY command to display real storage locations.
	DMKCDBDI	Displays virtual storage locations, storage keys, general registers, floating-point registers, PSW, CAW, and CSW at the terminal.
	DMKCDBDM	Dumps the contents of the specified real storage locations on the virtual printer spool file.
	DMKCDBDU	Dumps the contents of the specified virtual storage locations, registers, PSW, and storage keys on the virtual printer spool file.
DMKCDS		Pageable. Processes STORE and STCP commands.
	DMKCDSCP	Stores data into real storage (STCP command).
	DMKCDSTO	Stores data into virtual storage (STORE command).
DMKCFC		Pageable. Gets the address of the routine that processes the CP console function that was requested.
	DMKCFCMD	Processes a CP console function.
	DMKCFCSL	Processes the SLEEP command.
	DMKCFCBE	Processes the BEGIN command.

Module Name	Entry Points	Attributes, Function
DMKCFC (cont.)	DMKCFCQU	Processes the QUERY command.
	DMKCFCRQ	Presents an attention interruption to the virtual machine to simulate a real request key interruption.
DMKCFD		Pageable. Processes LOCATE and ADSTOP commands.
	DMKCFDAD	Stops virtual machine at specified address (ADSTOP command).
	DMKCFDLO	Displays address of real device blocks, or VMBLOK and/or virtual device blocks (LOCATE command).
DMKCFG		Pageable.
	DMKCFGSV	Saves a system's virtual storage space, including registers and PSW as they currently exist, in page form, on a DASD device. The name of the system and the DASD location at which it is to be saved is defined in DMKSVS.
DMKCFM		Resident. Processes the SLEEP, BEGIN, QUERY, and REQUEST commands. Also processes DIAGNOSE code 8. Its scans the command line and goes to the required module.
	DMKCFMAT	Posts an attention interrupt pending for the virtual machine.
	DMKCFMBK	Puts the terminal in console function (CP) mode (ATTN key pressed twice). Scans the command line and goes to the command handling routine.
	DMKCFMEN	Entered when DIAGNOSE code 8 is executed. Scans the command line and goes to the command handling routine.

Module Name	Entry Points	Attributes, Function
DMKCFP		Pageable. Simulates the operator's console for the virtual machine.
	DMKCFPII	Entry from DMKLOG to process IPL command (logon).
	DMKCFPIP	Entry from DMKCFM to process IPL command.
	DMKCFPRD	Handles virtual device reset for other CP routines.
	DMKCFPRR	Handles system resets for other CP routines. Resets the virtual machine.
	DMKCFPRI	Releases an IOBLOK when called by DMKNLD.
DMKCFS		Pageable. Processes the CP SET command.
	DMKCFSET	Entry point for SET command processor.
DMKCFT		Pageable. Processes user's terminal options.
	DMKCFTRM	Entry point for the TERMINAL command processor.
DMKCKP		Pageable. Saves pertinent data when a check point occurs.
	DMKCKPT	Retrieves accounting data from the VMBLOK, VDEVBLOK, and unpunched accounting cards. It retrieves accounting information for dedicated devices, saves the system log messages, and saves all control blocks for spool files. The data is written on the SYSWARM cylinder of the IPL pack. DMKCKP is loaded and executed by DMKDMP or initial program load.
DMKCKS		Pageable. Performs checkpoint processing.
	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.
	DMKCKSIN	Initializes the check point cylinder after a successful warm start from the standard recovery procedure or after a cold start.
	DMKCKSWM	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.
DMKCNS		Resident. Real console terminal manager.
	DMKCNSED	Edits the input line for the following characters: escape, line end, line delete, and character delete.
	DMKCNSEN	Enables or disables a low-speed terminal line.
	DMKCNSIC	Entered from DMKQCN to initialize read and write CCWs for the CONTASK built by DMKQCN.
	DMKCNSIN	Interruption return point and handler for terminal I/O.
DMKCPB		Pageable. Simulates the operator's console for the virtual machine.
	DMKCPBEX	Processes the EXTERNAL command to present an external interruption to the virtual machine.

Module Name	Entry Points	Attributes, Function
DMKCPB (cont.)	DMKCPBNR	Processes the NOTREADY command to cause the virtual device to appear not ready.
	DMKCPBRS	Processes the RESET command to reset all pending interrupts from the specified device.
	DMKCPBRW	Processes the REWIND command to issue a rewind to the real tape device.
	DMKCPBRY	Processes the READY command to simulate a device end interrupt to the specified device.
	DMKCPBSR	Processes the SYSTEM command to simulate system reset and PSW restart to allow clearing of storage.
DMKCPE		Resident. Contains data constants that define the end of the CP nucleus.
DMKCPI		Pageable. Prepares VM/370 for operation.
	DMKCPIEM	Enables the operator's console, initializes the TOD clock and directory, allows operator logon, prepares for warm start, and completes initialization.
	DMKCPINT	Initializes and prepares CP for operation.
DMKCPS		Pageable. Processes the SHUTDOWN, HALT, and VARY commands.
	DMKCPSSH	Processes the SHUTDOWN command.
	DMKCPSH	Processes the HALT command.
	DMKCPSRY	Processes the VARY command.
DMKCPV		Pageable.
	DMKCPVAA	Punches user accounting records.
DMKCPV (cont.)	DMKCPVAC	Processes the ACNT command to create accounting records for logged on users. Also, resets accumulated accounting information.
	DMKCPVAE	Enables system low-speed lines for system restart.
	DMKCPVDS	Processes the DISABLE command to disable an active line after the current user is finished with it.
	DMKCPVEN	Processes the ENABLE command to enable the system's low-speed lines for system log on.
	DMKCPVLK	Processes the LOCK command to lock specified pages of a user's virtual storage space into real main storage.
	DMKCPVUL	Processes the UNLOCK command to unlock pages that were locked by operator command (LOCK).
DMKCQG		Pageable. Processes the class G and class D QUERY commands.
	DMKCQGEN	Entry to QUERY command processor for class G users.
DMKCQP		Pageable. Processes the class B and class G QUERY command.
	DMKCQPRV	Entry to QUERY command processor for class E and G users.
DMKCQR		Pageable. Processes the QUERY command.
	DMKCQREY	Main entry point. Contains a branch table to get to the routine that processes the operand specified in the QUERY command; the operand can be one of the following: FILES, TIME, SET, LOGMSG, NAMES, USERS, DUMP, PAGING, HOLD, PRIORITY, TERMINAL, PF, SASSIST.

Module Name	Entry Points	Attributes, Function
DMKCQR (cont.)	DMKCQRFI	Retrieves the number of reader, punch, and print files.
DMKCSC		Pageable. Processes real spooling commands for real unit record devices.
	DMKCSOBS	Processes the BACKSPACE command.
	DMKCSODR	Processes the DRAIN command.
	DMKCSOFL	Processes the FLUSH command.
	DMKCSOLD	Processes the LOADBUF command (real UCS or FCB buffer).
	DMKCSORP	Processes the REPEAT command.
	DMKCSOSD	Starts entry point for warm start.
	DMKCSOSP	Processes the SPACE command.
	DMKCSOST	Processes the START command by device type.
	DMKCSOVL	Processes the LOADVFCB (load virtual forms control buffer) command.
DMKCSP		Pageable. Processes class D and G spooling commands.
	DMKCSPCL	Processes the CLOSE command.
	DMKCSPFR	Processes the FREE command.
	DMKCSPHL	Processes the HOLD command.
	DMKCSPSP	Processes the SPOOL command.
DMKCST		Pageable. Processes class G commands.
	DMKCSTAG	Entry point to process the TAG command.
DMKCSU		Pageable. Processes the class D and G spooling commands.
	DMKCSUCH	Processes the CHANGE command.
	DMKCSUCR	Processes the ORDER command.
	DMKCSUPU	Processes the PURGE command.
	DMKCSUTR	Processes the TRANSFER command.

Module Name	Entry Points	Attributes, Function
DMKCVT		Resident. Processes the conversion routines.
	DMKCVTBD	Converts a word of binary data into a doubleword of decimal digits.
	DMKCVTBH	Converts a word of binary data into a doubleword of hexadecimal data.
	DMKCVTDB	Converts a decimal field into a fullword of binary data.
	DMKCVTDT	Converts data and time to EBCDIC and inserts it into a specified location.
	DMKCVTFP	Converts a floating-point doubleword into 17 bytes of decimal data.
	DMKCVTHB	Converts the designated hexadecimal field into a binary fullword.
DMKDAS		Resident. DASD error retry program.
	DMKDASER	Retries the failing DASD channel program.
	DMKDASRD	Processes unsolicited device end interruptions.
	DMKDASSD	Collects DASD sense data.
DMKDDR		Residency not applicable. 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.

Module Name	Entry Points	Attributes, Function
DMKDDR (cont.)	DMKDDREP	DASD dump restore program entry point.
	DMKDDRED	End-of-load module for CMS.
DMKDEF		Pageable. Processes the DEFINE command to define a virtual device or storage.
	DMKDEFIN	Processes the DEFINE command to alter the virtual machine's configuration or storage size.
DMKDGD		Resident. Processes simple disk I/O.
	DMKDGDDK	Performs simple disk I/O of a standardized format with a minimum of CCW chain manipulation and interruption handling.
DMKDIA		Pageable.
	DMKDIACP	COUPLE command processor. Establishes a virtual connection between two channel-to-channel adapters on a single virtual machine.
	DMKDIADR	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 VM/370.
	DMKDIAL	Processes the DIAL command. Attaches a user's terminal as a dedicated device to an existing virtual 270X terminal line in the virtual machine addressed by the command line.
	DMKDIASM	Simulates sense data and status for virtual I/O to a simulated I/O device (2702 line or CTCA) that that has not yet been activated through either the console function DIAL for 2702 lines, or the console function COUPLE for virtual CTCAs.
DMKDIR		Pageable or standalone. 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.
	DMKDIRED	End of load module for CMS.
DMKDMP		Resident. Writes a dump of main storage, control registers, floating-point registers, general registers, and clocks to a specified device.
	DMKDMPDK	Writes the dump on the specified device.
	DMKDMPRS	Initial program loads the system over again.
DMKDRD		Pageable. Process spool files
	DMKDRDDD	Delete system dump spool file.
	DMKDRDER	Manipulates input spool files via a DIAGNOSE code X'0014' issued by the virtual machine.
	DMKDRDMP	Reads a system dump spool file via a DIAGNOSE code X'0034' issued by the virtual machine.
	DMKDRDSY	Reads the system symbol table CSECT via a DIAGNOSE code X'0038' issued by the virtual machine.
DMKDSP		Resident. Entered after each interruption handler is finished processing and after each stacked CPEXBLCK, 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 runable user is dispatched.

Module Name	Entry Points	Attributes, Function
DMKDSP (cont.)	DMKDSPA	Immediate redispatch path for virtual machines. The only status update that occurs is for virtual timers.
	DMKDSPB	Process new virtual PSW and dispatch. Entered if the virtual PSW has been entered outside of DMKDSP.
	DMKDSPCH	Main entry point. Updates timers and dispatch user.
	DMKDSPQS	Nonexecutable; dispatched user's maximum time slice.
	DMKDSPRQ	Queues anchor for IOBLCKs and CPEXBLOKs.
	DMKDSPNP	Number of dynamically assignable page frames now available in the system.
DMKEDM		Runs in a virtual machine under CMS control.
	DMKEDM	Reads the CP dump from the CMS file and edits and prints the following in a readable format: • PSWs • General registers and control registers • CSW and CAW • Load map
DMKEDM	DMKEDM	• Real device blocks and associated control blocks - RHCBLOK, RCUBLOK, RDEVBLOK, IOBLCK, RESPLCTL, SFBLOK, IOERBLOK, ALOCBLOC, RECBLOK • SFBLOK chains for reader, printer, and punch files • Core table • Each user's virtual device blocks and associated control blocks - VMBLOK, VCHBLOK, VCUBLOK, VDEVBLOK, VSPLCTL, SFBLOK, VCONCTL
DMKEDM (cont.)	DMKEDM (cont.)	• Each user's segment page and swap tables. Prints a hex dump of storage suppressing print lines that are duplicates of the preceding lines. Operator options allow: • Print suppression of a formatted dump • Print suppression of a hex dump • Erasing the CMS dump file • Printing a load map • Printing the dump at the user's console. The default of the options is a formatted hex dump printed on device 00E.
DMKEIG		Pageable. Analyses 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.
DMKEMA		Pageable.
	DMKEMA	Contains the framework of the common error messages that are generated at various places within CP. Module DMKERM references DMKEMA to write error messages that require variable data to be inserted into them.
DMKEMA	DMKEMA	This module contains no executable code and contains all error messages from 0 to 225.

Module Name	Entry Points	Attributes, Function
DMKEMB		Pageable.
	DMKEMB	Contains the framework for the common error messages that are generated at various places within CP. The module DMKERM references DMKEMB to write error messages that require variable data to be written into them. This module contains no executable code and contains error messages 256 and up.
DMKERM		Pageable.
	DMKERMSG	This is the message writer. Locates the requested message and inserts the module ID, message number, and data. It also prints the message.
DMKFCB		Pageable.
	DMKFCB	Contains the forms control load buffer images that the LOADBUF command uses to load the forms control buffer in the 3811 control unit for the 3211 printer. The LOADVFCB command also uses DMKFCB to load the forms control buffer in the virtual 3211 printer.
DMKFMT		Standalone program. Initial program loaded or run under CMS if on a CMS disk.
DMKFMT (cont.)	DMKFMT	Adapts parameters from the console or IPL device (card reader) and performs partial or complete formatting, allocating, and labeling of 2314, 2319, 3330, 3340, 3350 and 2305 DASD 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.
DMKFRE		Resident.
		Free storage manager.
	DMKFREE	Gets space from free storage.
	DMKFRERS	Returns subpools to free storage chain.
	DMKFRET	Returns space to free storage.
	DMKFRETR	Returns space to free storage; does not release pages.
DMKGIO		Pageable.
		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.

Module Name	Entry Points	Attributes, Function
DMKGRF		Resident. Supports local 3270 and 3066 devices. DMKGRF processes interruptions and CCWs for the devices. The processing includes message handling and screen management.
	DMKGRFIN	Handles the interruption via an IOBLOK.
	DMKGRFEN	Enables or disables the device.
	DMKGRFIC	Starts a CONTASK from DMKQCN.
DMKHVC		Resident.
	DMKHVCAL	Performs services for the virtual machine as requested via the DIAGNOSE instruction. The specific service performed depends on the code in the DIAGNOSE instruction.
DMKHVD		Pageable.
	DMKHVDAL	Performs services for virtual machines as requested by the DIAGNOSE instruciton.
DMKIOC	DMKIOCVT	Converts VM/370 device type to OS/VS device type.
DMKICE		Resident. 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 a check to determine if the recording cylinder is full. DMKIOE also interfaces with the pageable module that initializes and erases the error recording cylinders.

Module Name	Entry Points	Attributes, Function
DMKICE (cont.)	DMKIOECC	Entry for a channel error condition occurring on a SIO in DMKIOS with a response condition code of one.
	DMKIOECH	Entry for a stacked channel recording request from the channel check handler.
	DMKIOECJ	Entry for a stacked channel check recording request from ERP.
	DMKIOEFL	Entry point to locate the starting page record for recording.
	DMKIOEFM	Entry to clear and format the recording area on disk.
	DMKIOEMC	Entry for machine check recording.
	DMKIOEMH	Entry for a stacked machine check request.
	DMKIOENV	Entry for a stacked environmental recording request.
	DMKIOEOB	Entry for a stacked outboard error recording request.
	DMKIOEQQ	Calls to initiate error recording via DMKIOF (no DMKIOE function performed).
	DMKIOERC	Entry for a stacked erase request.
	DMKIOERN	Processes a 3704/3705 and remote 3270 request.
	DMKIOERR	Schedules recording for unit check, channel data check, and hardware environmental counts.
	DMKIOESD	Records 3330 data.
	DMKIOESR	Schedules statistical data recording.
	DMKIOEST	Schedules the update of a statistical data request.
	DMKIOEVR	Processes an SVC 76 request.
DMKIOF		Pageable. Records system and I/O errors on the system disk in predefined error recording cylinders.
	DMKIOFC1	Records channel check error from SIO in DMKIOS when CC=1.

Module Name	Entry Points	Attributes, Function
DMKIOF (cont.)	DMKIOFIN	Initializes pointers to available recording pages at IPL and after an erase has been completed.
	DMKIOFOB	Records OBR and MDR records.
	DMKIOFM1	Records machine checks.
	DMKIOFST	Updates statistical data counters.
	DMKIOFVR	Records errors when requested by SVC 76.
DMKICG		Pageable. 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 firsr 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.
	DMKIOGF1	Contains all function of DMKIOS except erase.
	DMKIOGF2	Erases either the machine check handler or channel check handler recording cylinder, or the outboard recording cylinder; either separately or combined.
DMKICS		Resident. Schedules requests for virtual machine and program I/O operations, and services all I/O interruptions.
	DMKIOSHA	Halts an active device and drains all interruptions.
	DMKIOSIN	Processes an I/O interruption.
	DMKIOSQR	Schedules CP generated I/C operation.
	DMKIOSQV	Schedules a virtual machine I/O operation.
	DMKIOSRW	Processes the IOBLOK used for REWIND.

Module Name	Entry Points	Attributes, Function
DMKISM		Pageable.
	DMKISMTR	Finds and modifies an ISAM CCW string.
DMKLDOO		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.
DMKLCC		Resident.
	DMKLOCK	Allows a system resource to be marked in use or not available by a unique 8-character name.
	DMKLOCKD	Dequeues a locked name.
	DMKLOCKQ	Queues or locks a name.
	DMKLOCKT	Tests to determine if a name is locked.
DMKLNK		Pageable.
	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.
	DMKLNKIN	Links to a virtual DASD device because of an issued LINK command.
	DMKLNKSB	LINK subroutines.
DMKLOG		Pageable. Logs on a user or operator.
	DMKLOGON	Logs on a user.
	DMKLOGCP	Logs on the operator.
	DMKLOGA	Processes the AUTOLOG command.
DMKMCC		Pageable.
	DMKMCCCL	Processes the MONITOR START or the MONITOR STOP command.

Module Name	Entry Points	Attributes, Function
DMKMCH		Resident.
	DMKMCHIN	Processes a machine check interruption.
	DMKMCHMS	Enables or disables soft machine check recording.
DMKMID		Pageable.
	DMKMIDNT	Changes the date in the system low storage at midnight and resets the clock comparator for the next midnight occurence. DMKMID also sends messages to all users about the date change.
DMKMCN		Pageable. Processes commands and requests associated with the MONITOR, including MONITOR CALL interruptions within CP.
	DMKMONIO	Processes tape interruptions returned by DMKIOS.
	DMKMONMI	Processes a MONITOR CALL program interruption.
	DMKMONSH	Routine to stop the MONITOR command.
	DMKMONTH	Routine to write MONITOR tape header records.
	DMKMONTI	Handle timer request interruptions.
DMKMSG		Pageable. Transmits messages to logged on users for the MESSAGE or WARNING commands. Receives and retransmits lines for the ECHO command for the number of times specified.
	DMKMSGEC	ECHO command processor.
	DMKMSGMS	MESSAGE command processor.
	DMKMSGWN	WARNING command processor.
DMKMSW		Resident.
	DMKMSWR	Allows system communication with the operator for the enhancement of error recovery procedures.

Module Name	Entry Points	Attributes, Function
DMKNEM		Pageable.
	DMKNEMOP	Gets a 5-byte mnemonic opcode for a System/370 binary opcode.
DMKNES		Pageable. Processes NETWORK operands as follows: POLLDLAY SHUTDOWN DISPLAY VARY TRACE
	DMKNESDS	Processes the NETWORK DISPLAY command.
	DMKNESEP	Processes the NETWORK VARY EP command to switch an NCP communication line to EP mode.
	DMKNESHD	Processes the NETWORK SHUTDOWN command.
	DMKNESPL	Processes the NETWORK POLLDLAY command.
	DMKNESTR	Processes the NETWORK TRACE command.
	DMKNESWN	Processes the NETWORK VARY NCP command to switch an EP communication line tc NCP mode.
DMKNET		Pageable. Decodes NETWORK command and enables bisync lines.
	DMKNETAE	Enable bisync lines and remote stations.
	DMKNETWK	NETWORK command decoder.
DMKNLD		Pageable.
	DMKNLDMP	Dumps the 3705 network control program.
	DMKNLDR	Loads the 3705 network control program. These routines may be called by a console command from DMKNET or or internally by DMKCPI (for LOAD) or DMKRNH (for DUMP).

Module Name	Entry Points	Attributes, Function
DMKOPR		Resident.
	DMKCPRWT	Provides the necessary support for the VM/370 system console. Certain routines within the control program can not call DMKQCN to issue writes to the system console. This module determines the system's primary console and builds a channel program to handle the requested call.
DMKPAG		Resident.
	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 load at user specified intervals.
DMKPER		Pageable.
	DMKPERCH	Sets a return code of zero in R2.
	DMKPERIL	Resets the interruption.
	DMKPERT	Resets program event recording.
DMKPGS		Pageable.
	DMKPGSPO	Release all the pages of a user's virtual storage - from the real storage and from auxiliary storage on the paging device.
	DMKPGSPP	Releases a specified part of virtual storage.
DMKPGT		Resident.
		DASD storage management.
	DMKPGTCG	Allocates contiguous space for a 3704/3705 dump.
	DMKPGTPG	Allocates a page of DASD storage for either virtual storage paging or for spool file page buffers.
DMKPGT (cont.)	DMKPGTPR	Releases DASD storage used for virtual storage paging.
	DMKPGTSD	Releases one page of DASD storage used for spooling.
	DMKPGTSG	Allccates a page of DASD storage for spooling.
	DMKPGTSR	Releases a group of DASD storage pages used for spooling.
	DMKPGTVG	Allccates a page of virtual storage belonging to the CP paging VMBLOK.
	DMKPGTVR	Releases a virtual storage page.
DMKPRG		Resident.
	DMKPRGIN	Processes a hardware program interruption.
	DMKPRGRF	Reflects an SVC interruption to the virtual machine.
	DMKPRGSM	Simulates a virtual program interruption.
DMKPRV		Resident.
	DMKPRVLG	Simulates a privileged operation.
DMKPSA		Resident.
	DMKPSADU	PSW restart processing. Forces an SVC 0 type of dump.
	DMKPSAEX	Processes external interruptions.
	DMKPSAFP	Checks for fetch protection violation per PSW key.
	DMKPSAID	Gets virtual address for any instruction.
	DMKPSARR	Gets the virtual address for an RR instruction.
	DMKPSARS	Gets the virtual address for RS,SI, or SS instruction.
	DMKPSARX	Gets the virtual address for an RX instruction.
	DMKPSASP	Checks for a storage protection violation per the PSW key.
	DMKPSASV	Processes SVC interruptions.

Module Name	Entry Points	Attributes, Function
	DMKPSAFC	Checks fetch protection per the CAW key.
	DMKPSASC	Checks storage protection per the CAW key.
DMKPTR		Resident. 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.
	DMKPTRFR	Gets a page of real storage.
	DMKPTRFT	Releases a page of real storage.
	DMKPTRLK	Locks a page of real storage.
	DMKPTRPW	Called to defer execution of system reset functions when user's virtual machine is in page wait.
	DMKPTRUL	Unlocks a page of real storage.
DMKQCN		Resident.
	DMKQCNCL	Clears CONTASK stack and returns all blocks to free storage.
	DMKQCNET	Processes completed CONTASKS for virtual console spooling, return or no return options, and returns the CONTASK blocks to free storage.
	DMKQCNRD	Starts and queues a console read request.
	DMKQCNSY	Synchronizes virtual machine console activity with internal supervisor activity. This is used during a virtual system reset and during the logoff process.
	DMKQCNTO	Disconnects a virtual machine and sets a TOD clock comparator request to logoff the virtual machine after a fifteen minute delay.

Module Name	Entry Points	Attributes, Function
DMKQCN (cont.)	DMKQCNWT	Starts and queues a console write request.
DMKRGA		Resident.
	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.
DMKRGB		Resident.
	DMKRGB	Supports the 3270 remote display and printer stations. It processes interruptions and CCWs for the remote stations including message handling and screen management.
	DMKRGBIC	Initializes and schedules CONTASKS.
	DMKRGBEN	Enables and disables bisync lines and remote stations.
DMKRIO		Resident.
	DMKRIO	Exists as a CSECT and defines the machine's configuration. A basic DMKRIO is shipped with VM/370. 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.
	DMKRND	This is the interface between the VM/370 dump spool file and the OS-SSP dump format program for printing and formatting dumps of the 3704 and 3705 communications controllers.

Module Name	Entry Points	Attributes, Function
DMKRNH		Resident.
	DMKRNHIC	Initializes and schedules the CONTASK fields that comprise the 3704 and 3705 Network Control Program transmission header.
	DMKRNHIN	This is the secondary interruption handler for the 3704 and 3705 communication controllers; it is read when operating in NCP or PEP mode.
	DMKRNHND	Schedules control functions for the 3705 or 3704 Network Control Program.
DMKRPA		Resident. Virtual storage mapping.
	DMKRPAGT	Page-in from DASD to user's virtual storage.
	DMKRPAPT	Page-out to DASD from user's virtual storage.
DMKRSE		Pageable. Real UR device I/O error handler.
	DMKRSERR	Retries and attempts to recover from real unit record device I/O errors.
DMKRSP		Resident. Manages all spooling operations on the real system unit record devices including printing and punching user-created spool files and reading and queueing reader files from the real card reader.
	DMKRSPER	Processes spooling errors (ERP).
	DMKRSPEX	Processes spooling operations. Entered via a GOTO when DMKDSPCH unstacks an IOBLOK with an interruption for the spooling unit record device.
DMKRSP	DMKRSPUR	Formats the active file message for real UR devices.
DMKSAV		Pageable. DMKSAVNC is entered via an LDT card from DMKLDR. DMKSAVRS is entered via a BALR from DMKCKP. DMKSAV saves and restores a page image count of the CP nucleus on the system residence disk.
	DMKSAVRS	Restores a page image copy of the CP nucleus.
	DMKSAVNC	Writes a page image copy of the CP nucleus.
DMKSCH		Resident. Maintains queues of runable and eligible users, alters the dispatching status of users, and periodically recalculates the working set size and dispaching 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.
	DMKSCHCP	Interruption from real CPU timer.
	DMKSCHDL	Alters a user's dispatching status.
	DMKSCHMD	Interruption for the midnight date change.
	DMKSCHRT	Resets a clock comparator interruption request.
	DMKSCHST	Establishes a clock comparator interruption request.
	DMKSCH80	Interruption for real timer at storage address 80.
	DMKSCHAE	Processes interruption occurring when the favored execution measurement interval expires.

Module Name	Entry Points	Attributes, Function
DMKSCN		Resident. Scans module.
	DMKSCNAU	Searches the chain of VMBLOKs for one whose userid matches the one pointed to by register one.
	DMKSCNFD	Finds the next field in an input message buffer.
DMKSCN	DMKSCNLI	Searches the logged on virtual machines for any links to a specified minidisk. A link is any virtual device whose ADEVBLOK pointer and relocation factor match those specified.
	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.
	DMKSCNRN	Returns the name of the real device to the caller in register 1.
	DMKSCNRU	Returns the addresses of the real channel, control unit, and device blocks for a given real device to the caller.
	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.
	DMKSCNVN	Returns the name of the virtual device to the caller in register 1.
	DMKSCNVS	Searches all the real device blocks for a device whose volume serial number matches the one pointed to by register 1.
	DMKSCNVU	Returns the addresses of the virtual channel, control unit, and device blocks for a given real device to the caller.

Module Name	Entry Points	Attributes, Function
DMKSEP		Pageable.
	DMKSEPSP	Prints and punches the respective output separators on real spooling devices.
DMKSEV		Pageable but locked.
	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.
DMKSIX		Pageable but locked. 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.
DMKSNC		Pageable.
	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.
DMKSNT		Pageable.
	DMKSNTBL	This module is assembled by the installation system programmer. It describes the system to be saved via the SAVESYS command and to be initial program loaded by name. Shared segments may be specified. These segments consist of all re-entrant code and no altering of this storage is allowed. There is no executable code in this module.

Module Name	Entry Points	Attributes, Function
DMKSPL		Pageable. Spool file manager.
	DMKSPLCR	Closes and queues a real reader spool file for virtual input.
	DMKSPLCV	Closes and queues a virtual printer or punch spool file for processing.
	DMKSPLDL	Deletes used files from the system and de-allocates the DASD page space.
	DMKSPLOR	Initializes control blocks and buffers for real input reader files.
	DMKSPLOV	Initializes control blocks and buffers for virtual printer and punch output spool files.
DMKSSP		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 is figured based on the installation's REALIO deck.
	DMKSSP01	Entered as a result of an IPL operation. Constructs the I/O blocks and system modules for a minimum system configuration.
DMKSTK		Resident. Stacks I/O blocks.
	DMKSTKCP	Stacks a CPEXBLOK.
	DMKSTKIO	Stacks an IOBLOK.
DMKSYM		Pageable
	DMKSYM	Provides a symbol table of all CSECTS and entry points.

Module Name	Entry Points	Attributes, Function
DMKSYS		Resident.
	DMKSYS	Exists as a CSECT that defines the system residence volume, paging space, operator ID, dump ID, storage size, and time zone.
DMKTAP		Pageable.
	DMKTAP	Examines the error condition resulting from a unit check while executing a CPP 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 re-executed or terminated.
	DMKTAPER	Retries the failing tape channel program, after a tape positioning command has been executed.
DMKTBL		Resident.
	DMKTBL	Contains the terminal translate tables.
DMKTBM		Pageable. Contains terminal translate tables for APL.
	DMKTBMZO	3270 APL compound write translation.
	DMKTBMZI	3270 APL compound read translation.
	DMKTBMMO	EBCDIC to APL correspondence terminal code. APL correspondence terminal code to APL.
	DMKTBMNO	EBCDIC to APL PTTC/EBCD terminal code.
	DMKTBMNI	APL PTTC/EBCD terminal code to EBCDIC.

Module Name	Entry Points	Attributes, Function
DMKTDK		Pageable.
	DMKTDKGT	Allocates cylinders of temporary disk space from owned volumes.
	DMKTDKRL	Releases temporary disk space to the pool of free space.
DMKTHI		Pageable. Displays data about use of and contention for major system resources.
	DMKTHIEN	Processes INDICATE command.
DMKTMR		Resident. Simulates the CPU timer and time-of-day clock comparator instructions for virtual System/370's operating in extended control mode.
	DMKTMRCK	Simulates virtual clock comparator interruptions.
	DMKTMRPT	Calculates user's total virtual problem time.
	DMKTMRTN	Simulates timer instruction.
	DMKTMRVT	Simulates virtual CPU timer interruptions.
DMKTRA		Pageable. Processes the TRACE command line. Provides a virtual machine with the facility to track SVC instructions, program interruptions, external interruptions, successful searches, or all instructions with output on the printer, terminal, or both.
	DMKTRACE	TRACE command processor.

Module Name	Entry Points	Attributes, Function
DMKTRC		Pageable. Processes the TRACE command functions.
	DMKTRCEX	Traces external interruptions.
	DMKTRCIO	Traces I/O interruptions.
	DMKTRCIT	Sets the needed SVC B2 for instruction tracing.
	DMKTRCND	Ends tracing.
	DMKTRCPB	Puts back user instructions altered by tracing.
	DMKTRCPG	Traces program interruptions.
	DMKTRCPV	Traces privileged instruction interruptions.
	DMKTRCSI	Traces I/O operations (SIO, TIO, HIO, TCH).
	DMKTRCSV	Processes an SVC, Branch, or full instruction TRACE.
	DMKTRCSW	Traces virtual and real CSWs.
	DMKTRCWT	Serialization entry for I/O and CCW tracing.
DMKTRM		Pageable.
	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.
DMKUCB		Pageable.
	DMKUCB	Contains the UCB buffer load images used by the LOAD command to load the universal character set buffer in the 3811 control unit. This module contains no executable code.

Module Name	Entry Points	Attributes, Function
DMKUCS		Pageable. Contains the UCS buffer load images that the LOAD command uses to load the universal character set buffer in the 2821 control unit. This module contains no executable code.
DMKUDR		Pageable.
	DMKUDRBV	Allows the DMKDIRCT or DMKCPINT programs to build a list of virtual page buffers; one for each UDIRBLOK page on disk.
	DMKUDRDS	Allows the DMKDIRCT program to swap the active user directory to newly created user directory.
	DMKUDRFD	Puts specified UDEVBLOK into the caller's buffer.
	DMKUDRFU	Finds a given user ID in the user directory and moves the user's directory entry into the caller's buffer.
	DMKUDRRD	Reads the next user directory into the caller's buffer.
	DMKUDRRV	Releases a virtual page used by the directory program as a buffer.
DMKUNT		Resident. Untranslates CCWs and CSWs.
	DMKUNTFR	Releases pages and free storage used for the CCW chain.
	DMKUNTIS	Finds the RCWTASKS that have been patched to handle OS ISAM self-modifying sequences and put them back the way DMKCCW had them to allow DMKUNTRN and DMKUNTFR to operate correctly.
	DMKUNTRN	Translates a real CSW into a virtual CSW.
DMKUNT (cont.)	DMKUNTRS	Relocates sense byte information. For a 3330, 3340, 3350, or 2305, computes virtual cylinder member in bytes 5 and 6 of the sense byte 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 three of the sense data from the real sense data given by the hardware.
DMKUSO		Pageable. Processes user termination.
	DMKUSODS	Processes the DISCONN (disconnect) command.
	DMKUSOFF	Logs off a user.
	DMKUSOFL	Processes the FORCE command.
	DMKUSOLG	Processes the LOGOFF command.
	DMKUSOFM	Returns subpools from the free storage chain and removes spool file blocks and allocation blocks from the dynamic paging area.
DMKVAT		Resident. Storage management for EC mode virtual machine.
	DMKVATAB	Allocates, initializes and maintains shadow, segment, and page tables for virtual machines that can relocate.
	DMKVATBC	Returns active shadow tables to free storage.
	DMKVATEX	Services page or segment exceptions for virtual extended control machines.
	DMKVATLA	Virtual - virtual to virtual address translation.
	DMKVATMD	Allocates and initializes shadow tables.

Module Name	Entry Points	Attributes, Function
DMKVAT (cont.)	DMKVATPF	Handles pseudo page fault interruption from a VS1 virtual machine.
	DMKVATPX	Processes paging exceptions for a virtual machine that performs paging.
	DMKVATRN	Virtual (shadow) — virtual to real address translation.
	DMKVATSX	Processes segment exception for a virtual machine that performs paging.
DMKVCA		Pageable. Simulates I/O for a virtual channel-to-channel adapter.
	DMKVCARD	Selectively resets a device for a virtual channel-to-channel adapter without decoupling the CTCA from the Y-side adapter.
	DMKVCARS	Does a final reset for a virtual channel-to-channel adapter and and disconnects the adapter from its coupled twin on the Y-side virtual machine.
	DMKVCASH	Simulates the execution of a Halt I/O or Halt Device instruction for a virtual machine channel-to-channel adapter.
	DMKVCAST	Simulates the channel and device operations of the channel-to-channel adapter (CTCA) connected between two virtual machines under VM/370.
	DMKVCATS	Simulates the TEST I/O instruction for a virtual channel-to-channel adapter that has no interruptions pending.
DMKVCH		Pageable.
	DMKVCHDC	Processes the ATTACH and DETACH real devices and channels) command.
DMKVCN		Resident.
	DMKVCNEX	Simulates all SIOs to a virtual console.
DMKVDB		Pageable.
	DMKVDBAT	Processes the ATTACH command to a real device as a virtual device to a user or dedicates all devices on a particular channel to a specific user.
	DMKVDBDE	Processes the DETACH command to detach a real or virtual device from a user or detaches a previously-dedicated channel from a user.
DMKVDR		Pageable.
	DMKVDREL	Releases a virtual or real device from a virtual user.
DMKVDS		Pageable.
	DMKVDSAT	Attaches a virtual device to a user.
	DMKVDSDF	Defines a new virtual device for user.
	DMKVDSLK	Links a virtual DASD device to a user.
DMKVER		Pageable. Processes error records from virtual machine via SVC 76.
	DMKVERD	Processes the SVC 76 from DOS or DOS/VS.
	DMKVERO	Processes the SVC 76 from CS, VS/1, VS/2, or VM/370.

Module Name	Entry Points	Attributes, Function
DMKVIO		Resident. Simulates the operation of privileged I/O instructions issued by the virtual machine and records and translates the interruptions and status associated with virtual I/O operations.
	DMKVIOEX	Simulates a SIO, TIO, HIO, or TCH.
	DMKVIOIN	Translate a virtual I/O interuption.
	DMKVIOMK	Address of a table of interruption masks, indexable by device address.
	DMKVIODC	Processes interruptions for dedicated channels.
DMKVMA		Resident.
	DMKVMACP	Called by the command processors via an SVC if the command execution is to change a shared page. The virtual machine is notified that the command has released the shared system. The user continues to run without a shared copy of the named system.
	DMKVMAPS	Called by DMKPTR when the paging manager detects that a shared page has been changed. The current LASTUSER is the only virtual machine that could have changed the page. The shared named system is loacated and made non-shared for the current RUNUSER. Any other shared systems that may exist for RUNUSER are left as a shared system. Other users of the shared system are unaffected by the violation caused by RUNUSER.
	DMKVMASH	Checks all shared pages associated with shared named systems and determines if they have been changed. If they were changed, the condition code is made non-zero and register 2 contains the real address of the page that was changed.
DMKVMI		Pageable - loaded into the user's virtual storage when invoked. Performs an IPL of a virtual machine.
	DMKVMIPL	Simulates a user's IPL sequence.
DMKVSP		Resident. Simulates all user SIOs to a virtual unit record device (real reader, punch, print, or psuedo timer) That is spooled rather than dedicated. It also handles control probram requests to print on the user's virtual printer.
	DMKVSPCO	Stops processing the file currently in the spooled printer or punch and clears all pending status from the spooled printer or punch.
	DMKVSPCP	Writes a print line to the console.
	DMKVSPCR	Stops processing the file currently in the spooled card reader and clears all pending status from the spooled card reader. status from the spooled printer
	DMKVSPEX	Simulates SIO to a spooled unit record device.
	DMKVSPRT	Puts a CP-generated line on the user's spooled printer.
	DMKVSPTO	Checks if the virtual reader is empty.
	DMKVSPVP	Simulates SIO to a spooled virtual console.
	DMKVSPWA	Nonexecutable index work area for 2311.
DMKWRM		Pageable.
	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.

Module	External References (Labels and Modules)												
DMKACO	ACNTBACK	ACNTBLOK	ACNTCCW	ACNTDATA	ACNTNEXT	ACNTSIZE	ACORETBL	ARIODV	ARIOPU	ARSPAC	ASYSLC	ASYSVM	CC
	CLASDASD	CPEXADD	CPEXBLOK	CPEXSIZE	DE	DEVCARD	DFRET	DMKCVTBH	DMKDSPCH	DMKERMSG	DMKFREE	DMKFRET	DMKIOSQR
	DMKPTRIK	DMKPTRUL	DMKQCNWT	DMKRSPEX	DMKSTKCP	DMKSTKIO	DMKSYSCK	DMKTMRPT	F1	F4	F60	F8	IOBCAW
	IOBCP	IOBCSW	IOBFATAL	IOBFLAG	IOBIRA	IOBLINK	IOBLOK	IOBMISC	IOBMISC2	IOBRADD	IOBSIZE	IOBSPEC	IOBSTAT
	IOBUSER	NORET	PRIORITY	PSA	RDEVACNT	RDEVBLOK	RDEVBUSY	RDEVCLAS	RDEVDED	RDEVDISA	RDEVDRAN	RDEVFLAG	RDEVSPL
	RDEVSTAT	RDEVTMAT	RDEVTYPE	R0	R1	R10	R11	R12	R13	R14	R15	R2	R3
	R4	R5	R7	R8	R9	SAVEAREA	SAVER11	SAVEWRK2	SAVEWRK3	SAVEWRK6	SAVEWRK7	SAVEWRK8	SAVEWRK9
	SILI	SKIP	SYSLOCS	TYP2540P	USERCARD	VDEVBLOK	VDEVFLAG	VDEVREAL	VDEVTDSK	VDEVTMAT	VDEVTYPC	VMBLOK	VMIOCNT
	VMLOGOFF	VMPGREAD	VMRSTAT	VMTIMEON	VMTTIME	VMVTIME	ZEROES						
DMKBLD	ACORETBL	ASYSLC	ASYSVM	AVMREAL	CLASSPEC	CLASTERM	CORCFLCK	CORFLAG	CORFPNT	CORLCNT	CORPGPNT	CORSWPNT	CORTABLE
	DELPAGES	DELSEGS	DMKCVTBH	DMKERMSG	DMKFREE	DMKFRET	DMKQCNWT	DMKRIORN	DMKSCHCP	DMKSCNRD	DMKSYSLE	DMKSYSLL	DMKTMRCK
	ECBLOK	EXTCCTRQ	EXTCPTRQ	EXTCR0	EXTCR14	EXTCR15	EXTCR2	EXTSIZE	FFS	F1	F15	F16	F4
	F4095	F7	F8	KEEPSEGS	MICBLOK	MICRSEG	NEWPAGES	NEWSEGS	NICBLOK	NICCIBM	NICLLEN	NICNAME	NICTERM
	NICTYPE	NICUSER	NORET	OLDVMSEG	PAGCORE	PAGINVAL	PAGSWP	PAGTABLE	PSA	RDEVBLOK	RDEVFLAG	RDEVLLEN	RDEVPSUP
	RDEVTYPC	RDEVTYPE	RDEVUSER	R0	R1	R10	R11	R13	R14	R15	R2	R3	R4
	R5	R6	R7	R8	R9	SAVEAREA	SAVER1	SAVER11	SAVER2	SAVER8	SAVEWRK1	SAVEWRK2	SEGPAGE
	SEGPLEN	SEGTABLE	SWPFLAG	SWPPAG	SWPRECMP	SWPTABLE	SWPVM	SYSLOCS	TRQBIRA	TRQBLOK	TRQBSIZE	TRQBUSER	TYPBSC
	TYP3705	VMAEX	VMBLOK	VMBSIZE	VMCFWAIT	VMCHTBL	VMECEXT	VMLOGON	VMMCODE	VMMICRO	VMMLEVEL	VMMSGON	VMMTEXT
	VMPAGES	VMPNT	VMPSTAT	VMPSW	VMQLEVEL	VMREAL	VMRSTAT	VMSEG	VMSEGDSP	VMSIZE	VMSTOR	VMTERM	VMTLEND
	VMTMOUTQ	VMTRMID	VMTTIME	VMUSER	VMVTERM	VMV370R	VMWNGON	VMWSPROJ	VRALOC	WAIT	XPAGNUM	ZEROES	
DMKBSC	BSCBLOK	BSCREAD	BSCRESP	CC	CCC	CDC	CHC	DMKFREE	DMKFRET	DMKIOEST	DMKMSWR	F1	F7
	F8	IFCC	IOBCAW	IOBERP	IOBFATAL	IOBFLAG	IOBIOER	IOBLOK	IOBRCAW	IOBRCNT	IOBRSTRT	IOBSTAT	IOERBLOK
	IOERCAN	IOERCSW	IOERDATA	IOERDW	IOEREXT	IOERFLG2	IOERFLG3	IOERIND3	IOERINFO	IOERLOC	IOERMSW	IOERNUM	IOEROVFL
	IOERREAD	IOERSIZE	PRGC	PRTC	PSA	RDEVBLOK	RDEVBSC	RDEVIOER	R0	R1	R10	R13	R2
	R3	R4	R5	R6	R7	R8	R9	SAVEAREA	SILI	UC	ZEROES		
DMKCCH	ALARM	ARIOCU	ARIODV	CAW	CCC	CCCPUID	CCDEVTYP	CCHADDR	CCHANID	CCHCAV	CCHCUA	CCHHIO	CCHINTB
	CCHLOG45	CCHLOG80	CCHRCV	CCHREC	CCHSIOB	CCHSIZE	CCHSIZE1	CCHSNSB	CCHTIO	CCPROGID	CCRECTYP	CDC	COMPSYS
	CPID	CPUID	CSW	C7	DEVCCH	DMKCVTBH	DMKFREE	DMKFRET	DMKIOECC	DMKMCHAR	DMKOPRWT	DMKQCNWT	DMKSCNRU
	DMKSYSCK	DMKSYSRM	ECSWLOG	FAILADD	FAILCCW	FAILCSW	FAILECSW	FFS	F16	F7	F8	HIOCCH	IFCC
	IGPRGFLG	IGTERMSQ	IGVALIDB	INTERCCH	INTTIO	IOBCCH	IOBCP	IOBCSW	IOBFLAG	IOBHIO	IOBIOER	IOBLOK	IOBRADD
	IOBSPEC	IOBTIO	IOBUSER	IOELPNTR	IOERBLOK	IOERCSW	IOERDATA	IOERECSW	IOEREXT	IOERSIZE	IOOPSW	MCHAREA	MCHMODEL
	MODEL145	MODEL165	NORET	OPERATOR	PSA	RCHADD	RCHBLOK	RCHCUTBL	RCUADD	RCUBLOK	RCUDVTBL	RDEVADD	RDEVAIOB
	RDEVBLOK	RDEVBUSY	RDEVSTAT	R0	R1	R10	R11	R13	R14	R15	R2	R3	R4
	R5	R6	R7	R8	R9	SAVEAREA	SAVEWRK1	SAVEWRK2	SAVEWRK3	SAVEWRK4	SAVEWRK5	SAVEWRK6	SAVEWRK7
	SAVEWRK8	SAVEWRK9	SIOCCH	TERMSYS	TIOCCH	VMBLOK	VMUSER						

Module	External References (Labels and Modules)											
DMKCCW	ACORETBL	APTRLK	BALRSAVE	BALR2	BALR3	BRING	CC	CD	CLASDASD	CLASGRAF	CLASSPEC	CLASTAPE
	CLASTERM	CLASURI	CLASURO	CORFLAG	CORSHARE	CORSWPNT	CORTABLE	CPSHRLK	CPSTAT2	C1	DEFER	DMKDASSD
	DMKDIASM	DMKFREE	DMKFRET	DMKISMTR	DMKPTRAN	DMKPTRFR	DMKPTRUL	DMKSYSRM	DMKUNTRS	DMKVMAPS	FFS	FTREXTSN
	F1	F10	F15	F16	F2	F240	F3	F4	F4095	F4096	F7	F8
	F9	IDA	IOBCAW	IOBCYL	IOBFLAG	IOBLOK	IOBMISC	IOBMISC2	IOBRELCU	IOBSIZE	IOBSTAT	IOBWRAP
	IOERBLOK	IOERDATA	IOEREXT	IOERLEN	IOERSIZE	LOCK	PCIF	PSA	RCWADDR	RCWCCNT	RCWCCW	RCWCNT
	RCWCOMND	RCWCTL	RCWFLAG	RCWGEN	RCWHEAD	RCWHMR	RCWINVL	RCWIO	RCWISAM	RCWPNT	RCWRCNT	RCWREL
	RCWSHR	RCWTASK	RCWVCAW	RCWVCNT	RCW2311	RDEVBLOK	RDEVFTR	RDEVSADN	R0	R1	R10	R11
	R12	R13	R14	R15	R2	R3	R4	R5	R6	R7	R8	R9
	SAVEAREA	SAVEREGS	SAVER1	SAVER10	SAVER12	SAVER2	SAVER8	SAVER9	SAVEWRK1	SAVEWRK2	SAVEWRK9	SILI
	SKIP	SWPFLAG	TEMPR10	TEMPR14	TEMPR15	TEMPR2	TEMPR3	TYPUNSUP	TYP1442R	TYP2305	TYP2311	TYP2314
	TYP2955	TYP3210	TYP3277	TYP3330	TYP3340	TYP3350	TYP3410	TYP3420	TYP3705	VDEVBLOK	VDEVBND	VDEVDED
	VDEVDIAL	VDEVENAB	VDEVFLAG	VDEVIOER	VDEVPOSN	VDEVRDO	VDEVREAL	VDEVRELN	VDEVRSRL	VDEVSAS	VDEVSTAT	VDEVTYPC
	VDEVTYPE	VDEVUC	VDEV231B	VDEV231T	VMBLOK	VMCLASSF	VMCLEVEL	VMDVSTRT	VMISAM	VMOSTAT	VMPSTAT	VMSEG
	VMSHR	VMSIZE	XPAGNUM	XRIGHT16	XRIGHT24	X2048BND	ZEROES					
DMKCDB	BRING	BUFFER	BUFNXT	C1	DEFER	DMKCVTBD	DMKCVTBH	DMKCVTDB	DMKCVTFP	DMKCVTHB	DMKDMPTR	DMKERMSG
	DMKFREE	DMKFRET	DMKPTRAN	DMKQCNWT	DMKSCNFD	DMKSYSRM	DMKVATAB	DMKVSPRT	ECBLOK	EXTCR0	FFS	F1
	F15	F16	F2	F24	F3	F4	F4096	F6	NORET	PSA	R0	R1
	R10	R11	R13	R14	R15	R2	R3	R4	R5	R6	R7	R8
	R9	SAVEAREA	SAVEWRK1	SAVEWRK2	SAVEWRK4	SAVEWRK5	SAVEWRK6	SAVEWRK8	VMBLCK	VMECEXT	VMESTAT	VMEXTCM
	VMFPRS	VMGPRS	VMINVPAG	VMINVSEG	VMLOGOFF	VMNEWCR0	VMOSTAT	VMPSTAT	VMPSW	VMRSTAT	VMSEG	VMSHR
	VMSIZE	VMVCR0	VMV370R	XPAGNUM	XRIGHT16	ZEROES						
DMKCDS	ACORETBL	BLANKS	BRING	CORFLAG	CORPGPNT	CORSHARE	CORSWPNT	CORTABLE	CPEXADD	CPEXBLOK	CPEXFPNT	CPEXR0
	CPEXR14	CPEXR15	CPEXR5	CPEXR7	CPEXSIZE	DEFER	DMKCVTBH	DMKCVTDB	DMKCVTHB	DMKERMSG	DMKFREE	DMKPAGIO
	DMKPSACC	DMKPSASC	DMKPTRAN	DMKPTRWQ	DMKQCNWT	DMKSCNFD	DMKSYSRM	DMKTRCIT	DMKTRCPB	DMKVATAB	DMKVATBC	DMKVATMD
	DMKVMACF	DMKVMAPS	ECBLOK	EXTCCTRQ	EXTCPTMR	EXTCR0	EXTMODE	F1	F15	F16	F2	F4
	F5	F6	F8	NORET	PAGCORE	PAGINVAL	PSA	RUNUSER	R0	R1	R11	R13
	R14	R15	R2	R3	R4	R5	R6	R7	R8	SAVEAREA	SAVER2	SAVEWRK1
	SAVEWRK2	SAVEWRK3	SAVEWRK4	SAVEWRK5	SAVEWRK8	SWPFLAG	SWPTRANS	TEMPR14	TEMPR15	TRANMODE	TRQBLOK	TRQBVAL
	VMBLOK	VMECEXT	VMESTAT	VMEXTCM	VMFPRS	VMGPRS	VMINVPAG	VMINVSEG	VMNEWCR0	VMPSTAT	VMPSW	VMSIZE
	VMTIMER	VMTRBRIN	VMTRCTL	VMUSER	VMVCR0	VMV370R	WAIT	XPAGNUM	ZEROES			
DMKCFC	ATTN	BLANKS	DMKCDBDC	DMKCDBDI	DMKCDBDM	DMKCDBDU	DMKCDSCP	DMKCDSTO	DMKCFDAD	DMKCFDLO	DMKCFGIP	DMKCFGSV
	DMKCFMAT	DMKCFMWU	DMKCFSET	DMKCFTRM	DMKCPBEX	DMKCPBNR	DMKCPBRS	DMKCPBRW	DMKCPBRY	DMKCPBSR	DMKCPVAC	DMKCPVDS
	DMKCPVEN	DMKCPSH	DMKCPVLK	DMKCPSRY	DMKCPSSH	DMKCPVUL	DMKCQGEN	DMKCQPRV	DMKCQREY	DMKCSOBS	DMKCSODR	DMKCSOFL
	DMKCSOLD	DMKCSORP	DMKCSOSP	DMKCSOST	DMKCSOVL	DMKCSPCL	DMKCSPFR	DMKCSPHL	DMKCSPSP	DMKCSTAG	DMKCSUCH	DMKCSUOR
	DMKCSUPU	DMKCSUTR	DMKCVTDB	DMKCVTHB	DMKDEFIN	DMKDIACP	DMKDIAL	DMKERMSG	DMKFREE	DMKLNKIN	DMKLOGA	DMKLOGON
	DMKMCCCL	DMKMSGEC	DMKMSGMS	DMKMSGWN	DMKNETWK	DMKQCNWT	DMKSCHRT	DMKSCHST	DMKSCNFD	DMKTHIEN	DMKTRACE	DMKTRCIT
	DMKTRCPB	DMKUSODS	DMKUSOFL	DMKUSOLG	DMKVDBAT	DMKVDBDE	FFS	F1	F2	F3	F4	F6
	F60	F8	LOCK	NORET	PSA	RDEVBLOK	RDEVTYPE	R0	R1	R11	R13	R15
	R2	R3	R4	R5	R6	R7	SAVEAREA	SAVERETN	SAVER1	SAVER2	SAVEWRK2	SAVEWRK4
	SYSTEM	TRQBIRA	TRQBLOK	TRQBSIZE	TRQBUSER	TRQBVAL	TYP2741	VMBLOK	VMCLASSA	VMCLASSB	VMCLASSC	VMCLASSD
	VMCLASSE	VMCLASSF	VMCLASSG	VMCLASSH	VMCLEVEL	VMCOMND	VMDELAY	VMLOGON	VMOSTAT	VMPSW	VMRSTAT	VMSLEEP
	VMTERM	VMTRBRIN	VMTRCTL	VMVIRCF								

Module	External References (Labels and Modules)												
DMKCFD	BLANKS	BRING	DEFER	DMKCVTBH	DMKCVTHB	DMKERMSG	DMKFREE	DMKFRET	DMKPSASC	DMKPTRAN	DMKQCNWT	DMKSCNAU	DMKSCNFD
	DMKSCNRU	DMKSCNVU	DMKVMACF	F1	F3	F6	NORET	PSA	R0	R1	R10	R11	R13
	R15	R2	R3	R4	R5	R6	R7	R8	SAVEAREA	SAVEWRK1	SAVEWRK2	SAVEWRK3	SAVEWRK4
	SAVEWRK5	SAVEWRK7	VMADSTOP	VMBLOK	VMESTAT	VMMCR6	VMMICSVC	VMMSVC	VMSHRSYS	VMSIZE	ZEROES		
DMKCFG	ASYSVM	AVMREAL	BRING	BUFFER	BUFNXT	C14	C15	C2	DEFER	DMKBLDRT	DMKCFPRR	DMKCVTBH	DMKCVTDB
	DMKCVTHB	DMKERMSG	DMKFREE	DMKFRET	DMKPGSFS	DMKPGSPC	DMKPGSPP	DMKPGSPS	DMKPGTVG	DMKPGTVR	DMKPTRAN	DMKPTRUL	DMKQCNRD
	DMKQCNWT	DMKRPAGT	DMKRPAPT	DMKSCNFD	DMKSCNVS	DMKSCNVU	DMKSNTBL	DMKVATMD	DMKVMAS1	DMKVMAS2	DMKVMI	ECBLOK	EDIT
	ERRMSG	EXTCR0	EXTMASK	EXTMODE	F0	F1	F15	F2	F256	F3	F4	F4095	F4096
	F7	F8	KEEPSEGS	LOCK	NEWPAGES	NORET	OLDVMSEG	PAGCORE	PAGSHR	PAGSWP	PAGTABLE	PSA	RDEVBLOK
	RDEVCODE	RDEVFLAG	RDEVOWN	RDEVSER	RDEVTYPE	R0	R1	R10	R11	R13	R14	R15	R2
	R3	R4	R5	R6	R7	R8	R9	SAVCREGS	SAVEAREA	SAVERETN	SAVER5	SAVER6	SAVEWRK1
	SAVEWRK2	SAVEWRK3	SAVEWRK4	SAVEWRK5	SAVEWRK6	SAVEWRK7	SAVEWRK9	SAVFPRES	SAVGREGS	SAVKEYS	SAVPSW	SAVTABLE	SHRBPNT
	SHRFPNT	SHRNAME	SHRPAGE	SHRSEGCT	SHRSEGNM	SHRTABLE	SHRTSIZE	SHRUSECT	SWPCHG1	SWPCHG2	SWPCYL	SWPFLAG	SWPKEY1
	SWPSHR	SYSCYL	SYSHRSEG	SYSNAME	SYSPAGCT	SYSPAGLN	SYSPAGNM	SYSPNT	SYSSEGLN	SYSSIZE	SYSSTART	SYSTBL	SYSTEM
	SYSVADDR	SYSVOL	TRANMODE	TYP2305	TYP2314	TYP3330	TYP3340	TYP3350	UCASE	VDEVBLOK	VDEVREAL	VDEVRELN	VMABLOK
	VMAFPNT	VMANAME	VMASHRBK	VMASIZE	VMASSIST	VMBLOK	VMCOMND	VMECEXT	VMESTAT	VMEXTCM	VMFPRS	VMGPRS	VMIOWAIT
	VMMLEVEL	VMNSHR	VMOSTAT	VMPA2APL	VMPEND	VMPSTAT	VMPSW	VMPSWDCT	VMQSTAT	VMRSTAT	VMSEG	VMSHR	VMSHRSYS
	VMSIZE	VMSTOR	VMVCR0	VMV370R	VSYSRES	XRIGHT16	X40FFS						
DMKCFM	ATTN	BALRSAVE	BLANKS	BUFCNT	BUFFER	BUFINLTH	BUFNXT	BUFSIZE	CLASGRAF	CLASTERM	CPEXADD	CPEXBLOK	CPEXREGS
	CPEXSIZE	DMKCFCMD	DMKDSPB	DMKDSPCH	DMKFREE	DMKFRET	DMKQCNRD	DMKQCNWT	DMKSCHRT	DMKSCNFD	DMKSTKCP	DMKVIOMK	EDIT
	IOMASK	NOAUTO	NORET	NOTIME	PSA	RDEVBLOK	RDEVTYPC	RDEVTYPE	R0	R1	R11	R12	R13
	R14	R15	R2	R3	R4	R5	R6	R7	R8	R9	SAVEAREA	SAVER11	SAVER2
	SAVEWRK6	TREXLOCK	TREXT	TREXTERM	TRQBSIZE	TYPBSC	UCASE	VCHADD	VCHBLOK	VCHCUINT	VCHCUTBL	VCUADD	VCUBLOK
	VCUDVINT	VCUDVTBL	VDEVADD	VDEVBLOK	VDEVBUSY	VDEVCHBS	VDEVINTS	VDEVPEND	VDEVSTAT	VMBLOK	VMCF	VMCFREAD	VMCFRUN
	VMCFWAIT	VMCHSTRT	VMCHTBL	VMCLASSA	VMCLASSB	VMCLASSC	VMCLASSD	VMCLASSE	VMCLASSF	VMCLASSG	VMCLASSH	VMCPWAIT	VMCUSTRT
	VMDELAY	VMDVSTRT	VMIOINT	VMIOPND	VMKILL	VMLOGOFF	VMLOGON	VMMLEVEL	VMMSTMP	VMOSTAT	VMPEND	VMPRIDSP	VMPSW
	VMQSTAT	VMRSTAT	VMSLEEP	VMSTKO	VMSYSOP	VMTERM	VMTREXT	VMVIRCF	VMVTERM	WAIT			
DMKCFP	AVMREAL	CHBWAIT	CHXBLOK	CHXCNCT	CHXFLAG	CLASGRAF	CLASSPEC	CLASTERM	CLASURI	CLASURO	CUE	DELPAGES	DMKBLDRL
	DMKBLDRT	DMKDIADR	DMKDSPCH	DMKFREE	DMKFRET	DMKIOSHA	DMKIOSQR	DMKLOCKD	DMKLOCKQ	DMKPERT	DMKPGSPO	DMKPGSPP	DMKPTRPW
	DMKQCNSY	DMKSCHRT	DMKSCNVU	DMKSTKCP	DMKSTKIO	DMKTRCPB	DMKUNTFR	DMKVATBC	DMKVCARD	DMKVDREL	DMKVIOMK	DMKVSPCO	DMKVSPCR
	ECBLOK	EXTCCTRQ	EXTCPTMR	EXTCR0	EXTCR14	EXTCR15	EXTCR2	EXTCR4	FFS	IOBCAW	IOBFLAG	IOBHIO	IOBHVC
	IOBIOER	IOBIRA	IOBLINK	IOBLOK	IOBMISC	IOBMISC2	IOBRES	IOBSIZE	IOBSPEC	IOBTIO	IOBUSER	IOERBLOK	IOEREXT
	IOERSIZE	KEEPSEGS	NEWPAGES	NEWSEGS	OLDVMSEG	PGBLOK	PGBSIZE	PGPNT	PSA	RDEVAIOB	RDEVATT	RDEVBLOK	RDEVIOER
	R0	R1	R10	R11	R12	R13	R14	R15	R2	R3	R4	R5	R6
	R7	R8	R9	SAVEAREA	SAVEWRK6	SAVEWRK8	TRQBFPNT	TRQBLOK	TRQBVAL	TYPCTCA	TYP3210	TYP3215	VCHADD
	VCHBLOK	VCHBUSY	VCHCEDEV	VCHCEPND	VCHCUINT	VCHCUTBL	VCHDED	VCHSTAT	VCONCTL	VCONRBSZ	VCONRBUF	VCONWBSZ	VCONWBUF
	VCUACTV	VCUADD	VCUBLOK	VCUBUSY	VCUCEPND	VCUCHBSY	VCUCUEPN	VCUDVINT	VCUDVTBL	VCUINTS	VCUSTAT	VDEVADD	VDEVAUCR
	VDEVBLOK	VDEVBUSY	VDEVCCW1	VDEVCFLG	VDEVCHAN	VDEVCHBS	VDEVCON	VDEVCSW	VDEVCUE	VDEVDED	VDEVDIAL	VDEVENAB	VDEVFEED
	VDEVFLAG	VDEVINTS	VDEVIOB	VDEVIOER	VDEVNRDY	VDEVPEND	VDEVREAL	VDEVSFLG	VDEVSPL	VDEVSTAT	VDEVTYPC	VDEVTYPE	VDEVUC
	VMBLOK	VMCHSTRT	VMCHTBL	VMCUSTRT	VMDSTAT	VMDVSTRT	VMECEXT	VMESTAT	VMEXWAIT	VMIDLE	VMINVPAG	VMIOACTV	VMIOINT
	VMIOPND	VMIOWAIT	VMKILL	VMLOGOFF	VMMICSVC	VMNOTRAN	VMOSTAT	VMPAGEX	VMPEND	VMPGPND	VMPGPNT	VMPSTAT	VMPSW
	VMPXINT	VMRSTAT	VMSEG	VMSIZE	VMSTOR	VMTIO	VMTRBRIN	VMTRCTL	VMUSER	VMVCR0	VMVTERM	VMV370R	WAIT
	XINTBLOK	XINTEXT	XINTSIZE	XINTSORT	XRIGHT24	ZEROES							

Label	External References (Labels and Modules)											
DMKCPS	ACORETEL	ASYSLC	AVMREAL	BLANKS	BUFCNT	BUFFER	BUFNXT	CLASTAPE	CLASURO	CORFLAG	CORRSV	CORTABLE	CPMICAVL
	CPMICON	CPSTAT2	DMKBLDEC	DMKCFPRR	DMKCVTBH	DMKCVTDB	DMKCVTDT	DMKCVTHB	DMKDMPAU	DMKDMPDV	DMKDMPSW	DMKDSPNP	DMKERMSG
	DMKFREE	DMKFRET	DMKIOEIR	DMKMCHAR	DMKMCHMS	DMKPTRRC	DMKPTRRL	DMKPTRRU	DMKQCNRD	DMKQCNWT	DMKSCHAP	DMKSCHAU	DMKSCHPG
	DMKSCHRT	DMKSCH80	DMKSCNAU	DMKSCNFD	DMKSCNRD	DMKSCNRU	DMKSYSDT	DMKSYSDW	DMKSYSLG	DMKSYSLW	DMKSYSRV	DMKSYSTM	ECBLOK
	EDIT	EXTCCTRQ	EXTCPTRQ	EXTSIZE	F1	F2	F3	F4	F5	F7	F8	IRMAND	IRMBIT1
	IRMBIT2	IRMBLOK	IRMBYT1	IRMBYT2	IRMFLG	IRMLMT	IRMOR	IRMRLADD	IRMSIZE	MCHAREA	MICBLOK	MICCREG	MICRSEG
	MICSIZE	MICVPSW	MICWORK	MODFLAG1	MOD1RETY	NOAUTO	NORET	PSA	RDEVBLOK	RDEVDED	RDEVDISA	RDEVFLAG	RDEVIRM
	RDEVSTAT	RDEVSYS	RDEVTYPC	RDEVTYPE	RDEVUSER	R0	R1	R10	R11	R13	R14	R15	R2
	R3	R4	R5	R6	R7	R8	R9	SAVEAREA	SAVEWRK1	SAVEWRK2	SAVEWRK3	SAVEWRK4	SAVEWRK5
	SAVEWRK7	SAVEWRK8	SYSLOCS	TEMPSAVE	TRQEIRA	TRQBLOK	TRQBSIZE	TRQBUSER	TYPPRT	UCASE	VMADSTOP	VMAEX	VMAEXP
	VMBLOK	VMCFRUN	VMCLASSA	VMCLASSB	VMCLASSC	VMCLASSD	VMCLASSE	VMCLASSF	VMCLASSG	VMCLEVEL	VMECEXT	VMESTAT	VMHIPRI
	VMISAM	VMMACCON	VMMADDR	VMMCODE	VMMCR6	VMMFE	VMMICRO	VMMICSVC	VMMIMSG	VMMLEVEL	VMMLINED	VMMLVL2	VMMSGON
	VMMSVC	VMMTEXT	VMM360	VMNOTRAN	VMOSTAT	VMPAGEX	VMPFUNC	VMPSTAT	VMPSW	VMQLEVEL	VMRON	VMRPAGE	VMSEG
	VMSTMPI	VMSTOR	VMTLEVEL	VMTON	VMTRQBLK	VMUPRIOR	VMUSER	VMVCR0	VMV370R	VMWNGON	X40FFS	ZEROES	
DMKCFT	ASYSLC	CLASGRAF	CLASSPEC	CLASTERM	DMKCVTBH	DMKCVTDB	DMKERMSG	DMKSCNFD	DMKSCNVD	DMKSYSCD	DMKSYSES	DMKSYSLD	DMKSYSLE
	F1	F2	F255	F4095	NICAPL	NICATOF	NICBLOK	NICFLAG	NICLLEN	NICPSUP	NICSIZE	NICTMCD	PSA
	RDEVAPLP	RDEVATOF	RDEVBLOK	RDEVFLAG	RDEVLLEN	RDEVNICL	RDEVPSUP	RDEVTFLG	RDEVTMCD	RDEVTYPC	RDEVTYPE	R0	R1
	R10	R11	R13	R14	R15	R2	R3	R4	R5	R6	R7	R8	SAVEAREA
	SAVEWRK1	SYSLOCS	TYPBSC	TYPTTY	TYP3277	VMBLOK	VMDVSTRT	VMMCPENV	VMMLEVEL	VMMSTMP	VMTCDEL	VMTERM	VMTESCP
	VMTLDEL	VMTLEND	VMTRMID	VMVTERM	X40FFS	ZEROES							
DMKCKP	ACNTBLOK	ACNTCCW	ACNTDATA	ACNTNEXT	ALARM	ARIOCC	ARIOCH	ARIOCT	ARIOCU	ARICDV	ARIOPR	ARIOPU	ARIORD
	ARSPPR	ASYSLC	ASYSVM	ATTN	BUSY	CAW	CC	CE	CLASDASD	CLASGRAF	CLASSPEC	CLASTAPE	CLASTERM
	CLASURI	CLASURO	CPCREG0	CPID	CSW	CUE	C0	C2	C3	DE	DEVCARD	DMKOPRWT	DMKRSPAC
	DMKRSPCV	DMKRSPDL	DMKRSPHQ	DMKRSPID	DMKRSPPR	DMKRSPPU	DMKRSPRD	DMKSAV	DMKSAVRS	DMKSYSCK	DMKSYSDT	DMKSYSLG	DMKSYSOC
	DMKSYSCW	DMKSYSRM	DMKSYSTP	DMKSYSWM	DMKTMRPT	FTR35MB	INTPR	INTTIO	IONPSW	IOOPSW	IPLPSW	NICBLOK	NICDISA
	NICDISB	NICENAB	NICFLAG	NICLGRP	NICLINE	NICSIZE	NICSTAT	NICTERM	NICTYPE	OWNDLIST	OWNDRDEV	PRNPSW	PROPSW
	PSA	RCHADD	RCHBLOK	RCHCUTBL	RCUADD	RCUBLOK	RCUCHA	RCUDVTBL	RCUPRIME	RCUSUB	RCUTYPE	RDEVACNT	RDEVADD
	RDEVAIOB	RDEVAUTO	RDEVBLOK	RDEVCLAS	RDEVCUA	RDEVDED	RDEVDISA	RDEVDISB	RDEVDRAN	RDEVENAB	RDEVFLAG	RDEVFTR	RDEVLCEP
	RDEVLNCP	RDEVMAX	RDEVMDL	RDEVNCP	RDEVNICL	RDEVRECS	RDEVSEP	RDEVSPL	RDEVSTAT	RDEVTYPC	RDEVTYPE	RECBLOK	RECCYL
	RECMAX	RECPNT	RECSIZE	RECUSED	RSPLCTL	RSPSFBLK	R0	R1	R10	R11	R12	R13	R14
	R15	R2	R3	R4	R5	R6	R7	R8	R9	SFBCLAS	SFBCOPY	SFBDATE	SFBDIST
	SFBFIRST	SFBFLAG	SFBFLAG2	SFBLAST	SFBLOK	SFBORIG	SFBPNT	SFBPURGE	SFBRECER	SFBRECS	SFBSIZE	SFBSTART	SFBUSER
	SHQBSIZE	SILI	SKIP	STARTIME	SYSLOCS	TYPBSC	TYPPRT	TYPPUN	TYP2305	TYP2314	TYP3210	TYP3277	TYP3284
	TYP3330	TYP3340	TYP3350	TYP3705	UC	USERCARD	VCHBLOK	VCHCUTBL	VCUBLOK	VCUDVTBL	VDEVBLOK	VDEVCLAS	VDEVCOPY
	VDEVDED	VDEVEXTN	VDEVFLAG	VDEVSFLG	VDEVSPL	VDEVSTAT	VDEVTDSK	VDEVTYPC	VDEVTYPE	VDEVXFER	VMBLOK	VMCHSTRT	VMCHTBL
	VMCUSTRT	VMDIST	VMDVSTRT	VMLOGON	VMPNT	VMRSTAT	VMUSER	VSPLCTL	VSPSFBLK	VSPXBLOK	VSPXXUSR		

Module	External References (Labels and Modules)											
DMKCKS	ACTSFB	ADDSFB	ARIODV	ARSPPR	ARSPPU	ARSPRD	BRING	CHGSFB	DEFER	DELSFB	DMKCVTBD	DMKERMSG
	DMKFREE	DMKFRET	DMKLOCKD	DMKLOCKQ	DMKLOCKT	DMKPGTTU	DMKPGTVG	DMKPGTVR	DMKPTRAN	DMKPTRUL	DMKQCNSY	DMKRPAGT
	DMKRPAPT	DMKRSPHQ	DMKRSPID	DMKSCNRD	DMKSCNRU	DMKSYSCH	DMKSYSCN	DMKSYSOW	FFS	F1	F10	F24
	F255	F3	F4	LOCK	OPNSFB	OWNDLIST	OWNDRDEV	OWNDVSER	PCHCHN	PRTCHN	PSA	RDEVALLN
	RDEVBLOK	RDEVCLAS	RDEVCODE	RDEVDISA	RDEVDRAN	RDEVFLAG	RDEVRECS	RDEVSER	RDEVSPL	RDEVSTAT	RDEVTYPE	RDRCHN
	RECBLOK	RECCYL	RECMAP	RECMAX	RECPNT	RECSIZE	RECUSED	R0	R1	R10	R11	R13
	R14	R15	R2	R3	R4	R5	R6	R7	R8	R9	SAVEAREA	SAVER1
	SAVER2	SAVER8	SAVEWRK1	SAVEWRK2	SAVEWRK3	SAVEWRK4	SAVEWRK5	SAVEWRK6	SAVEWRK7	SAVEWRK8	SAVEWRK9	SFBCOPY
	SFBDUMP	SFBEOF	SFBFILID	SFBFLAG	SFBFLAG2	SFBINUSE	SFBLAST	SFBLOK	SFBOPEN	SFBPNT	SFBRECER	SFBRECNO
	SFBRECS	SFBRSTRT	SFBSIZE	SFBSTART	SFBTIME	SFBUHOLD	SHQBLOK	SHQBSIZE	SHQUSER	SPLINK	SPNXTPAG	SPRECNUM
	SYSIPLDV	SYSTEM	TYP2314	TYP3330	TYP3350	VMBLOK	ZEROES					
DMKCNS	ALARM	ASYSVM	ATTN	BALRSAVE	BALR3	BALR6	BALR9	BLANKS	BRING	BUSY	CAW	CC
	CCC	CD	CDC	CE	CHC	CLASTERM	CMDREJ	CONACTV	CONADDR	CONCCW1	CONCCW2	CONCCW3
	CONCCW4	CONCNT	CONCNTL	CONCOMND	CONDATA	CONESCP	CONFLAG	CONOUTPT	CONPARM	CONPNT	CONRESP	CONRETN
	CONRTRY	CONSPLT	CONSTAT	CONSYNC	CONTASK	CONTSIZE	CONTSKSZ	CONUSER	CPID	CSW	DATACHK	DE
	DEFER	DMKBLDVM	DMKCFMAT	DMKCFMBK	DMKCPIEM	DMKCVTBH	DMKDSPCH	DMKERMSG	DMKFREE	DMKFRET	DMKIOERR	DMKIOEST
	DMKIOSQR	DMKMSWR	DMKPTRAN	DMKQCNCL	DMKQCNET	DMKQCNTO	DMKSCNRD	DMKSCNRU	DMKTBLCI	DMKTBLCO	DMKTBLPI	DMKTBLPO
	DMKTBLTI	DMKTBLTO	DMKTBLUP	DMKTBMMI	DMKTBMMO	DMKTBMNI	DMKTBMNO	DMKTRMID	EDIT	FFS	F1	F10
	F15	F16	F2	F256	F4	F8	IFCC	IL	INHIBIT	INTREQ	IOBCAW	IOBCC1
	IOBCC3	IOBCSW	IOBERP	IOBFATAL	IOBFLAG	IOBIOER	IOBIRA	IOBLINK	IOBLOK	IOBRADD	IOBRES	IOBSIZE
	IOBSPEC	IOBSTAT	IOBUNSL	IOBUSER	IOERBLOK	IOERDATA	IOEREXT	IOERFLG3	IOERNUM	IOEROVFL	IOERREAD	IOERSIZE
	LOGDROP	LOGHOLD	NOAUTO	PRGC	PRIORITY	PRTC	PSA	RDEVACTV	RDEVATNC	RDEVATOF	RDEVBLOK	RDEVCON
	RDEVCORR	RDEVCTL	RDEVDISA	RDEVDISB	RDEVENAB	RDEVEPMD	RDEVFLAG	RDEVHIO	RDEVIDNT	RDEVIOER	RDEVLOG	RDEVNRDY
	RDEVPREP	RDEVPSUP	RDEVPTTC	RDEVRCNT	RDEVREST	RDEVSADN	RDEVSTAT	RDEVTFLG	RDEVTMCD	RDEVTYPC	RDEVTYPE	RDEVUSC8
	RDEVUSER	R0	R1	R10	R11	R12	R13	R14	R15	R2	R3	R4
	R5	R6	R7	R8	R9	SAVEAREA	SAVER0	SAVER1	SAVER2	SILI	SKIP	SM
	SYSTEM	TEMPR0	TEMPSAVE	TRACBEF	TRACCURR	TRACEND	TRACFLG2	TRACSTRT	TYPTTY	TYPUNDEF	TYP1050	TYP2741
	TYP3210	UC	UCASE	UE	VMBLOK	VMCF	VMCFWAIT	VMLOGOFF	VMLOGON	VMMCPENV	VMMLEVEL	VMOSTAT
	VMRSTAT	VMSYSOP	VMTCDEL	VMTLEND	VMTTIME							
DMKCPB	ASYSVM	BLANKS	BRING	CLASSPEC	CLASTAPE	CLASTERM	CLASURI	CLASURO	DE	DEFER	DMKCFPRD	DMKCFPRR
	DMKCVTBH	DMKCVTHB	DMKDSPCH	DMKERMSG	DMKFREE	DMKFRET	DMKIOSQR	DMKIOSRW	DMKPGSPO	DMKPTRAN	DMKQCNWT	DMKSCNFD
	DMKSCNVU	DMKVATBC	DMKVATMD	DMKVIOMK	EXTMODE	F3	F4	F6	IOBCAW	IOBIRA	IOBLOK	IOBMISC
	IOBSIZE	IOBUSER	NORET	PSA	RCHBLOK	RCUBLOK	RCUCHA	RCUPRIME	RCUSUB	RCUTYPE	RDEVBLOK	RDEVBUSY
	RDEVCUA	RDEVSTAT	R0	R1	R10	R11	R12	R13	R15	R2	R3	R4
	R5	R6	R7	R8	SAVEAREA	SAVERETN	SAVEWRK2	SAVEWRK4	SAVEWRK5	SILI	SYSTEM	TRANMODE
	TYPCTCA	TYP3210	VCHADD	VCHBLOK	VCHCUINT	VCUADD	VCUBLOK	VCUDVINT	VDEVADD	VDEVBLOK	VDEVBUSY	VDEVDED
	VDEVINTS	VDEVNRDY	VDEVPEND	VDEVREAL	VDEVSTAT	VDEVTYPC	VDEVTYPE	VMBLOK	VMESTAT	VMEXTCM	VMIOINT	VMIOPND
	VMPA2APL	VMPEND	VMPSTAT	VMPSW	VMPXINT	VMQSTAT	VMV370R	XINTBLOK	XINTCODE	XINTNEXT	XINTSIZE	XINTSORT
	X40FFS	ZEROES										

Module	External References (Labels and Modules)												
DMKCPI	ACORETBL	ALARM	ALOCBLOK	ALOCCYL1	ALOCCYL2	ALOCMAP	ALOCMAX	ALOCPNT	ALOCUSED	APAGCP	ARIOCH	ARIOCT	ARIOCU
	ARIODV	ASYSOP	ASYSVM	BALRSAVE	BALR0	BALR1	BALR14	BALR2	BALR6	BALR8	BLKMPX	BRING	BUFCNT
	BUFFER	BUFIN	BUFNXT	BUFSIZE	BUSY	CAW	CC	CCC	CE	CKCMASK	CLASDASD	CLASGRAF	CLASTAPE
	CLASTERM	CORCFLCK	CORCP	CORFLAG	CORFPNT	CORFREE	CORSWPNT	CORTABLE	CPCREG0	CPEXSIZE	CPID	CPMICAVL	CPMICON
	CPSTATUS	CPSTAT2	CPUID	CPULOG	CPUMCDEL	CPUVERSN	CPWAIT	CSW	CUE	C0	C1	C14	C6
	DAMAGRPT	DATE	DE	DEFER	DMKBLDRT	DMKCFMEN	DMKCKP	DMKCNSEN	DMKCPEID	DMKCPEND	DMKCPVAE	DMKCQRFI	DMKCSOSD
	DMKCVTBD	DMKCVTBH	DMKCVTDT	DMKDMPAU	DMKDMPDV	DMKDMPRC	DMKDMPSF	DMKDSPCH	DMKDSPNP	DMKFREE	DMKFREHI	DMKFRELG	DMKFRELO
	DMKFRESV	DMKFRET	DMKFRETR	DMKIOEFL	DMKIOSIN	DMKIOSQR	DMKLOGOP	DMKMCHIN	DMKNETAE	DMKNLDR	DMKPAGHI	DMKPAGLO	DMKPAGST
	DMKPGTBN	DMKPGTPG	DMKPGTP0	DMKPGTP4	DMKPGTP5	DMKPGTTM	DMKPGTTU	DMKPGTT0	DMKPGTT4	DMKPGTT5	DMKPGT4P	DMKPGT4T	DMKPGT5P
	DMKPGT5T	DMKPGT90	DMKPRGIN	DMKPSADU	DMKPSAEX	DMKPSAHI	DMKPSALO	DMKPSANS	DMKPSASV	DMKPTRAN	DMKPTRFA	DMKPTRFN	DMKPTRF1
	DMKPTRLK	DMKPTRUL	DMKPTRU1	DMKQCNRD	DMKQCNWT	DMKRIOCN	DMKRIORN	DMKRPAPT	DMKSAV	DMKSCHLI	DMKSCHMD	DMKSCHQ1	DMKSCHQ2
	DMKSCHST	DMKSCHTI	DMKSCNRD	DMKSCNRU	DMKSCNVS	DMKSYM	DMKSYMTB	DMKSYSDU	DMKSYSDW	DMKSYSNU	DMKSYSOC	DMKSYSOW	DMKSYSRM
	DMKSYSRV	DMKSYSTI	DMKSYSTZ	DMKSYSUD	DMKSYSVL	DMKUDRBV	DMKVMI	DMKWRMST	EDIT	EXNPSW	EXTMODE	FFS	FTRRPS
	FTR35MB	FTR7CMB	F0	F1	F10	F2	F3	F4	F4096	F5	F7	F8	F9
	HARDSTCP	IDLEWAIT	IFCC	INTMASK	INTREQ	IOBCAW	IOBIRA	IOBLOK	IOBSIZE	IOBUSER	IONTWAIT	IPLCCW1	IPLPSW
	IPUADDR	KEYMASK	MCHEK	MCNPSW	NEWPAGES	NEWSEGS	NICBLOK	NICDISA	NICNAME	NICSIZE	NICSTAT	NOAUTO	NORET
	NOTIME	OWNDLIST	OWNDPREF	OWNDRDEV	OWNDVSER	PAGCORE	PAGEWAIT	PAGE4K	PRNPSW	PROBTIME	PROPSW	PSA	PSENDCLR
	RCHADD	RCHBLOK	RCHCUTBL	RCHDISA	RCHSTAT	RCUADD	RCUBLOK	RCUDISA	RCUDVTBL	RCUPRIME	RCUSTAT	RCUSUB	RCUTYPE
	RDEVADD	RDEVAIOB	RDEVALLN	RDEVATOF	RDEVAUTO	RDEVBLOK	RDEVCODE	RDEVDISA	RDEVENAB	RDEVFLAG	RDEVFTR	RDEVIDNT	RDEVMAX
	RDEVMDI	RDEVNICL	RDEVNRDY	RDEVOWN	RDEVPNT	RDEVPREF	RDEVPTTC	RDEVRUN	RDEVSER	RDEVSTAT	RDEVSYS	RDEVTFLG	RDEVTMCD
	RDEVTYPC	RDEVTYPE	RDEVUSER	RECBLOK	RECCYL	RECMAP	RECMAX	RECPNT	RECSIZE	RECUSED	RUNCR0	RUNCR1	RUNUSER
	R0	R1	R10	R11	R12	R13	R14	R15	R2	R3	R4	R5	R6
	R7	R8	R9	SAVESIZE	SEGPAGE	SFBLOK	SFBORIG	SFBSTART	SFBUSER	SILI	SM	STARTIME	SVCNPSW
	SWPCHG1	SWPCHG2	SWPCYL	SWPFLAG	SYNCLOG	SYSIPLDV	SYSTEM	TEMPR0	TEMPR14	TEMPR15	TEMPR2	TEMPR3	TEMPR4
	TEMPR5	TEMPSAVE	TIMER	TODATE	TRACCURR	TRACEFLG	TRACEND	TRACSTRT	TRQBIRA	TRQBLOK	TRQBSIZE	TRQBTOD	TRQBUSER
	TRQBVAL	TYPBSC	TYP2305	TYP2314	TYP2741	TYP3066	TYP3210	TYP3277	TYP3330	TYP3340	TYP3350	UC	UCASE
	VMBLOK	VMLOGON	VMMFE	VMMSVC	VMPAGES	VMRSTAT	VMSEG	VMSIZE	VMTERM	VMUSER	WAIT	XPAGNUM	XRIGHT16
	ZEROES												
DMKCPS	ARIOCH	ARIOCT	ARIOCU	ARIODV	ASYSOP	ASYSVM	CFSTOP	CLASDASD	CLASGRAF	CLASTAPE	CLASTERM	CLASURI	CLASURO
	CPCREG8	CPEXBLOK	CPEXR0	CPEXSIZE	CPID	C8	DE	DFRET	F2	F3	IOBCC3	IOBCP	IOBCSW
	IOBFLAG	IOBHIO	IOBHVC	IOBIOER	IOBIRA	IOBLINK	IOBLOK	IOBMISC	IOBMISC2	IOBRADD	IOBSIZE	IOBSPEC	IOBSTAT
	IOBTIO	IOBUNSL	IOBUSER	IOBVADD	IOERBLOK	IOERDATA	IOEREXT	IOERSIZE	MONAIOB	MONARDB	MONCOM	MONFLAG1	MONUSER
	NICSIZE	NORET	PRIORITY	PSA	RCHBLOK	RCHCUTBL	RCHDISA	RCHSTAT	RCUBLOK	RCUDISA	RCUDVTBL	RCUSTAT	RDEVADD
	RDEVAIOB	RDEVBLOK	RDEVBUSY	RDEVCTRS	RDEVDED	RDEVDISK	RDEVDRAN	RDEVENAB	RDEVEPLN	RDEVFLAG	RDEVIOER	RDEVLCEP	RDEVLNCP
	RDEVLNKS	RDEVMAX	RDEVMOUT	RDEVNICL	RDEVNRDY	RDEVOWN	RDEVRCVY	RDEVRSVD	RDEVSCED	RDEVSPL	RDEVSTAT	RDEVSYS	RDEVTYPC
	RDEVTYPE	RDEVUSER	R0	R1	R10	R11	R12	R13	R14	R15	R2	R3	R4
	R5	R6	R7	R8	SAVEAREA	SAVER11	SAVEWRK1	SAVEWRK2	SAVEWRK3	SAVEWRK4	SAVEWRK9	TBUSY	
	TYPBSC	TYP2305	TYP3705	VDEVBLOK	VDEVSPL	VMBLOK	VMDVSTRT	VMTTIME	VMUSER	VMVTERM	X40FFS	ZEROES	

Module	External References (Labels and Modules)
DMKCPV	ACORETBL ARIOCH ARIOCT ARIOCU ARIODV ASYSOP ASYSVM AVMREAL BALRSAVE BRING CLASDASD CLASGRAF
	CLASTERM CORCFLCK CORFLAG CORFPNT CORTABLE CPEXADD CPEXBLOK CPEXREGS CPEXR12 CPEXSIZE DEFER F1 F2
	F3 F4096 F8 F9 LOCK NORET PSA RCHBLOK RCHCUTBL RCUBLOK RCUDVTBL RDEVBLOK RDEVDED
	RDEVDISA RDEVDISB RDEVENAB RDEVFLAG RDEVLCG RDEVSTAT RDEVTFLG RDEVTYPC RDEVTYPE RDEVUSER R0 R1 R10
	R11 R13 R14 R15 R2 R3 R4 R5 R6 R7 R8 R9 SAVEAREA
	SAVEREGS SAVER11 SAVEWRK1 SAVEWRK2 SAVEWRK3 SAVEWRK4 SAVEWRK3 SAVEWRK4 SAVEWRK8 SYSTEM TYPTTY TYP3066 TYP3277
	TYP3284 VCHBLOK VCHCUTBL VCUBLOK VCUDVTBL VDEVBLOK VDEVDED VDEVFLAG VDEVSTAT VDEVTDSK VDEVTYPC
	VMBLOK VMCHSTRT VMCHTBL VMCUSTRT VMDVSTRT VMLOGOFF VMLOGON VMMACCON VMMLEVEL VMPNT VMRSTAT VMSEG VMSIZE
	VMSTOR VMTTIME VMUSER VMWSPROJ X40FFS ZEROES
DMKCQG	ARSPPR ARSPPU ARSPRD BLANKS CHXBLOK CHXOTHR CHXYADD CLASDASD CLASGRAF CLASSPEC CLASTAPE CLASTERM CLASURI
	CLASURO DMKCVTBD DMKCVTBH DMKCVTDB DMKCVTHB DMKERMSG DMKFREE DMKFRET DMKQCNWT DMKSCNAU DMKSCNFD DMKSCNRD DMKSCNRN
	DMKSCNVN DMKSCNVU F1 F2 F3 F8 NORET PSA RDEVBLOK RDEVSER RDEVTYPC RDEVTYPE R0
	R1 R10 R11 R12 R13 R15 R2 R3 R4 R5 R6 R7 R8
	R9 SAVEAREA SAVER0 SAVER11 SAVEWRK1 SAVEWRK2 SAVEWRK3 SAVEWRK4 SAVEWRK5 SAVEWRK8 SFBCLAS SFBCOPY SFBDATE
	SFBDIST SFBDUMP SFBFILID SFBFLAG SFBFNAME SFBFTYPE SFBINUSE SFBLOK SFBORIG SFBPNT SFBRECNO SFBSHOLD SFBTIME
	SFBTYPE SFBUHOLD SFBUSER TYPBSC TYPCTCA TYPPRT TYPRDR TYPTIMER TYP2305 TYP2311 TYP2314 TYP3210 TYP3330
	TYP3340 TYP3350 VCHADD VCHBLOK VCHCUTBL VCUADD VCUBLOK VCUDVTBL VDEVADD VDEVBLOK VDEVBND VDEVCLAS VDEVCONT
	VDEVCOPY VDEVCSPL VDEVDED VDEVENAB VDEVEOF VDEVEXTN VDEVFLAG VDEVFOR VDEVHOLD VDEVNRDY VDEVRDO VDEVREAL VDEVSFLG
	VDEVSTAT VDEVTDSK VDEVTERM VDEVTYPC VDEVTYPE VDEVXFER VDEV231B VDEV231T VMBLOK VMCHSTRT VMCHTBL VMCLASSD VMCLASSG
	VMCLEVEL VMCUSTRT VMDISC VMDIST VMDVSTRT VMFBMX VMFSTAT VMOSTAT VMSTKO VMSTOR VMTERM VMTRMID VMUSER
	VSPXBLOK VSPXXUSR ZEROES
DMKCQP	ALARM ARIOCH ARIOCT ARIOCU ARIODV ASYSVM BLANKS CLASDASD CLASGRAF CLASSPEC CLASTAPE CLASTERM CLASURI
	CLASURO DMKCVTBD DMKCVTBH DMKCVTHB DMKERMSG DMKFREE DMKFRET DMKQCNWT DMKRIORN DMKRSPUR DMKSCNAU DMKSCNFD DMKSCNRD
	DMKSCNRN DMKSCNRU DMKSCNVD DMKSCNVU DMKSYSRM DMKSYSRV ERRMSG FFS F1 F2 F3 F6 NORET
	PSA RCHBLOK RCHCUTBL RCUBLOK RCUDVTBL RDEVACNT RDEVADD RDEVATT RDEVAUTO RDEVBLOK RDEVCLAS RDEVDED RDEVDISA
	RDEVDRAN RDEVENAB RDEVFLAG RDEVLCEP RDEVLNCP RDEVLNKS RDEVMOUT RDEVNCP RDEVNRDY RDEVOWN RDEVSEP RDEVSER RDEVSLOW
	RDEVSPL RDEVSTAT RDEVSYS RDEVTYPC RDEVTYPE RDEVUSER RSPLCTL RSPSFBLK R0 R1 R10 R11 R12
	R13 R14 R2 R3 R4 R5 R6 R7 R8 R9 SAVEAREA SAVEREGS SAVER0
	SAVER1 SAVER11 SAVER2 SAVEWRK1 SAVEWRK2 SAVEWRK3 SAVEWRK5 SAVEWRK6 SAVEWRK8 TYPBSC TYP2305 TYP3705 VDEVADD
	VDEVBLOK VDEVBND VDEVFLAG VDEVRDO VDEVREAL VDEVRELN VDEVSIZE VDEVTDSK VDEVTYPC VMBLOK VMDISC VMDVCNT VMDVSTRT
	VMOSTAT VMPNT VMSTKO VMTERM VMTRMID VMUSER XRIGHT16 ZEROES
DMKCQR	ARIODV ARSPPR ARSPPU ARSPRD BLANKS CLASDASD CLASSPEC CLASTAPE CLASTERM CPMICON CPSTAT2 DFRET DMKACOTM
	DMKCVTBD DMKCVTBH DMKCVTDB DMKCVTDT DMKDMPDV DMKDMPSW DMKERMSG DMKFREE DMKFRET DMKPAGQR DMKPTRFF DMKPTRSS DMKQCNWT
	DMKRIOPR DMKRSPHQ DMKRSPPR DMKRSPPU DMKRSPRD DMKSCHPG DMKSCNAU DMKSCNFD DMKSCNRD DMKSYSDW DMKSYSLG DMKSYSND DMKSYSNM
	DMKSYSTI F0 F1 F2 F4095 F60 F8 NICAPL NICATOF NICBLOK NICFLAG NICLLEN NICPSUP
	NICSIZE NICTMCD NORET PAGEWAIT PSA RDEVAPLP RDEVATOF RDEVBLOK RDEVFLAG RDEVLLEN RDEVNICL RDEVPSUP RDEVTFLG
	RDEVTMCD RDEVTYPC RDEVTYPE R0 R1 R10 R11 R12 R13 R14 R15 R2 R3
	R4 R5 R6 R7 R8 R9 SAVEAREA SAVER0 SAVEWRK1 SAVEWRK2 SAVEWRK4 SAVEWRK8 SFBCLAS
	SFBFLAG SFBINUSE SFBLOK SFBPNT SFBSHOLD SFBUHOLD SFBUSER SHQBLOK SHQSHOLD SHQUSER STARTIME TEMPSAVE TYPBSC
	TYPPRT TYPPUN VMBLOK VMCFRUN VMCLASSA VMCLASSB VMCLASSD VMCLASSE VMCLEVEL VMDISC VMISAM VMMACCON VMMCODE
	VMMCPENV VMMCR6 VMMFE VMMIMSG VMMLEVEL VMMLINED VMMLVL2 VMMSGON VMMSVC VMMTEXT VMOSTAT VMPAGEX VMPFUNC
	VMPNT VMPSTAT VMRON VMSTKO VMTCDEL VMTERM VMTESCP VMTLDEL VMTLEND VMTLEVEL VMTON VMTRMID VMUPRIOR
	VMUSER VMV370R VMWNGON ZEROES

Module	External References (Labels and Modules)												
DMKCSO	ACORETBL	APTRAN	ARIODV	ARIOPR	ARIOPU	ARIORD	ASYSVM	BALRSAVE	BLANKS	BRING	BUFFER	BUFNXT	CC
	CHGRDV	CLASURI	CLASURO	C1	DE	DEFER	DMKCKSPL	DMKCVTBH	DMKCVTDB	DMKCVTHB	DMKDSPCH	DMKERMSG	DMKFCBLD
	DMKFREE	DMKFRET	DMKIOSQR	DMKPTRUL	DMKQCNWT	DMKRSPEX	DMKSCNFD	DMKSCNRU	DMKSCNVU	DMKSPLDL	DMKSTKIO	DMKUCBLD	DMKUCSLD
	FTRUCS	F1	F2	F3	F4	F8	IOBCAW	IOBCP	IOBCSW	IOBFATAL	IOBFLAG	IOBIRA	IOBLINK
	IOBLOK	IOBMISC	IOBMISC2	IOBRADD	IOBRSTRT	IOBSIZE	IOBSTAT	IOBUSER	LOCK	NORET	OPERATOR	PSA	RDEVACNT
	RDEVAIOB	RDEVBACK	RDEVBLOK	RDEVBUSY	RDEVCLAS	RDEVDED	RDEVDISA	RDEVDRAN	RDEVFLAG	RDEVFTR	RDEVIOER	RDEVLOAD	RDEVNRDY
	RDEVRSTR	RDEVSEP	RDEVSPAC	RDEVSPL	RDEVSTAT	RDEVTERM	RDEVTYPC	RDEVTYPE	RDEVUSER	RSPLCTL	RSPMISC	RSPSFBLK	R0
	R1	R10	R11	R12	R13	R14	R15	R2	R3	R4	R5	R6	R7
	R8	R9	SAVEAREA	SAVEWRK1	SAVEWRK2	SAVEWRK4	SAVEWRK5	SAVEWRK6	SAVEWRK7	SAVEWRK8	SFBCOPY	SFBFLAG	SFBFLAG2
	SFBLOK	SFBRECER	SFBRECOK	SFBREQUE	SFBSHOLD	SILI	SKIP	SYSTEM	TYPPRT	TYPPUN	TYP3211	UE	VDEVBLOK
	VDEVDED	VDEVFCBK	VDEVSTAT	VDEVTYPE	VFCBBLOK	VFCBCNT	VFCBLOAD	VFCBNDEX	VFCBSIZE	VMBLOK	VMOSTAT	VMSEG	VMSYSOP
	VMUSER	ZEROES											
DMKCSP	ARSPPR	ARSPPU	ARSPRD	BLANKS	BUFFER	CHGSFB	CHGSHQ	CLASTERM	CLASURI	CLASURO	DE	DELSFB	DMKCKSPL
	DMKCSOSD	DMKCVTDB	DMKCVTHB	DMKERMSG	DMKFREE	DMKFRET	DMKRSPHQ	DMKSCNFD	DMKSCNVU	DMKSTKIO	DMKUDRFU	DMKVIOIN	DMKVSPCO
	DMKVSPCR	FFS	F1	F2	F3	F4	F7	F8	IOBCSW	IOBIRA	IOBLINK	IOBLOK	IOBSIZE
	IOBUSER	IOBVADD	PSA	R0	R1	R10	R11	R13	R14	R2	R3	R4	R5
	R6	R7	R8	R9	SAVEAREA	SAVEWRK1	SAVEWRK2	SAVEWRK4	SAVEWRK5	SAVEWRK6	SAVEWRK8	SAVEWRK9	SFBCLAS
	SFBDIST	SFBFILID	SFBFLAG	SFBFLAG2	SFBFNAME	SFBHOLD	SFBINUSE	SFBLOK	SFBNOHLD	SFBSHOLD	SFBUHOLD	SFBUSER	SHQBLOK
	SHQBSIZE	SHQFLAGS	SHQPNT	SHQSHOLD	SHQUSER	TEMPR2	TYPPRT	TYPPUN	TYPRDR	TYP3210	UDIRBLOK	UDIRPASS	UDIRSIZE
	VCHADD	VCHBLOK	VCHCUTBL	VCUADD	VCUBLOK	VCUDVTBL	VDEVADD	VDEVBLOK	VDEVCLAS	VDEVCONT	VDEVCOPY	VDEVCSPL	VDEVCSW
	VDEVDED	VDEVEOF	VDEVEXTN	VDEVFLAG	VDEVFOR	VDEVHOLD	VDEVPEND	VDEVPURG	VDEVSFLG	VDEVSIZE	VDEVSPL	VDEVSTAT	VDEVTERM
	VDEVTYPC	VDEVTYPE	VDEVXFER	VMBLOK	VMCHSTRT	VMCHTBL	VMCUSTRT	VMDIST	VMDVCNT	VMDVSTRT	VMUSER	VSPLCTL	VSPSFBLK
	VSPXBLOK	VSPXDIST	VSPXLEN	VSPXSIZE	VSPXSPAR	VSPXTGLN	VSPXXUSR	ZEROES					
DMKCST	ARSPRD	BRING	BUFCNT	BUFFER	CLASTERM	CLASURI	CLASURO	DMKCVTBH	DMKCVTDB	DMKCVTHB	DMKERMSG	DMKFREE	DMKFRET
	DMKPGTVG	DMKPGTVR	DMKRPAGT	DMKRPAPT	DMKSCNFD	DMKSCNVD	DMKSCNVN	DMKSCNVU	FFS	F1	F2	F3	PSA
	R0	R1	R10	R11	R13	R14	R2	R3	R4	R5	R6	R7	R8
	R9	SAVEAREA	SAVEWRK1	SAVEWRK2	SAVEWRK4	SAVEWRK5	SAVEWRK6	SAVEWRK7	SFBFILID	SFBLOK	SFBPNT	SFBSTART	SFBUSER
	SKIP	SYSTEM	TYPPRT	TYPPUN	TYPRDR	TYP3210	VDEVADD	VDEVBLOK	VDEVDED	VDEVEXTN	VDEVFOR	VDEVSFLG	VDEVSIZE
	VDEVSTAT	VDEVTYPC	VDEVTYPE	VDEVXFER	VMBLOK	VMDIST	VMDVCNT	VMDVSTRT	VMSTKO	VMUSER	VSPXBLOK	VSPXDIST	VSPXLEN
	VSPXSIZE	VSPXSPAR	VSPXTAG	VSPXTGLN	VSPXXUSR	ZEROES							
DMKCSU	ARSPPR	ARSPPU	ARSPRD	BLANKS	BUFFER	BUFNXT	CHGSFB	CLASURI	DE	DMKCKSPL	DMKCSOSD	DMKCVTBD	DMKCVTDB
	DMKERMSG	DMKFREE	DMKFRET	DMKQCNWT	DMKSCNAU	DMKSCNFD	DMKSPLDL	DMKSTKIO	DMKUDRFU	DMKVIOIN	FFS	F1	F2
	F24	F3	F4	F5	F6	F7	F8	IOBCSW	IOBIRA	IOBLINK	IOBLOK	IOBSIZE	IOBUSER
	IOBVADD	NORET	PSA	R0	R1	R10	R11	R12	R13	R14	R15	R2	R3
	R4	R5	R6	R7	R8	R9	SAVEAREA	SAVER11	SAVEWRK1	SAVEWRK2	SAVEWRK4	SAVEWRK5	SAVEWRK6
	SAVEWRK8	SAVEWRK9	SFBCLAS	SFBCOPY	SFBDIST	SFBFILID	SFBFLAG	SFBFNAME	SFBINUSE	SFBLOK	SFBORIG	SFBPNT	SFBSHOLD
	SFBUHOLD	SFBUSER	TEMPR2	TEMPR3	TEMPR4	TYPPRT	TYPPUN	TYPRDR	VCHADD	VCHBLOK	VCHCUTBL	VCUADD	VCUBLOK
	VCUDVTBL	VDEVADD	VDEVBLOK	VDEVCLAS	VDEVCSW	VDEVPEND	VDEVSPL	VDEVSTAT	VDEVTYPC	VDEVTYPE	VMBLOK	VMCHSTRT	VMCHTBL
	VMCLASSD	VMCLEVEL	VMCOMND	VMCUSTRT	VMDVSTRT	VMMIMSG	VMMLEVEL	VMMLVL2	VMMSGON	VMTTIME	VMUSER	ZEROES	

Module	External References (Labels and Modules)												
DMKCVT	BALRSAVE	BALR1	BALR2	CPID	DATE	F1	F10	F240	F4	F60	PSA	R0	R1
	R10	R14	R15	R2	R3	R5	R6	R7	R8	R9	TEMPSAVE	TODATE	
DMKDAS	ALARM	ASYSVM	CC	CCC	CD	CDC	DFRET	DMKCVTBH	DMKDSPCH	DMKFREE	DMKFRET	DMKIOESD	DMKIOEST
	DMKIOSQR	DMKMSWR	DMKQCNWT	DMKSCNRU	FTREXTSN	FTRRPS	FTR35MB	FTR70MB	F1	F10	F2	F256	F4095
	F4096	F8	IDA	IFCC	IOBCAW	IOBCC3	IOBCP	IOBCSW	IOBERP	IOBFATAL	IOBFLAG	IOBIOER	IOBIRA
	IOBLINK	IOBLOK	IOBRADD	IOBRCAW	IOBRCNT	IOBRSTRT	IOBSIZE	IOBSPEC	IOBSTAT	IOBTIO	IOBUSER	IOERACT	IOERADR
	IOERBLOK	IOERCAL	IOERCAN	IOERCEMD	IOERCSW	IOERDASD	IOERDATA	IOERDEC	IOERDW	IOERECF	IOERETRY	IOEREXT	IOERFLG1
	IOERFLG2	IOERFLG3	IOERHA	IOERIGNR	IOERIND3	IOERIND4	IOERINFO	IOERLEN	IOERLOC	IOERMSG	IOERMSW	IOERNUM	IOEROVFL
	IOERPEND	IOERPNT	IOERREAD	IOERSIZE	IOERSTAT	IOERSTRT	IOERVOL1	IOERVSER	NORET	OPERATOR	PRGC	PRTC	PSA
	RDEVBLCK	RDEVDED	RDEVDISA	RDEVFLAG	RDEVFTR	RDEVIOER	RDEVMOUT	RDEVNRDY	RDEVOWN	RDEVSER	RDEVSTAT	RDEVSYS	RDEVTYPE
	R0	R1	R10	R11	R12	R13	R15	R2	R3	R4	R5	R6	R7
	R8	R9	SAVEAREA	SAVER11	SILI	SKIP	TYP2305	TYP2314	TYP3330	TYP3340	TYP3350	UC	XRIGHT16
	ZEROES												
DMKDDR	ATTN	BUSY	CC	CD	CE	CLASDASD	CLASTAPE	CLASTERM	CUE	DE	ERRMSG	INTREQ	R0
	R1	R10	R11	R12	R13	R14	R15	R2	R3	R4	R5	R6	R7
	R8	R9	SILI	SKIP	TYP2305	TYP2311	TYP2314	TYP2319	TYP2401	TYP2415	TYP2420	TYP3330	TYP3340
	TYP3350	TYP3410	TYP3411	TYP3420	UC	UE	WAIT						
DMKDEF	CLASDASD	CLASGRAF	CLASSPEC	CLASTERM	CLASURI	CLASURO	DELPAGES	DELSEGS	DMKBLDRL	DMKBLDRT	DMKCFPRD	DMKCFPRR	DMKCVTBD
	DMKCVTBH	DMKCVTDB	DMKCVTHB	DMKERMSG	DMKFREE	DMKFRET	DMKLOCKD	DMKLOCKQ	DMKPGSPO	DMKQCNWT	DMKSCNFD	DMKSCNVD	DMKSCNVN
	DMKSCNVU	DMKUDRFU	DMKUDRRD	DMKUDRRV	DMKVCARS	DMKVDSDF	FFS	F3	F4	F5	F8	NEWPAGES	NEWSEGS
	NORET	PSA	RDEVATT	RDEVBLOK	R0	R1	R10	R11	R12	R13	R15	R2	R3
	R4	R5	R6	R7	R8	SAVEAREA	SAVER2	SAVEWRK1	SAVEWRK2	SAVEWRK3	SAVEWRK4	SAVEWRK5	SAVEWRK6
	SAVEWRK7	SAVEWRK8	SAVEWRK9	TYPCTCA	TYPIBM1	TYPPRT	TYPTELE2	TYP1052	TYP1403	TYP2305	TYP3211	UDBFBLOK	UDBFSIZE
	UDBFVADD	UDEVADD	UDEVBLOK	UDEVCLAS	UDEVDISP	UDEVFTR	UDEVNCYL	UDEVSTAT	UDEVTDSK	UDEVTYPC	UDEVTYPE	UDEV3158	UDIRBLOK
	UDIRDISP	UMACBLOK	UMACMCOR	VCHADD	VCHBLCK	VCHBMX	VCHCUTBL	VCHDED	VCHSEL	VCHSTAT	VCHTYPE	VCUADD	VCUBLOK
	VCUCTCA	VCUDVTBL	VCUTYPE	VDEVADD	VDEVBLOK	VDEVDED	VDEVFLAG	VDEVLINK	VDEVPOSN	VDEVRELN	VDEVSTAT	VDEVTDSK	VDEVTYPC
	VMBLOK	VMCHSTRT	VMCHTBL	VMCUSTRT	VMDVSTRT	VMFBMX	VMFSTAT	VMMIMSG	VMMLVL2	VMSIZE	VMSTOR	VMUSER	VMVTERM
	VRALOC												
DMKDGD	ACORETBL	BRING	CC	CD	CLASDASD	CORPGPNT	CORSWPNT	CORTABLE	CPSHRLK	CPSTAT2	CSW	DEFER	DMKDSPCH
	DMKFREE	DMKFRET	DMKIOSQV	DMKPSACC	DMKPSASC	DMKPTRAN	DMKPTRFR	DMKPTRFT	DMKPTRUL	DMKSCNVU	DMKVMAPS	FFS	F1
	F15	F16	F3	F4	F4095	F4096	F5	F6	F8	IDA	IOBCAW	IOBCC1	IOBCC3
	IOBCSW	IOBCYL	IOBFATAL	IOBFLAG	IOBHVC	IOBIOER	IOBIRA	IOBLINK	IOBLOK	IOBMISC	IOBMISC2	IOBSIZE	IOBSTAT
	IOERBLOK	IOEREXT	IOERSIZE	LOCK	PCIF	PSA	RCWADDR	RCWCCW	RCWCNT	RCWCOMND	RCWCTL	RCWFLAG	RCWIO
	RCWSHR	R0	R1	R10	R11	R12	R13	R14	R15	R2	R3	R4	R5
	R6	R7	R8	R9	SAVEAREA	SAVEREGS	SAVER12	SAVER2	SAVEWRK9	SILI	SWPFLAG	TYP2305	TYP2311
	TYP2314	TYP3330	TYP3340	TYP3350	VDEVBLOK	VDEVBND	VDEVBUSY	VDEVCHAN	VDEVDED	VDEVFLAG	VDEVIOB	VDEVIOER	VDEVPEND
	VDEVPOSN	VDEVPOST	VDEVRDO	VDEVRELN	VDEVSTAT	VDEVTYPC	VDEVTYPE	VDEVUC	VMACTDEV	VMBLOK	VMCOMP	VMDVSTRT	VMESTAT
	VMEXTCM	VMEXWAIT	VMGPRS	VMIOCNT	VMIOWAIT	VMLOGOFF	VMLOPRI	VMPSW	VMQLEVEL	VMRSTAT	XPAGNUM		

Module	External References (Labels and Modules)												
DMKDIA	ARIOCU	ARIODV	ASYSVM	BALRSAVE	BALR1	BLANKS	CC	CCDESMD	CD	CE	CHBSIZE	CHXBLOK	CHXOTHR
	CHXYADD	CHYBLOK	CHYOTHR	CHYXADD	CLASGRAF	CLASSPEC	CLASTERM	CMDREJ	CCNCCW3	CONDATA	CONDCNT	CONSYSR	CPEXADD
	CPEXBLOK	CPEXSIZE	CRESDQ	CRESIMD	CSETDSM	CSWLMEP	CSWLNCP	CTRMLTR	DE	DFRET	DMKACODV	DMKBLDVM	DMKCFPRD
	DMKCVTED	DMKCVTBH	DMKCVTHB	DMKDSPCH	DMKERMSG	DMKFREE	DMKFRET	DMKIOSHA	DMKIOSQR	DMKQCNCL	DMKQCNWT	DMKRIORN	DMKRNHND
	DMKSCNAU	DMKSCNFD	DMKSCNRD	DMKSCNRN	DMKSCNRU	DMKSCNVD	DMKSCNVU	DMKSTKCP	DMKSTKIO	DMKSYSCK	DMKSYSND	DMKSYSRM	DMKVCARS
	DMKVIOIN	FFS	F1	F240	F3	F4095	IDA	IL	INTREQ	IOBCAW	IOBCC1	IOBCP	IOBCSW
	IOBFLAG	IOBIOER	IOBIRA	IOBLINK	IOBLOK	IOBMISC	IOBRADD	IOBRCAW	IOBRSTRT	IOBSIZE	IOBSTAT	IOBUSER	IOBVADD
	IOERBLOK	IOERCCW	IOERCSW	IOERDATA	IOEREXT	IOERSIZE	LOGHOLD	NICBLOK	NICCIBM	NICDISA	NICENAB	NICEPAD	NICEPMD
	NICFLAG	NICLINE	NICLTRC	NICNAME	NICQPNT	NICSESN	NICSIZE	NICSTAT	NICSWEP	NICTELE	NICTYPE	NICUSER	NORET
	OPERATOR	PRGC	PRIORITY	PRTC	PSA	RCHBLOK	RCHCUTBL	RCUBLOK	RCUDVTBL	RCWADDR	RCWCCW	RCWCNT	RCWCOMND
	RCWCTL	RCWFLAG	RCWINVL	RDEVACTV	RDEVADD	RDEVAIOB	RDEVAIRA	RDEVATT	RDEVBASE	RDEVBLOK	RDEVCON	RDEVCORD	RDEVCTL
	RDEVCUA	RDEVCYL	RDEVDED	RDEVEPDV	RDEVEPLN	RDEVEPMD	RDEVFLAG	RDEVHIO	RDEVLCEP	RDEVLNCP	RDEVNICL	RDEVNRDY	RDEVPREP
	RDEVRCVY	RDEVRUN	RDEVSTAT	RDEVTFLG	RDEVTMAT	RDEVTYPC	RDEVTYPE	RDEVUSER	RUNUSER	R0	R1	R10	R11
	R12	R13	R14	R15	R2	R3	R4	R5	R6	R7	R8	R9	SAVEAREA
	SAVERETN	SAVER11	SAVER2	SAVER8	SAVEWRK1	SAVEWRK2	SAVEWRK3	SAVEWRK4	SAVEWRK5	SAVEWRK6	SAVEWRK7	SAVEWRK8	SAVEWRK9
	SILI	SKIP	TRQBSIZE	TYPBSC	TYPCTCA	TYPIBM1	TYPTELE2	TYP3277	UC	VCHADD	VCHBLOK	VCHCUTBL	VCUADD
	VCUBLOK	VCUDVTBL	VDEVADD	VDEVBLOK	VDEVDED	VDEVDIAL	VDEVENAB	VDEVFLAG	VDEVIOB	VDEVNRDY	VDEVREAL	VDEVSTAT	VDEVTYPC
	VDEVTYPE	VMBLCK	VMBSIZE	VMCF	VMCHSTRT	VMCHTBL	VMCUSTRT	VMDVSTRT	VMIOWAIT	VMKILL	VMLOGOFF	VMOSTAT	VMPNT
	VMRSTAT	VMTERM	VMTRMID	VMTTIME	VMUSER	ZEROES							
DMKDIR	ATTN	BUSY	CC	CD	CE	CLASDASD	CLASGRAF	CLASSPEC	CLASTERM	CLASURI	CLASURO	CUE	DE
	ERRMSG	FTR2311B	FTR2311T	R0	R1	R10	R11	R12	R13	R14	R15	R2	R3
	R4	R5	R9	SILI	TYPCTCA	TYPIBM1	TYPTELE2	TYPTIMER	TYP1052	TYP1403	TYP1443	TYP2305	TYP2311
	TYP2314	TYP2501	TYP2540P	TYP2540R	TYP3158	TYP3210	TYP3211	TYP3215	TYP3277	TYP3330	TYP3340	TYP3350	TYP3505
	TYP3525	UC	UDEVADD	UDEVBLOK	UDEVCLAS	UDEVDASD	UDEVDED	UDEVDISP	UDEVFTR	UDEVLINK	UDEVLKDV	UDEVLKID	UDEVLM
	UDEVIONG	UDEVLR	UDEVLW	UDEVMODE	UDEVMR	UDEVMW	UDEVNCYL	UDEVPASM	UDEVPASR	UDEVPASW	UDEVR	UDEVRELN	UDEVRR
	UDEVSIZE	UDEVSPOO	UDEVSTAT	UDEVTDSK	UDEVTYPC	UDEVTYPE	UDEVVSER	UDEVW	UDEVWR	UDEV3158	UDIRBLOK	UDIRDASD	UDIRDISP
	UDIRPASS	UDIRSIZE	UDIRUSER	UE	UMACACC	UMACACCT	UMACBLOK	UMACBMX	UMACCDEL	UMACCLEV	UMACCORE	UMACDASD	UMACDISP
	UMACDIST	UMACDVCT	UMACECOP	UMACES	UMACIPL	UMACISAM	UMACLDEL	UMACLEND	UMACMCOR	UMACNSVC	UMACOPT	UMACPRIR	UMACRT
	UMACSIZE	UMACVROP											
DMKDMP	ACORETBL	ALARM	ARIODV	ARSPRD	ATTN	BALR2	BUSY	CAW	CC	CE	CHGSFB	CLASDASD	CLASTAPE
	CLASURC	CORCP	CORFLAG	CORFPNT	CORTABLE	CPABEND	CPID	CSW	CUE	C0	C14	C15	C2
	DAMAGRPT	DATE	DE	DMKOPRWT	DMKPRGMC	DMKRIOPR	DMKRSPID	DMKSCNRD	DMKSCNRU	DMKSYSCH	DMKSYSCK	DMKSYSRM	DMKSYSRV
	DMPFLAG	DMPFPRS	DMPGPRS	DMPINREC	DMPKEY	DMPKYREC	DMPLCORE	DMPPGMAP	DMPSYSRV	DMPTODCK	EXTMODE	FFS	F4095
	F4096	F60	F8	HALFPAGE	HARDSTOP	INTREQ	INTTIO	IOBSIZE	ICMASK	ICNPSW	IPLCCW1	IPLPSW	MCHEK
	MONAIOB	MONARDB	MONCOM	MONFLAG2	PRNPSW	PSA	RDEVBLOK	RDEVRECS	RDEVTYPC	RDEVTYPE	RDRCHN	RECBLOK	RECCYL
	RECMAP	RECPNT	RECUSED	R0	R1	R10	R11	R12	R13	R14	R15	R2	R3
	R4	R5	R6	R7	R8	R9	SFBDATE	SFBDUMP	SFBFILID	SFBLAST	SFBLOK	SFBPNT	SFBSIZE
	SFBSTART	SFBTIME	SFBTYPE	SILI	SKIP	SYSIPLDV	TODATE	TRUN	TYPPRT	TYP1403	TYP2314	TYP3330	TYP3340
	TYP3350	UC	UE	WAIT	Y0	Y2	Y4	Y6	ZEROES				

Module	External References (Labels and Modules)												
DMKDRD	ARIODV	ARSPRD	ASYSVM	BRING	CLASURI	DEFER	DMKFREE	DMKFRET	DMKHVCPC	DMKPGTSD	DMKPGTVG	DMKPGTVR	DMKPSASP
	DMKPTRAN	DMKRPAGT	DMKSCNVU	DMKSYM	DMKSYSOW	DMKVSPCR	FFS	F1	F255	F256	F4096	F8	OWNDLIST
	OWNDRDEV	PSA	RDEVBLOK	RDEVTYPE	R0	R1	R10	R11	R12	R13	R14	R15	R2
	R3	R4	R5	R6	R7	R8	R9	SAVEAREA	SAVER0	SAVER2	SAVER6	SAVEWRK1	SAVEWRK2
	SAVEWRK3	SAVEWRK6	SFBCLAS	SFBCOPY	SFBDUMP	SFBEOF	SFBFILID	SFBFLAG	SFBINUSE	SFBLAST	SFBLOK	SFBOPEN	SFBPNT
	SFBRECER	SFBSIZE	SFBSTART	SFBTYPE	SFBUHOLD	SFBUSER	SKIP	SPLINK	SPNXTPAG	SPPREPAG	SYSTEM	TYPPRT	TYPPUN
	TYPRDR	TYP2305	TYP2319	TYP3330	TYP3340	TYP3350	VDEVBLOK	VDEVBUSY	VDEVCLAS	VDEVCONT	VDEVDIAG	VDEVSFLG	VDEVSPL
	VDEVSTAT	VDEVTYPC	VDEVTYPE	VMBLOK	VMDVSTRT	VMPSW	VMSEG	VMUSER	VSPCAW	VSPCCW	VSPDPAGE	VSPLCTL	VSPSFBLK
	VSPSIZE	XPAGNUM	X2048BND	ZEROES									
DMKDSP	ASYSVM	ATTN	BRING	CPCREG0	CPEX	CPEXADD	CPEXBLOK	CPEXBPNT	CPEXFPNT	CPEXREGS	CPEXR10	CPEXR11	CPEXSIZE
	CPMICON	CPRUN	CPSHRLK	CPSTATUS	CPSTAT2	CPWAIT	CSW	CUE	C0	C1	C11	C13	C4
	C5	C6	C7	C9	DEFER	DMKCFMBK	DMKCVTBH	DMKFREE	DMKFRET	DMKIOSER	DMKIOSRC	DMKPERT	DMKPTRAN
	DMKPTRFD	DMKPTRFE	DMKPTRFP	DMKPTRRC	DMKPTRRT	DMKQCNWT	DMKSCHDL	DMKSCHN1	DMKSCHN2	DMKSCHRL	DMKSCNVU	DMKTRCEX	DMKTRCIO
	DMKTRCIT	DMKTRCPG	DMKUSOFF	DMKVATAB	DMKVATBC	DMKVATEX	DMKVATMD	DMKVIOMK	DMKVMAPS	DMKVMASH	ECBLOK	ERRMSG	EXNPSW
	EXOPSW	EXTCR0	EXTCR2	EXTCR4	EXTCR7	EXTMODE	EXTPERAD	EXTPERCD	EXTSHCR0	EXTSHCR1	FFS	F0	F1
	IDLEWAIT	INTEX	INTEXF	INTPRL	INTTIO	IOBBPNT	IOBCSW	IOBFLAG	IOBFPNT	IOBIRA	IOBLOK	IOBPAG	IOBUSER
	IONPSW	IONTWAIT	IOOPSW	LASTUSER	MICBLOK	MICPEND	MICVIP	NORET	PAGEWAIT	PCI	PERADD	PERCODE	PERMODE
	PGADDR	PGBLCK	PGBSIZE	PGPNT	PRNPSW	PROBMODE	PROPSW	PSA	QUANTUM	QUANTUMR	RUNCR0	RUNCR1	RUNPSW
	RUNUSER	R0	R1	R10	R11	R12	R14	R15	R2	R3	R4	R5	R6
	R7	R8	R9	SIGMASK	TIMER	TRACCURR	TRACEND	TRACFLG2	TRACSTRT	TRAC0A	TRAC0C	TRAC10	TRANMODE
	TREXCR9	TREXIN1	TREXIN2	TREXT	UC	VCHADD	VCHBLOK	VCHBUSY	VCHCEDEV	VCHCEPND	VCHCUINT	VCHCUTBL	VCHSEL
	VCHSTAT	VCHTYPE	VCUADD	VCUBLOK	VCUCEPND	VCUCTCA	VCUDVINT	VCUDVTBL	VCUINTS	VCUSHRD	VCUSTAT	VCUTYPE	VDEVADD
	VDEVBLCK	VDEVCHAN	VDEVCSW	VDEVCUE	VDEVFLAG	VDEVINTS	VDEVPEND	VDEVPOST	VDEVSTAT	VMAEXP	VMBLOK	VMCF	VMCFREAD
	VMCFRUN	VMCHSTRT	VMCHTBL	VMCOMP	VMCPWAIT	VMCUSTRT	VMDSP	VMDSTAT	VMDVSTRT	VMECEXT	VMESTAT	VMEXTCM	VMEXTPND
	VMEXWAIT	VMFPRS	VMGPRS	VMHIPRI	VMIDLE	VMINQ	VMINVPAG	VMINVSEG	VMIOACTV	VMIOINT	VMIOPND	VMKILL	VMLOGOFF
	VMLOPRI	VMMADDR	VMMCR6	VMMFE	VMMICRO	VMMNOSK	VMMPROB	VMMSHADT	VMNDCNT	VMNEWCR0	VMNORUN	VMOSTAT	VMPAGES
	VMPEND	VMPERCM	VMPERPND	VMPGPND	VMPGPNT	VMPGWAIT	VMPNT	VMPRGIL	VMPRIDSP	VMPSTAT	VMPSW	VMPSWAIT	VMPXINT
	VMQLEVEL	VMQSEND	VMQSTAT	VMRON	VMRSTAT	VMRUN	VMSEG	VMSHADT	VMSHR	VMSYSOP	VMTIDLE	VMTIMER	VMTIONT
	VMTLEVEL	VMTMINQ	VMTMOUTQ	VMTON	VMTPAGE	VMTRBRIN	VMTRCTL	VMTREX	VMTREXT	VMTRIO	VMTRPER	VMTRPRG	VMTRPRV
	VMTRSVC	VMTSEND	VMTTIME	VMVCR0	VMV370R	WAIT	XINTBLOK	XINTCODE	XINTMASK	XINTNEXT	XINTSIZE	XINTSORT	XRIGHT16
	XTNDLOCK	Y0	Y2	Y4	Y6	ZEROES							
DMKEDM	ACORETBL	ARIOCT	ARSPPR	ASYSVM	CLASDASD	CLASGRAF	CLASTERM	CLASURI	CLASURO	CONPNT	CONTASK	CONTSKSZ	CORCP
	CORDISA	CORFLAG	CORFLUSH	CORFPNT	CORFREE	CORSHARE	CORSWPNT	CORTABLE	CPABEND	DATE	DMPCRS	DMPFPRS	DMPGPRS
	DMPINREC	DMPLCORE	DMPPGMAP	DMPSYSRV	DMPTODCK	ECBLOK	EDIT	EXTSIZE	INTKFLIN	IOBFPNT	IOBLOK	IOBSIZE	IOERBLOK
	IOERSIZE	PSA	RCHADD	RCHBLOK	RCHCUTBL	RCHFIOB	RCHSIZE	RCUADD	RCUBLOK	RCUDVTBL	RCUFIOB	RCUSIZE	RDEVADD
	RDEVAICB	RDEVBLOK	RDEVCON	RDEVFIOB	RDEVFLAG	RDEVIOER	RDEVOWN	RDEVPAGE	RDEVRECS	RDEVSIZE	RDEVSPL	RDEVTYPC	RECBLOK
	RECPNT	RECSIZE	RSPLCTL	RSPSFBLK	RSPSIZE	R0	R1	R10	R11	R12	R13	R14	R15
	R2	R3	R4	R5	R6	R7	R8	R9	SFBLOK	SFBPNT	SFBSIZE	SFBUSER	SWPTABLE
	SWPVM	TODATE	TREXSIZE	TREXT	TYP3215	VCHADD	VCHBLOK	VCHCUTBL	VCHSIZE	VCONSIZE	VCUADD	VCUBLOK	VCUDVTBL
	VCUSIZE	VDEVADD	VDEVBLOK	VDEVCON	VDEVIOB	VDEVIOER	VDEVSIZE	VDEVSPL	VDEVTYPC	VDEVTYPE	VMBLOK	VMBSIZE	VMCHSTRT
	VMCHTBL	VMCUSTRT	VMDVSTRT	VMECEXT	VMESTAT	VMEXTCM	VMPNT	VMSEG	VMSIZE	VMTRCTL	VMTREXT	VMUSER	VSPLCTL
	VSPSFBLK	VSPSIZE											

Module	External References (Labels and Modules)												
DMKEIG	CCC	CCHCMDV	CCHDAV	CCHDI	CCHLOG80	CCHRCV	CCHREC	CCHUSV	COMPFES	COMPSEL	COMPSYS	CSW	FFS
	IFCC	IGBLAME	IGTERMSQ	IGVALIDB	INTERCCH	IOELPNTR	IOERBLOK	PSA	RTCODE0	RTCODE1	RTCODE2	RTCODE3	RTCODE4
	RTCODE5	R0	R1	R12	R13	R14	R15	R2	R3	R4	R9	SAVEAREA	SAVEWRK1
	SAVEWRK9	TERMSYS	TIOCCH										
DMKERM	ALARM	BLANKS	BRING	BUFCNT	BUFFER	BUFINLTH	BUFSIZE	DEFER	DFRET	DMKCVTBD	DMKEMA00	DMKEMB00	DMKFREE
	DMKFRET	DMKPTRAN	DMKQCNWT	DMKSYSRM	ERRMSG	F2	F255	NORET	OPERATOR	PSA	R0	R1	R10
	R11	R12	R13	R14	R15	R2	R3	R4	R5	R6	R7	R8	R9
	SAVEAREA	SAVER0	SAVER1	SAVER2	SAVER3	SYSTEM	VMBLOK	XRIGHT16					
DMKFMT	ATTN	BUSY	CAW	CC	CD	CE	CSW	CUE	DE	ICNPSW	IOOPSW	PROPSW	PSA
	R0	R1	R10	R11	R12	R13	R14	R15	R2	R3	R4	R5	R6
	R7	R8	R9	SILI	SKIP	SM	UC	UE					
DMKFRE	ACORETEL	AFREE	ASYSVM	AVMREAL	BALRSAVE	CORPGPNT	CORTABLE	C2	DMKCPE	DMKDSPNP	DMKPTRFR	DMKPTRFT	DMKSYSRM
	FREER0	FREER1	FREER14	FREER15	FREESAVE	F1	F4096	PSA	R0	R1	R10	R11	R12
	R13	R14	R15	R2	R3	R4	R5	R6	R7	R8	R9	SAVESIZE	TEMPSAVE
	TRACCURR	TRACEND	TRACFLG1	TRACSTRT	TRAC67	XPAGNUM	XTNDLOCK						
DMKGIO	BRING	CLASDASD	CLASTAPE	CSW	DEFER	DMKCCWTR	DMKDSPCH	DMKFREE	DMKFRET	DMKIOSQV	DMKPTRAN	DMKSCNVU	DMKUNTFR
	DMKUNTRN	IL	IOBCAW	IOBCC3	IOBCSW	IOBFATAL	IOBFLAG	IOBHVC	IOBIOER	IOBIRA	IOBLINK	IOBLOK	IOBMISC
	IOBMISC2	IOBSIZE	IOBSTAT	IOERBLOK	IOERCSW	IOERDATA	IOEREXT	IOERSIZE	PSA	R0	R1	R10	R11
	R12	R13	R15	R2	R4	R5	R6	R7	R8	R9	SAVEAREA	SAVER2	UE
	VDEVBLOK	VDEVBUSY	VDEVCHAN	VDEVCSW	VDEVDED	VDEVFLAG	VDEVIOB	VDEVIOER	VDEVPEND	VDEVPOST	VDEVSTAT	VDEVTYPC	VDEVUC
	VMACTDEV	VMBLCK	VMCOMP	VMDVSTRT	VMESTAT	VMEXTCM	VMEXWAIT	VMGPRS	VMIOCNT	VMIOWAIT	VMLOPRI	VMPSW	VMQLEVEL
	VMRSTAT												
DMKGRF	ALARM	ARIODV	ASYSVM	ATTN	BLANKS	BRING	BUFCNT	BUFFER	BUFINLTH	BUFSIZE	CC	CCC	CD
	CDC	CE	CHC	CLASGRAF	CONACTV	CONADDR	CONCCW1	CONCCW2	CONCCW4	CONCNT	CONDATA	CONESCP	CONOUTPT
	CONPARM	CONPNT	CONRESP	CONRETN	CONRSV3	CONSTAT	CONSYNC	CONTASK	CONTSIZE	CONTSKSZ	CPEXADD	CPEXBLOK	CPEXR0
	CPEXSIZE	CPID	DE	DEFER	DMKBLDVM	DMKBOXBX	DMKCFMAT	DMKCFMBK	DMKCFMEN	DMKCNSED	DMKCPIEM	DMKCVTBD	DMKCVTDB
	DMKCVTHB	DMKDSPCH	DMKFREE	DMKFRET	DMKIOERR	DMKIOEST	DMKIOSQR	DMKMSWR	DMKPTRAN	DMKQCNCL	DMKQCNET	DMKQCNTO	DMKQCNWT
	DMKRIOCN	DMKSCHRT	DMKSCHST	DMKSCNRD	DMKSCNRU	DMKSTKCP	DMKSYSNM	DMKTBLGR	DMKTBLUP	DMKTBMZI	DMKTBMZO	EDIT	F0
	F1	F255	F256	F3	F4	F5	F8	IFCC	INHIBIT	INTREQ	IOBCAW	IOBCOPY	IOBCSW
	IOBERP	IOBFATAL	IOBFLAG	IOBIOER	IOBIRA	IOBLINK	IOBLOK	IOBMISC	IOBRADD	IOBRCNT	IOBSENS	IOBSIZE	IOBSPEC
	IOBSTAT	IOBUNSL	IOBUSER	IOERBLOK	IOERCSW	IOERDATA	IOEREXT	IOERFLG3	IOERNUM	IOEROVFL	IOERREAD	IOERSIZE	LOGDROP
	LOGHOLD	NORET	NOTIME	PRGC	PRIORITY	PRTC	PSA	RCUBLOK	RCUDVTBL	RDEVACTV	RDEVAIOB	RDEVAIRA	RDEVAPLP
	RDEVBLOK	RDEVCON	RDEVCORD	RDEVCPNA	RDEVCTL	RDEVCUA	RDEVDED	RDEVDISA	RDEVDISB	RDEVENAB	RDEVFLAG	RDEVHIO	RDEVHOLD
	RDEVIOER	RDEVLOG	RDEVMORE	RDEVREAD	RDEVRUN	RDEVSTAT	RDEVTFLG	RDEVTMCD	RDEVTRQ	RDEVTYPC	RDEVTYPE	RDEVUSER	R0
	R1	R10	R11	R12	R13	R14	R15	R2	R3	R4	R5	R6	R7
	R8	R9	SAVEAREA	SAVER0	SAVER2	SILI	SYSTEM	TEMPSAVE	TRQBIRA	TRQBLOK	TRQBSIZE	TRQBUSER	TRQBVAL
	TYP3066	TYP3277	TYP3284	UC	UCASE	UE	VCONCTL	VCONRBSZ	VCONRBUF	VCONRCNT	VDEVBLOK	VDEVCON	VMBLOK
	VMCF	VMCFWAIT	VMDVSTRT	VMGENIO	VMLOGOFF	VMLOGON	VMMCPENV	VMMLEVEL	VMMLINED	VMOSTAT	VMPA2APL	VMPFUNC	VMPXINT
	VMQSTAT	VMRSTAT	VMSYSOP	VMTLEND	VMTTIME	VMVTERM	XINTBLOK	XINTCODE	XINTNEXT	XINTSIZE	XINTSORT	XTNDLOCK	ZEROES

Module	External References (Labels and Modules)											
DMKHVC	BLANKS	BRING	CCC	CDC	CE	CHC	CLASGRAF	CLASTERM	CPUID	DE	DEFER	DMKCCWTC	DMKCCWTR
	DMKCFGCL	DMKCFMBK	DMKCFMEN	DMKCVTDT	DMKDGDDK	DMKDSPCH	DMKFREE	DMKFRET	DMKGIOEX	DMKHVDAL	DMKPGSSS	DMKPRGSM	DMKPSASP
	DMKPTRAN	DMKSCNVU	DMKTMRPT	DMKUNTFR	DMKVIOEX	F1	F16	F256	F4	F4095	F60	IFCC	IL
	IOBCAW	IOBCSW	IOBLOK	IOBRADD	IOBSIZE	NICBLOK	NICGRAF	NICSIZE	NICTYPE	PCI	PRGC	PRTC	PSA
	RCWADDR	RCWCCW	RCWCTL	RCWPNT	RCWTASK	RDEVBLOK	RDEVNICL	RDEVTYPC	RDEVTYPE	R0	R1	R10	R11
	R12	R13	R14	R15	R2	R3	R4	R5	R6	R7	R8	R9	TEMPR6
	TEMPR8	TYPBSC	TYP3277	UC	UE	VDEVBLOK	VDEVIOB	VMBLOK	VMCF	VMCFWAIT	VMCOMND	VMDVSTRT	VMESTAT
	VMEXTCM	VMEXWAIT	VMGPRS	VMINST	VMICWAIT	VMMCODE	VMMLEVEL	VMMTEXT	VMNOTRAN	VMOSTAT	VMPA2APL	VMPRIDSP	VMPSTAT
	VMPSW	VMQSTAT	VMRSTAT	VMSLEEP	VMSTOR	VMTERM	VMTRMID	VMTTIME	VMVIRCF	VMWSCHG	XPAGNUM		
DMKHVD	ACCTACNO	ACCTBLOK	ACCTDIST	ACCTLENG	ACCTUSER	ACNTBLOK	ACNTCODE	ACNTDATA	ACNTNUM	ACNTSIZE	ACNTUSER	BRING	CLASGRAF
	CLASSPEC	CLASTERM	CLASURI	CLASURO	CPUMCELL	CPUVERSN	DEFER	DMKACOQU	DMKCPEID	DMKCPVAA	DMKCVTDB	DMKDRDER	DMKDRDMP
	DMKDRDSY	DMKFREE	DMKFRET	DMKIOEFM	DMKPSASP	DMKPTRAN	DMKRPAGT	DMKSCNRU	DMKSCNVD	DMKSCNVU	DMKSNCP	DMKSYSER	DMKSYSRM
	DMKUDRDS	DMKUDRFU	DMKUDRRD	DMKUDRRV	FFS	FTR35MB	F256	F3	F4	F4095	F4096	F60	F8
	IPUADDR	NICBLOK	NICGRAF	NICLLEN	NICSIZE	NICTYPE	PSA	RDEVBLOK	RDEVCODE	RDEVFTR	RDEVLLEN	RDEVMDL	RDEVNICL
	RDEVTYPC	RDEVTYPE	R0	R1	R10	R11	R13	R14	R15	R2	R3	R4	R5
	R6	R7	R8	R9	SAVEAREA	SAVER0	SYSIPLDV	TYPBSC	TYP2305	TYP2319	TYP3210	TYP3277	TYP3330
	TYP3340	TYP3350	UDBFBLOK	UDBFSIZE	UDBFVADD	UDIRBLOK	UDIRDISP	UDIRUSER	UMACACCT	UMACBLOK	VDEVBLOK	VDEVDED	VDEVREAL
	VDEVSTAT	VDEVTYPC	VDEVTYPE	VMACCOUN	VMACCUNT	VMBLOK	VMCLASSA	VMCLASSB	VMCLASSC	VMCLASSE	VMCLASSF	VMCLEVEL	VMDVSTRT
	VMESTAT	VMEXTCM	VMGPRS	VMINST	VMPA2APL	VMPSTAT	VMPSW	VMQSTAT	VMTERM	VMTRMID	VMUSER	VMVTERM	XPAGNUM
	XRIGHT24	X2048BND											
DMKIOC	CLASDASD	CLASTERM	OBRDEVSH	OBRDEVTN	OBRRECN	OBRSHOBR	OBRSWSN	PSA	RCUBLOK	RCUTYPE	RCU2701	RCU2702	RDEVBLOK
	RDEVCUA	RDEVMDL	RDEVSADN	RDEVTMCD	RDEVTYPC	RDEVTYPE	R0	R12	R13	R4	R5	R6	R8
	R9	SAVEAREA	TYP2305	TYP2311	TYP3330	ZEROES							
DMKIOE	CCC	CDC	CLASDASD	CLASGRAF	CLASSPEC	CLASTAPE	CLASTERM	CONCCW3	CCNDATA	CONDCNT	CPEXADD	CPEXBLOK	CPEXFPNT
	CPEXREGS	CPEXSIZE	CPUID	DMKCCHRT	DMKCFMBK	DMKCFPRR	DMKDSPCH	DMKFREE	DMKFRET	DMKIOFC1	DMKIOFIN	DMKIOFM1	DMKIOFOB
	DMKICFST	DMKIOFVR	DMKIOGF1	DMKIOGF2	DMKQCNWT	DMKSTKCP	DMKSYSTZ	ERRBLOK	ERRCCNT	ERRCCW	ERRCORR	ERRHEADR	ERRIOB
	ERRIOER	ERRKEY	ERRMIOB	ERRMIOER	ERRPARM	ERRSDR	ERRSIZE	ERRVOLID	FTREXTSN	F1	F10	F255	F4
	F7	F8	IFCC	IOBCP	IOBFATAL	IOBFLAG	IOBHVC	IOBIOER	IOBLOK	IOBRADD	IOBSTAT	IOBUSER	IOERADR
	IOERBLOK	IOERCEMD	IOERCSW	IOERDATA	IOEREXT	IOERFLG2	IOERLEN	IOERPNT	IOERSIZE	IOERVSER	IRMAND	IRMBIT1	IRMBIT2
	IRMBLOK	IRMBYT1	IRMBYT2	IRMFLG	IRMLMT	IRMLMTCT	IRMMAXCT	IRMOR	IRMRLADD	IRMSIZE	NORET	OBRCORL	OBRCPIDN
	OBRCSWN	OBRCUAIN	OBRCUAPR	OBRDDCNT	OBRFCCWN	OBRHAN	OBRKEYN	OBRLSKN	OBRPGMN	OBRRECN	OBRSDRCT	OBRSENSN	OBRSNSCT
	OBRSWSN	OBRTAPSN	OBRURSNS	OBRVOLN	OBR3211S	OBR33SNS	OBR3420S	PSA	RDEVBLOK	RDEVCTRS	RDEVFTR	RDEVIOER	RDEVIRM
	RDEVMDI	RDEVSER	RDEVSTAT	RDEVTYPC	RDEVTYPE	R0	R1	R10	R11	R12	R13	R14	R15
	R2	R3	R4	R5	R6	R7	R8	R9	SAVEAREA	SAVEREGS	SAVER1	SDRBLOK	SDRBSIZE
	SDRCTRS	SDRFLAGS	SDRLNGTH	SDRSHRT	TNSCPIDN	TNSDEVAD	TNSKEYN	TNSREC	TNSSNS1	TNSSWS3	TNSVOLID	TYP2305	TYP3211
	TYP3330	TYP3340	TYP3350	TYP3410	TYP3420	TYP3505	UC	VMBLOK	VMCLASSF	VMCLEVEL	VMUSER	XOBRFLAG	XOBRT1
	XOBRT3	XOBR010	XOBR150	XOBR180	XOBR512	ZEROES							

Module	External References (Labels and Modules)											
DMKIOF	BRING	CDC	CLASDASD	CLASGRAF	CLASSPEC	CLASTAPE	CLASTERM	CLASURI	CLASURO	CPEXBLOK	CPEXFPNT	CPEXREGS	CPEXR6
	CPEXSIZE	CPUID	CPUVERSN	DEFER	DMKERMSG	DMKFREE	DMKFRET	DMKIOCVT	DMKIOECQ	DMKIOEES	DMKIOEIQ	DMKIOEMP	DMKIOEMQ
	DMKIOEMS	DMKIOEMX	DMKIOENI	DMKIOENQ	DMKIOEOP	DMKIOERP	DMKIOERQ	DMKIOESQ	DMKIOEVQ	DMKPGTVG	DMKPGTVR	DMKPTRAN	DMKPTRUL
	DMKRPAGT	DMKRPAPT	DMKSTKCP	ERRBLOK	ERRCCNT	ERRCCW	ERRCONT	ERRCORR	ERRIOB	ERRIOER	ERRKEY	ERRMIOB	ERRMIOER
	ERRPARM	ERRSDR	ERRVOLID	FFS	FTREXTSN	F15	F255	F4	F7	IOBFATAL	IOERADR	IOERBLOK	IOERCSW
	IOERDATA	IOEREXT	IOERFLG3	IOERLEN	IOEROVFL	IOERPNT	IOERREAD	IOERVSER	LOCK	OBRCORL	OBRCPIDN	OBRCSWN	OBRCUAIN
	OBRCUAPR	OBRDDCNT	OBRDEVTN	OBRFCCWN	OBRHAN	OBRIORTY	OBRKEYN	OBRLSKN	OBRPGMN	OBRRECN	OBRSDRCT	OBRSDRSH	OBRSHOBR
	OBRSNSCT	OBRSSDR1	OBRSWSN	OBRTEMP	OBRVOLN	OBR33SNS	PSA	RCUBLOK	RCUTYPE	RCU2701	RCU2702	RDEVBLOK	RDEVCTRS
	RDEVCUA	RDEVFTR	RDEVMDL	RDEVSADN	RDEVTMCD	RDEVTYPC	RDEVTYPE	RECCCPD	RECFLAG1	RECNXT	RECPAG	RECPAGFL	RECPAGIU
	R0	R1	R10	R11	R12	R13	R14	R15	R2	R3	R4	R5	R6
	R7	R8	R9	SAVEAREA	SDRBLOK	SDRCTRS	SDRCUA	SDRFLAGS	SDRFLCT	SDRLNGTH	SDRMAX	SDROVFWK	SDRPRMCT
	SDRRDEV	SDRSHRT	SDRSIZE	SDRSIZE1	SYSTEM	TNSCPIDN	TNSDEVAD	TNSKEYN	TNSREC	TNSSNS1	TNSSWS3	TNSVOLID	TYPTTY
	TYP1050	TYP1403	TYP1443	TYP2305	TYP2311	TYP2501	TYP2520R	TYP2540R	TYP2700	TYP2741	TYP3066	TYP3210	TYP3211
	TYP3330	TYP3340	TYP3350	TYP3410	TYP3420	TYP3505	VMBLOK	VMUSER	XOBRFLAG	XOBRT1	XOBRT3	XOBR010	XOBR150
	XOBR180	XOBR512	ZEROES										
DMKIOG	ARIOCH	ARIOCT	BRING	CHANID	CPEXSIZE	CPUID	CPUMCELL	CPUMODEL	CPUVERSN	DEFER	DMKCCHCF	DMKCCHMX	DMKCCHSZ
	DMKCCH60	DMKEIG80	DMKERMSG	DMKFREE	DMKIOEES	DMKIOEMP	DMKIOEMS	DMKIOEMX	DMKIOENI	DMKIOEOP	DMKMCHAR	DMKMCHBL	DMKMCHRD
	DMKPGTVG	DMKPGTVR	DMKPTRAN	DMKRPAGT	DMKRPAPT	DMKSCNRU	DMKSEV70	DMKSIX60	DMKSYSER	ECSWLOG	F7	IOELPNTR	LOCK
	MCDAMLEN	MCHAREA	MCHFIX	MCHMODEL	MCNPSW	MODEFLAG	MODEL135	MODEL145	MODEL155	MODEL158	MODEL165	MODEL168	MODEQUIT
	NOMODEL	PSA	RCHBLOK	RCHTYPE	RCH370	RDEVBLOK	RDEVCODE	RDEVTYPE	RECCCPD	RECFLAG1	RECFLAG2	RECNXT	RECPAG
	RECPAGFL	RECPAGFM	RECPAGFR	RECPAGIU	R0	R1	R10	R11	R12	R13	R14	R15	R2
	R3	R4	R5	R6	R8	R9	SAVEAREA	SAVEREGS	SAVEWRK2	SAVEWRK3	SAVEWRK7	SYSIPLDV	SYSTEM
	TYP2305	TYP3330	TYP3340	TYP3350	VMBLOK	WAIT							
DMKIOS	ADSPCH	ASYSVM	ATTN	BUSY	CAW	CC	CCC	CDC	CE	CHC	CLASDASD	CLASGRAF	CLASSPEC
	CLASTAPE	CLASTERM	CLASURI	CLASURO	CPCREG0	CPCREG8	CPEXADD	CPEXBLOK	CPEXR13	CPEXSIZE	CPSTATUS	CPWAIT	CSW
	CUE	C0	C8	DE	DMKBSCER	DMKCCHIS	DMKCCHNT	DMKCNSIN	DMKDASER	DMKDASRD	DMKDSPCH	DMKFREE	DMKFRET
	DMKGRFIN	DMKIOERR	DMKRGAIN	DMKRNHIN	DMKRSPER	DMKRSPEX	DMKSCHDL	DMKSCNRU	DMKSTKCP	DMKSTKIO	DMKTAPER	DMKTRCSI	DMKVIOIN
	FTRRPS	F0	F1	IFCC	IL	INTREQ	INTTIO	IOBBPNT	IOBCAW	IOBCC1	IOBCC2	IOBCC3	IOBCP
	IOBCSW	IOBCYL	IOBERP	IOBFATAL	IOBFLAG	IOBFPNT	IOBHIO	IOBHVC	IOBIOER	IOBIRA	IOBLINK	IOBLOK	IOBPAG
	IOBRADD	IOBRCAW	IOBRELCU	IOBRES	IOBRSTRT	IOBSIOF	IOBSIZE	IOBSNSIO	IOBSPEC	IOBSPLT	IOBSTAT	IOBTIO	IOBUC
	IOBUNSI	IOBUSER	IOBVADD	IOERBLOK	IOERCCW	IOERCSW	IOERDATA	IOEREXT	IOERLEN	IOERSIZE	IOOPSW	MNCLSEEK	MNCOCYL
	PCI	PRGC	PRTC	PSA	QUANTUMR	RCHADD	RCHBLOK	RCHEMX	RCHBUSY	RCHDISA	RCHFIOB	RCHMPX	RCHQCNT
	RCHSEL	RCHSTAT	RCHTYPE	RCH370	RCUADD	RCUBLOK	RCUBUSY	RCUCHA	RCUDISA	RCUFIOB	RCUPRIME	RCUQCNT	RCUSCED
	RCUSHRD	RCUSTAT	RCUSUB	RCUTYPE	RDEVADD	RDEVAIOE	RDEVATT	RDEVBLOK	RDEVBUCH	RDEVBUSY	RDEVCONC	RDEVCUA	RDEVCYL
	RDEVDED	RDEVDISA	RDEVFIOB	RDEVFLAG	RDEVFTR	RDEVIOCT	RDEVIOER	RDEVLIOB	RDEVQCNT	RDEVRACT	RDEVSCED	RDEVSKUP	RDEVSTAT
	RDEVSTA2	RDEVTYPC	RDEVTYPE	RDEVUSER	RUNUSER	R0	R1	R10	R11	R12	R13	R14	R15
	R2	R3	R4	R5	R6	R7	R8	R9	SAVEAREA	SAVER11	SILI	SKIP	SM
	TEMPR14	TIMER	TRACBEF	TRACCURR	TRACEND	TRACFLG1	TRACFLG2	TRACSTRT	TRAC05	TYPBSC	TYPCTCA	UC	VDEVBLOK
	VDEVIOCT	VDEVREAL	VMBLOK	VMESTAT	VMEXTCM	VMFPRS	VMGPRS	VMIDLE	VMIOWAIT	VMPSW	VMRSTAT	VMTMOUTQ	VMTRCTL
	VMTRSIO	VMTTIME	Y0	Y2	Y4	Y6							

Module	External References (Labels and Modules)												
DMKISM	CD	DMKFREE	DMKPTRAN	DMKPTRUL	DMKUNTIS	F16	F2	F4	F8	IDA	IOBCAW	IOBIRA	IOBLOK
	IOBMISC	PSA	RCWCCNT	RCWCCW	RCWIC	RCWPNT	RCWRCNT	RCWTASK	RCWVCAW	R0	R1	R10	R11
	R12	R13	R14	R15	R2	R3	R4	R5	R6	R7	R8	R9	SAVEAREA
	VMBLOK												
DMKLD00	DMKCPE	DMKPSA	DMKWRM	R0	R1	R10	R12	R13	R14	R15	R2	R3	R4
	R5	R6	R7	R8	R9								
DMKLNK	BLANKS	BUFFER	BUFINLTH	BUFNXT	BUFSIZE	CLASDASD	DMKCVTBD	DMKCVTBH	DMKCVTHB	DMKEPSWD	DMKERMSG	DMKFREE	DMKFRET
	DMKLOCK	DMKLCCKD	DMKQCNRD	DMKQCNWT	DMKSCNAU	DMKSCNFD	DMKSCNLI	DMKSCNVN	DMKSCNVS	DMKSCNVU	DMKUDRFD	DMKUDRFU	DMKUDRRV
	DMKVDREL	DMKVDSLK	EDIT	ERRMSG	FFS	FTR2311E	FTR2311T	F1	F15	F2	F4095	F7	F8
	INHIBIT	NORET	PSA	RDEVBLOK	RDEVTYPC	RDEVTYPE	R0	R1	R10	R11	R12	R13	R14
	R15	R2	R3	R4	R5	R6	R7	R8	R9	SAVEAREA	SAVERETN	SAVER11	SAVER2
	SAVEWRK1	SAVEWRK2	SAVEWRK4	SAVEWRK5	SAVEWRK6	SAVEWRK7	SAVEWRK8	SAVEWRK9	TYP2311	TYP2314	UCASE	UDBFBLOK	UDBFSIZE
	UDBFVADD	UDEVADD	UDEVBLOK	UDEVDED	UDEVDISP	UDEVFTR	UDEVLINK	UDEVLKDV	UDEVLKID	UDEVLM	UDEVLONG	UDEVLR	UDEVLW
	UDEVMODE	UDEVPASR	UDEVR	UDEVRELN	UDEVSTAT	UDEVTDSK	UDEVTYPC	UDEVTYPE	UDEVVSER	UDEVW	UDIRBLOK	UDIRDISP	VCHBLOK
	VCHDED	VCHSTAT	VDEVBLOK	VDEVFLAG	VDEVRDO	VDEVREAL	VDEVRELN	VDEVTYPC	VDEVTYPE	VDEVUSER	VMBLOK	VMCOMND	VMKILL
	VMLOGON	VMOSTAT	VMPSWDCT	VMRSTAT	VMUSER	VMVIRCF	ZEROES						
DMKLOC	ASYSLC	BALRSAVE	BALR14	CPEXADD	CPEXBLOK	CPEXFPNT	CPEXREGS	CPEXSIZE	DMKDSPCH	DMKFREE	DMKFRET	DMKSTKCP	DMKSYSLB
	LOCKBLOK	LOCKNAME	LOCKNEXT	LOCKQUE	LOCKSIZE	PSA	R0	R1	R10	R12	R14	R15	R2
	R3	R4	R5	R6	R7	R8	R9	SYSLOCS					
DMKLOG	ARICDC	ARIODV	ASYSLC	ASYSOP	ASYSVM	BLANKS	BUFCNT	BUFFER	BUFNXT	BUFSIZE	CLASDASD	CLASSPEC	CLASTERM
	CPMICAVL	CPSTAT2	DMKACON	DMKBLDEC	DMKBLDRT	DMKBLDVM	DMKCFGII	DMKCQRFI	DMKCVTBD	DMKCVTBH	DMKCVTDT	DMKEPSWD	DMKERMSG
	DMKFREE	DMKFRET	DMKLNKSB	DMKQCNSY	DMKQCNWT	DMKSCHDL	DMKSCHRT	DMKSCH80	DMKSCNAU	DMKSCNFD	DMKSCNRD	DMKSCNRU	DMKSCNVD
	DMKSCNVN	DMKSCNVU	DMKSYSCK	DMKSYSDT	DMKSYSDW	DMKSYSLG	DMKSYSLW	DMKSYSMA	DMKSYSMU	DMKSYSNM	DMKSYSTI	DMKSYSTM	DMKUDRFU
	DMKUDRRD	DMKUDRRV	DMKUSOFF	DMKVDSAT	DMKVDSDF	FFS	F1	F240	F4095	F7	F8	INHIBIT	IOBLOK
	IOBUSER	MICBLOK	MICCREG	MICRSEG	MICSIZE	MICVPSW	MICWORK	NEWPAGES	NEWSEGS	NICBLOK	NICFLAG	NICPSUP	NICSIZE
	NICUSER	NORET	OPERATOR	PSA	RDEVADD	RDEVAIOB	RDEVBLOK	RDEVDED	RDEVDISA	RDEVFLAG	RDEVNICL	RDEVOWN	RDEVPSUP
	RDEVSER	RDEVSIZE	RDEVSTAT	RDEVSYS	RDEVTYPC	RDEVTYPE	RDEVUSER	RUNUSER	R0	R1	R10	R11	R12
	R13	R14	R15	R2	R3	R4	R5	R6	R7	R8	R9	SAVEAREA	SAVERETN
	SAVER11	SAVER2	SAVER9	SAVEWRK1	SAVEWRK2	SAVEWRK8	SYSLOCS	TEMPSAVE	TRQBIRA	TRQBLOK	TRQBSIZE	TRQBUSER	TYPBSC
	TYP1052	TYP2305	UDBFBLOK	UDBFSIZE	UDBFVADD	UDEVADD	UDEVBLOK	UDEVDED	UDEVDISP	UDEVLINK	UDEVLKDV	UDEVLKID	UDEVLONG
	UDEVMODE	UDEVSIZE	UDEVSTAT	UDEVTDSK	UDEVTYPC	UDEVTYPE	UDEVVSER	UDIRBLOK	UDIRDISP	UDIRPASS	UDIRUSER	UMACACC	UMACACCT
	UMACBLOK	UMACBMX	UMACCDEL	UMACCLA	UMACCLEV	UMACCORE	UMACDIST	UMACDVCT	UMACECOP	UMACES	UMACIPL	UMACISAM	UMACLDEL
	UMACLEND	UMACNSVC	UMACOPT	UMACPRIR	UMACRT	UMACVROP	VCHADD	VCHBLOK	VCHSIZE	VCONCTL	VCONRBSZ	VCONRBUF	VCONRCNT
	VCUADD	VCUBLOK	VCUSIZE	VDEVADD	VDEVAUCR	VDEVBLOK	VDEVCFLG	VDEVCON	VDEVSIZE	VMACCOUN	VMACNT	VMACOUNT	VMBLOK
	VMBSIZE	VMCF	VMCFREAD	VMCFWAIT	VMCHCNT	VMCHSTRT	VMCLEVEL	VMCOMND	VMCUCNT	VMCUSTRT	VMDELAY	VMDISC	VMDIST
	VMDVCNT	VMDVSTRT	VMECEXT	VMESTAT	VMFBMX	VMFSTAT	VMISAM	VMKILL	VMLOGON	VMMACCCN	VMMCODE	VMMCPENV	VMMCR6
	VMMFE	VMMICRO	VMMICSVC	VMMIMSG	VMMLEVEL	VMMLINED	VMMLVL2	VMMSGON	VMMSVC	VMMTEXT	VMM360	VMOSTAT	VMPNT
	VMPSTAT	VMPSW	VMPSWDCT	VMQSTAT	VMREAL	VMRON	VMRSTAT	VMSEG	VMSIZE	VMSLEEP	VMSTOR	VMSYSOP	VMTCDEL
	VMTERM	VMTESCP	VMTIMEON	VMTIMER	VMTLDEL	VMTLEND	VMTLEVEL	VMTMOUTQ	VMTON	VMTRMID	VMTRQBLK	VMTTIME	VMUPRIOR
	VMUSER	VMVCR0	VMVIRCF	VMVTERM	VMVTIME	VMV370R	VMWNGON	VRALOC	WAIT	ZEROES			

Module	External References (Labels and Modules)												
DMKMCC	ACORETBL	ASYSVM	BLANKS	BRING	CC	CPSTOP	CLASTAPE	CORCP	CORFLAG	CORTABLE	CPCREG8	CPEXSIZE	C8
	DASDCL	DEFER	DMKCVTDB	DMKCVTHB	DMKERMSG	DMKFREE	DMKFRET	DMKMONMI	DMKMONSH	DMKMONTH	DMKMONTI	DMKPRGC8	DMKPRGMC
	DMKPRGMI	DMKPRGTI	DMKPTRAN	DMKPTRFR	DMKQCNWT	DMKSCHRT	DMKSCHST	DMKSCNFD	DMKSCNRU	ERROR	FFS	F1	F3
	F4	F4095	F60	F8	IOBCAW	IOBLOK	IOBMISC	IOBSIZE	IOBUSER	LOCK	MNBHDLEN	MONAIOB	MONARDB
	MONATRB	MONCCM	MONCTEE1	MONDVLST	MONDVNUM	MONFLAG1	MONNEXT	MONSIZE	MONUSER	NORET	PAGECUR	PAGEND	PAGENXT
	PERFCL	PSA	RDEVBLOK	RDEVDED	RDEVDISA	RDEVFLAG	RDEVSTAT	RDEVSYS	RDEVTYPC	RDEVUSER	R0	R1	R10
	R11	R13	R2	R3	R4	R5	R6	R7	R8	R9	SAVEAREA	SAVEWRK1	SAVEWRK3
	SCHEDCL	SILI	SPROFCL	SYSTEM	TBUSY	TRACCURR	TRACEFLG	TRACSTRT	TRQBIRA	TRQBLOK	TRQBSIZE	TRQBTOD	TRQBUSER
	TRQBVAL	USERCL	VMBLOK	VMUSER	ZEROES								
DMKMCH	ACORETBL	ALARM	ASYSVM	AVMREAL	CORBPNT	CORDISA	CORFLAG	CORFPNT	CORIOLCK	CORPGPNT	CORSWPNT	CORTABLE	CPCREG0
	CPEXADD	CPEXBLOK	CPEXSIZE	CPID	CPUID	CPUVERSN	C0	C13	C3	C7	DMKCFMBK	DMKCFPRR	DMKDMPRS
	DMKDSPCH	DMKERMSG	DMKFREE	DMKIOEMC	DMKOPRWT	DMKPGSPO	DMKPTRFT	DMKQCNWT	DMKSCNFD	DMKSTKCP	DMKSYSCK	FFS	F255
	F6	F8	INTMC	MCCPUID	MCFXDLOG	MCHEK	MCNPSW	MCOLDPW	MCOPSW	MCPROGID	MCREC	MCRECORD	MCRECTYP
	NORET	OPERATOR	PAGCORE	PAGINVAL	PROBMODE	PSA	QUANTUMR	RECOVRPT	RUNUSER	R0	R1	R10	R11
	R12	R13	R14	R15	R2	R3	R4	R5	R6	R7	R8	R9	SAVEAREA
	SWPCHG1	SWPCHG2	SWPFLAG	SWPKEY1	SWPKEY2	TIMER	TRACCURR	TRACEND	TRACFLG1	TRACSTRT	TRAC04	TRANMODE	VMBLOK
	VMESTAT	VMEXTCM	VMEXWAIT	VMFPRS	VMGPRS	VMINVPAG	VMKILL	VMOSTAT	VMPSW	VMRSTAT	VMTMOUTQ	VMTTIME	VMUSER
	Y0	Y2	Y4	Y6	ZEROES								
DMKMID	ALARM	ASYSVM	DATE	DMKCVTDT	DMKERMSG	DMKQCNWT	DMKSCHST	DMKSYSDW	DMKSYSTI	NORET	PSA	R0	R1
	R10	R11	R12	R13	R2	R3	R4	R5	R6	R7	R8	R9	SAVEAREA
	SAVER11	TEMPSAVE	TODATE	TRQBLOK	TRQBVAL	VMBLOK	VMMLEVEL	VMMSGON	VMPNT	VMTTIME			

Module	External References (Labels and Modules)												
DMKMON	ALOCBLOK	ALOCMAX	ALOCUSED	ARIOCH	ARIOCT	ARIOCU	ARIODV	ASYSVM	CC	CPSTOP	CLASDASD	CLASTAPE	CONADDR
	CONCNT	CONTASK	CORCP	CORFLAG	CORFPNT	CORTABLE	CPCREG8	CPEXADD	CPEXBLOK	CPEXR0	CPEXSIZE	CPUID	CUE
	C8	DASDCL	DE	DMKCPEID	DMKCVTDT	DMKDSPAC	DMKDSPBC	DMKDSPCC	DMKDSPCH	DMKDSPCK	DMKDSPIT	DMKDSPNP	DMKDSPPT
	DMKERMSG	DMKFREE	DMKFRET	DMKHVCDI	DMKIOSCT	DMKIOSQR	DMKPAGCC	DMKPAGPS	DMKPRGCT	DMKPRGC8	DMKPRGGR	DMKPRGMC	DMKPRGMI
	DMKPRGTI	DMKPRVCD	DMKPRVCE	DMKPRVCH	DMKPRVCP	DMKPRVCS	DMKPRVCT	DMKPRVDI	DMKPRVEK	DMKPRVEP	DMKPRVIK	DMKPRVIP	DMKPRVLC
	DMKPRVLP	DMKPRVLR	DMKPRVMN	DMKPRVMO	DMKPRVMS	DMKPRVNC	DMKPRVPB	DMKPRVPE	DMKPRVPT	DMKPRVRR	DMKPRVTC	DMKPRVTE	DMKPSANX
	DMKPTRCS	DMKPTRFC	DMKPTRFF	DMKPTRFN	DMKPTRFT	DMKPTRF0	DMKPTRPR	DMKPTRRC	DMKPTRRF	DMKPTRSC	DMKPTRSS	DMKPTRSW	DMKPTRUL
	DMKSCHAL	DMKSCHCT	DMKSCHN1	DMKSCHN2	DMKSCHPU	DMKSCHQ1	DMKSCHRT	DMKSCHST	DMKSCHW1	DMKSCHW2	DMKSTKCP	DMKSYSND	DMKSYSNM
	DMKSYSOC	DMKSYSOW	DMKVIOCI	DMKVIOCT	DMKVIOCW	DMKVIOHD	DMKVIOHI	DMKVIOSF	DMKVIOSI	DMKVIOTC	DMKVIOTI	ERROR	F0
	F1	F3	F4	F4095	IDLEWAIT	IOBCAW	IOBCSW	IOBCYL	IOBFATAL	IOBFLAG	IOBIOER	IOBIRA	IOBLOK
	IOBMISC	IOBMISC2	IOBSIZE	IOBSTAT	IOERSIZE	IONTWAIT	IPLPSW	MNBHDLEN	MNCLDAST	MNCLPERF	MNCLSYS	MNCLUSER	MNCODA
	MNCODAS	MNCODASH	MNCOSUS	MNCOSYS	MNCOTH	MNCOTT	MNCOUSER	MNHCLASS	MNHCODE	MNHDR	MNHDRLEN	MNHRECSZ	MNHTOD
	MN000	MN000INT	MN000LEN	MN000PPA	MN000PPC	MN000PRB	MN000PSI	MN000Q1E	MN000Q2E	MN000WID	MN000WIO	MN000WPG	MN097
	MN097CPU	MN097CR8	MN097DAT	MN097LEN	MN097LEV	MN097TIM	MN097UID	MN098	MN098LEN	MN098UID	MN099	MN099CNT	MN099LEN
	MN099TOD	MN10X	MN10XADD	MN10XLEN	MN10XUID	MN10YCNT	MN10YIO	MN10YLEN	MN2RSV1	MN20X	MN20XNPP	MN20XQNM	MN20XQ1E
	MN20XQ1N	MN20XQ2E	MN20XQ2N	MN20XSWS	MN20XUID	MN20XWSS	MN20YTTI	MN20YVTI	MN202APR	MN202CRD	MN202IOC	MN202LEN	MN202LIN
	MN202PGR	MN202PNC	MN202PRI	MN202PST	MN202REF	MN202RES	MN203LEN	MN204LEN	MN204PRI	MN4RSV1	MN400	MN400CRD	MN400INT
	MN400IOC	MN400LEN	MN400LIN	MN400PDK	MN400PDR	MN400PGR	MN400PGW	MN400PNC	MN400PST	MN400QLV	MN400RES	MN400RST	MN400TTI
	MN400UID	MN400UPR	MN400VTI	MN400WSS	MN500	MN500INS	MN500LEN	MN500OVH	MN500UID	MN500VAD	MN600ADD	MN600CNT	MN600DEV
	MN600DLN	MN600HDR	MN600HLN	MN600MAX	MN600NUM	MN600SER	MN600TY	MN700	MN700ADD	MN700CCY	MN700CYL	MN700DIR	MN700LEN
	MN700QCH	MN700QCU	MN700QDV	MN700UID	MN802CLN	MN802CNT	MN802CTR	MN802DEV	MN802DLN	MN802NAU	MN802NPP	MN802NUM	MN802PGR
	MN802PGW	MN802PRB	MN802WID	MN802WIO	MN802WPG	MONAIOB	MONARDB	MONATRB	MONCLASS	MONCLOCK	MONCODE	MONCOM	MONCTEB1
	MONDVLST	MONDVNUM	MONFLAG1	MONFLAG2	MONNEXT	MONSAVE	MONSIZE	MONSUSCK	MONSUSCT	MONTIINT	MONUSER	PAGECUR	PAGEND
	PAGENXT	PAGEWAIT	PERFCL	PGREAD	PGWRITE	PROBTIME	PROPSW	PSA	PSASVCCT	RCHADD	RCHBLOK	RCHCUTBL	RCHQCNT
	RCUADD	RCUBLOK	RCUCHA	RCUDVTBL	RCUPRIME	RCUQCNT	RCUSUB	RCUTYPE	RDEVADD	RDEVALLN	RDEVBLOK	RDEVCUA	RDEVCYL
	RDEVDISA	RDEVFLAG	RDEVIOCT	RDEVPREF	RDEVQCNT	RDEVSER	RDEVSKUP	RDEVSTAT	RDEVSYS	RDEVTYPC	R0	R1	R10
	R11	R12	R13	R14	R15	R2	R3	R4	R5	R6	R7	R8	R9
	SAVEAREA	SPROFCL	SUSPEND	TBUSY	TRQBLOK	TRQBSIZE	TRQBTOD	TRQBVAL	TRUN	UC	UE	USERCL	VMAEX
	VMBLOK	VMCRDS	VMEPRIOR	VMINST	VMICCNT	VMLINS	VMLOGON	VMPAGES	VMPDISK	VMPDRUM	VMPGREAD	VMPGRINQ	VMPGWRIT
	VMPNCH	VMPNT	VMPSTAT	VMPSW	VMQLEVEL	VMQPRIOR	VMQ1	VMRDINQ	VMRSTAT	VMSTEALS	VMTERM	VMTTIME	VMUPRIOR
	VMUSER	VMVTIME	VMWSPROJ	ZEROES									
DMKMSG	ALARM	ASYSOP	BLANKS	BUFFER	BUFNXT	DMKCVTDB	DMKCVTDT	DMKERMSG	DMKFREE	DMKFRET	DMKQCNRD	DMKQCNWT	DMKSCNAU
	DMKSCNFD	F1	F2	F3	NORET	NOTIME	PRIORITY	PSA	R0	R1	R10	R11	R13
	R15	R2	R3	R4	R5	R7	R8	R9	SAVEAREA	SAVER11	SAVER2	SAVEWRK1	SAVEWRK2
	SAVEWRK4	SAVEWRK6	SAVEWRK8	VMBLOK	VMCLASSA	VMCLASSB	VMCLEVEL	VMDISC	VMKILL	VMLOGOFF	VMMLEVEL	VMMLINED	VMMSGON
	VMOSTAT	VMPNT	VMRSTAT	VMTTIME	VMUSER	VMWNGON	XRIGHT16						
DMKMSW	ALARM	ASYSOP	CCC	CDC	CLASDASD	DMKCVTBH	DMKFREE	DMKFRET	DMKQCNRD	DMKQCNWT	DMKSCNRN	EDIT	F10
	F20	F4	F6	F8	F9	IFCC	INTREQ	IOBLOK	IOBRADD	IOERACT	IOERADR	IOERBLOK	IOERCNCL
	IOERCSW	IOERDASD	IOERDATA	IOERDEC	IOERETRY	IOERFLG1	IOERIGN	IOERIGNR	IOERIND3	IOERIND4	IOERINFO	IOERLEN	IOERNUM
	IOERPEND	IOERSTRT	NORET	NOTIME	OPERATOR	PSA	RDEVBLOK	RDEVCLAS	RDEVDED	RDEVIOER	RDEVSTAT	R0	R1
	R10	R11	R12	R13	R2	R3	R4	R5	R6	R7	R8	R9	SAVEAREA
	SAVER0	SAVER11	TYP3340	UCASE	VMBLOK	VMDISC	VMOSTAT	VMTERM	VMTTIME	VMUSER	ZEROES		

Module	External References (Labels and Modules)												
DMKNEM	R0	R1	R12	R13	R15	R2	R3	R4	R5	SAVEAREA	SAVER0		
DMKNES	ARIOCU	ARIODV	ASYSVM	BLANKS	CACTLTR	CDISPLY	CLASSPEC	CLASTERM	CONCCW3	CONDATA	CONSYSR	CONTASK	CSWLMEP
	CSWLNCP	CTRMLTR	DMKCVTBH	DMKCVTDB	DMKCVTHB	DMKERMSG	DMKFREE	DMKFRET	DMKIOESR	DMKQCNCL	DMKQCNTO	DMKQCNWT	DMKRGBEN
	DMKRIORN	DMKRNHND	DMKRNHTR	DMKSCNFD	DMKSCNRD	DMKSCNRU	FFS	F1	F255	F3	F4	F4095	NICBLOK
	NICCIBM	NICDISA	NICENAB	NICEPAD	NICEPMD	NICFLAG	NICLBSC	NICLINE	NICLTRC	NICPSUP	NICQPNT	NICSESN	NICSIZE
	NICSTAT	NICSWEP	NICTYPE	NICUSER	NORET	PSA	RCHBLOK	RCHCUTBL	RCUBLOK	RCUDISA	RCUDVTBL	RCUSTAT	RDEVADD
	RDEVBASE	RDEVBLOK	RDEVCON	RDEVCTRS	RDEVCUA	RDEVDED	RDEVDISA	RDEVDISB	RDEVENAB	RDEVEPDV	RDEVEPLN	RDEVEPMD	RDEVFLAG
	RDEVIRM	RDEVLNCP	RDEVMAX	RDEVMDL	RDEVNICL	RDEVNRDY	RDEVPDLY	RDEVPTTC	RDEVRCVY	RDEVRSVD	RDEVSADN	RDEVSLOW	RDEVSTAT
	RDEVTBTU	RDEVTCTL	RDEVTMCD	RDEVTYPC	RDEVTYPE	RDEVUSC8	RDEVUSER	RDEVWAIT	R0	R1	R10	R11	R13
	R15	R2	R3	R4	R5	R6	R7	R8	R9	SAVEAREA	SAVER11	SAVER2	SAVER9
	SAVEWRK1	SAVEWRK2	SAVEWRK3	SAVEWRK4	SAVEWRK5	SAVEWRK7	SAVEWRK8	SAVEWRK9	TYPBSC	TYPTTY	TYPUNDEF	TYP2700	TYP3705
	VMBLOK	VMOSTAT	VMTTIME	VMUSER	VMVIRCF								
DMKNET	ARIODV	ASYSVM	BLANKS	CACTLIN	CDCTLIN	CLASSPEC	CLASTERM	CONCCW3	CONSYSR	CONTACT	CRESIMD	DMKCVTBH	DMKCVTHB
	DMKERMSG	DMKFREE	DMKFRET	DMKIOESR	DMKNESDS	DMKNESEP	DMKNESHD	DMKNESPL	DMKNESTR	DMKNESWN	DMKNLDMP	DMKNLDR	DMKQCNWT
	DMKRGBEN	DMKRIORN	DMKRNHND	DMKSCNFD	DMKSCNRD	F255	F3	F4	F4095	F60	F8	NICBLOK	NICCIBM
	NICDISA	NICDISB	NICENAB	NICEPAD	NICEPMD	NICFLAG	NICGRAF	NICLBSC	NICLGRP	NICLINE	NICNAME	NICRSPL	NICSESN
	NICSIZE	NICSTAT	NICTELE	NICTERM	NICTYPE	NICUSER	NORET	PSA	RDEVBLOK	RDEVCTRS	RDEVDED	RDEVDISA	RDEVDISB
	RDEVENAB	RDEVFLAG	RDEVLNCP	RDEVMAX	RDEVNICL	RDEVNRDY	RDEVRSVD	RDEVSTAT	RDEVTYPC	RDEVUSER	R0	R1	R10
	R11	R13	R14	R15	R2	R3	R4	R5	R6	R7	R8	R9	SAVEAREA
	SAVER2	SAVER9	SAVEWRK1	SAVEWRK2	SAVEWRK3	SAVEWRK4	SAVEWRK5	SAVEWRK7	SAVEWRK8	SAVEWRK9	TEMPSAVE	VMBLOK	VMCLASSA
	VMCLASSB	VMCLASSC	VMCLASSD	VMCLASSE	VMCLASSF	VMCLASSG	VMCLEVEL	VMOSTAT	VMSTKO	VMUSER	VMVIRCF	ZEROES	
DMKNLD	ABORT	ADDSFB	ARSPRD	ASYSVM	BLANKS	BRING	CC	CCPARM	CCPENTRY	CCPMAXID	CCPNAME	CCPPSIZE	CCPRESID
	CCPRSTAT	CCPRSTEP	CCPRSTYP	CCPSIZE	CCPTEP	CCPTPEP	CCPTYPE	CDC	CLASSPEC	CUE	DEFER	DMKCFPRI	DMKCKSPL
	DMKCVTBH	DMKCVTDT	DMKCVTHB	DMKDSPCH	DMKERMSG	DMKFREE	DMKFRET	DMKIOSQR	DMKPGTCG	DMKPGTSD	DMKPGTVG	DMKPGTVR	DMKPTRAN
	DMKPTRUL	DMKQCNCL	DMKQCNRD	DMKQCNTO	DMKQCNWT	DMKRNHIN	DMKRNTBL	DMKRPAGT	DMKRPAPT	DMKRSPID	DMKSCNFD	DMKSCNRD	DMKSCNRU
	DMKSCNVS	DMKSCNVU	DMKSTKIO	DMKSYSDU	DMKVDREL	EDIT	ERRMSG	FFS	FTRTYP1	F0	F1	F256	F3
	F4	F4096	F5	F8	IL	INTREQ	IOBBPNT	IOBCAW	IOBCC1	IOBCC3	IOBCP	IOBCSW	IOBFATAL
	IOBFLAG	IOBFPNT	IOBIOER	IOBIRA	IOBLCK	IOBMISC	IOBMISC2	IOBRADD	IOBRCAW	IOBRCNT	IOBRES	IOBRSTRT	IOBSIZE
	IOBSPEC	IOBSTAT	IOBTIO	IOBUSER	IOERBLOK	IOERDATA	IOERETN	IOEREXT	IOERSIZE	IPLREQ	LOCK	NCPNAME	NCPPAGCT
	NCPPNT	NCPSTART	NCPTBL	NCPVOL	NICBLCK	NICCIBM	NICEPAD	NICEPMD	NICFLAG	NICNAME	NICPSUP	NICSIZE	NICSTAT
	NICSWEP	NICTERM	NICTYPE	NICUSER	NOAUTO	NORET	OPERATOR	PSA	RCUBLOK	RCUDISA	RCUDVTBL	RCUSTAT	RDEVADD
	RDEVAIOB	RDEVATT	RDEVAUTO	RDEVBASE	RDEVBLOK	RDEVCODE	RDEVCUA	RDEVDED	RDEVDISA	RDEVENAB	RDEVEPDV	RDEVEPLN	RDEVEPMD
	RDEVFICB	RDEVFLAG	RDEVFTR	RDEVIRM	RDEVLCEP	RDEVLNCP	RDEVMAX	RDEVMDL	RDEVNCP	RDEVNICL	RDEVNRDY	RDEVOWN	RDEVPTTC
	RDEVRCVY	RDEVRSVD	RDEVSTAT	RDEVSTA2	RDEVTFLG	RDEVTMCD	RDEVTYPC	RDEVTYPE	RDEVUSER	RDRCHN	R0	R1	R10
	R11	R12	R13	R14	R15	R2	R3	R4	R5	R6	R7	R8	R9
	SAVEAREA	SAVER11	SAVER2	SAVEWRK1	SAVEWRK2	SAVEWRK3	SAVEWRK4	SAVEWRK5	SAVEWRK6	SAVEWRK7	SAVEWRK8	SAVEWRK9	SFBCLAS
	SFBCOPY	SFBDATE	SFBDIST	SFBDUMP	SFBFILID	SFBFLAG	SFBFNAME	SFBFTYPE	SFBLAST	SFBLOK	SFBORIG	SFBPNT	SFBRECNO
	SFBRECSZ	SFBSIZE	SFBSTART	SFBTIME	SFBTYPE	SFBUSER	SILI	SM	SYSTEM	TEMPSAVE	TYPBSC	TYPIBM1	TYPPRT
	TYPUNDEF	TYP2314	TYP3330	TYP3350	TYP3705	UC	UCASE	VCUBLOK	VCUDVTBL	VDEVADD	VDEVBLOK	VDEVDIAL	VDEVFLAG
	VMBLOK	VMTTIME	VMUSER	X40FFS									

Module	External References (Labels and Modules)												
DMKOPR	ALARM	CAW	CC	CD	CLASGRAF	CPUID	CPUVERSN	CSW	DMKRIOCN	DMKRIODV	FFS	NOAUTO	PSA
	RDEVBLOK	RDEVCORD	RDEVTYPC	RDEVTYPE	R0	R1	R10	R14	R15	R2	R3	R4	R5
	R8	SILI	TYP3066	UC	XRIGHT16								
DMKPAG	ACORETBL	ALARM	ARIODV	CC	CORTABLE	CPEXADD	CPEXBLOK	CPEXBPNT	CPEXFPNT	CPEXMISC	CPEXR0	CPEXR11	CPEXR5
	CPEXR7	DMKCVTBH	DMKDSPCH	DMKFREE	DMKFRET	DMKIOSQR	DMKOPRWT	DMKPTRFF	DMKPTRRQ	DMKPTRSS	DMKPTRWQ	DMKSCNRD	DMKSTKCP
	DMKSYSOW	FTR70MB	F1	F2	F3	F4	F5	F8	IL	IOBBPNT	IOBCAW	IOBCP	IOBCSW
	IOBCYL	IOBFATAL	IOBFLAG	IOBFPNT	IOBIRA	IOBLOK	IOBMISC	IOBPAG	IOBRADD	IOBSIZE	IOBSTAT	IOBUSER	OWNDLIST
	OWNDRDEV	PAGELOAD	PAGERATE	PAGEWAIT	PGSRATIO	PGWAITPG	PSA	RDEVBLOK	RDEVFTR	RDEVMDL	RDEVTYPE	R0	R1
	R10	R11	R12	R13	R14	R15	R2	R3	R4	R5	R6	R7	R8
	R9	SILI	SKIP	SWPCODE	SWPCYL	SWPDPAGE	SWPFLAG	SWPTRANS	TYP2305	TYP2314	TYP3330	TYP3340	TYP3350
	VMBLOK	VMTTIME	XTNDLOCK										
DMKPER	VMBLOK	VMPEND	VMPERPND	VMTRCTL	VMTRPER								
DMKPGS	ACORETBL	ASYSVM	AVMREAL	CORBPNT	CORCFLCK	CORFLAG	CORFPNT	CORIOLCK	CORPGPNT	CORRSV	CORSHARE	CORTABLE	DEFER
	DELPAGES	DMKBLDRL	DMKBLDRT	DMKDSPNP	DMKFRET	DMKPGTPR	DMKPTRAN	DMKPTRFT	DMKPTRPW	DMKPTRRC	DMKPTRSC	FFS	F0
	F15	F4	F4096	F8	KEEPSEGS	NEWPAGES	NEWSEGS	OLDVMSEG	PAGCORE	PAGINVAL	PAGREF	PSA	R0
	R1	R10	R11	R13	R14	R15	R2	R3	R4	R5	R6	R7	R8
	R9	SAVEAREA	SAVER1	SAVER2	SAVEWRK1	SAVEWRK2	SAVEWRK3	SAVEWRK4	SAVEWRK7	SAVEWRK9	SEGPAGE	SEGPLEN	SEGTABLE
	SHRBPNT	SHRFPNT	SHRNAME	SHRSEGCT	SHRSEGNM	SHRTABLE	SHRTSIZE	SHRUSECT	SWPCYL	SWPFLAG	SWPKEY1	SWPRECMP	SWPSHR
	SWPTABLE	SWPVM	SWPVPAGE	TREXANSI	TREXIN1	TREXNSI	TREXT	VMABLOK	VMADSTOP	VMAFPNT	VMANAME	VMASIZE	VMASSIST
	VMBLOK	VMESTAT	VMINVPAG	VMLOGOFF	VMNSHR	VMOSTAT	VMPAGES	VMPGWAIT	VMPSTAT	VMRSTAT	VMSEG	VMSHR	VMSHRSYS
	VMSIZE	VMSTOR	VMTIMER	VMTREXT	XPAGNUM								
DMKPGT	ALARM	ALOCBLOK	ALOCMAP	ALOCMAX	ALOCUSED	ARIODV	ASYSVM	BALRSAVE	BALR0	BALR1	BALR8	CPEXADD	CPEXBLOK
	CPEXSIZE	CPID	DMKCKP	DMKDSPCH	DMKFREE	DMKFRET	DMKQCNWT	DMKSTKCP	DMKSYSOW	FFS	FTR70MB	F1	F3
	F4	IOBCYL	IOBFPNT	IOBLOK	NORET	OPERATOR	OWNDLIST	OWNDRDEV	PSA	RDEVALLN	RDEVBLOK	RDEVCODE	RDEVCYL
	RDEVFIOB	RDEVFLAG	RDEVFTR	RDEVPAGE	RDEVPNT	RDEVPREF	RDEVRECS	RDEVTYPE	RECBLOK	RECCYL	RECMAP	RECMAX	RECPNT
	RECSIZE	RECUSED	R0	R1	R10	R11	R12	R13	R14	R15	R2	R3	R4
	R5	R6	R7	R8	R9	SWPCYL	SWPDPAGE	SWPFLAG	SWPRECMP	TYP2305	TYP2314	TYP3330	TYP3340
	TYP3350	VMBLCK	VMPDISK	VMPDRUM									
DMKPRG	BRING	CPABEND	CPCREG0	CPCREG8	C0	C8	DEFER	DMKCFMBK	DMKDMPDK	DMKDMPGR	DMKDSPB	DMKDSPCH	DMKPERIL
	DMKPRVLG	DMKPTRAN	DMKQCNWT	DMKTRCPG	DMKVATPF	DMKVATPX	DMKVATSX	ECBLOK	EXTPERAD	EXTPERCD	FFS	F1	INTPR
	INTPRL	INTSVCL	MONCLASS	MONCODE	NORET	PERADD	PERCODE	PRNPSW	PROBMODE	PROPSW	PSA	QUANTUMR	RUNUSER
	R0	R1	R10	R11	R12	R13	R14	R15	R2	R3	R4	R5	R6
	R7	R8	R9	SVCNPSW	SVCOPSW	TEMPR14	TEMPR15	TIMER	TRACCURR	TRACEND	TRACFLG1	TRACSTRT	TRAC03
	TRANMODE	TREXADD	TREXINTC	TREXINTL	TREXPERA	TREXPERC	TREXPSW	TREXT	VMBLOK	VMCFRUN	VMCFWAIT	VMECEXT	VMESTAT
	VMEXTCM	VMEXWAIT	VMFPRS	VMGPRS	VMIOPND	VMIOWAIT	VMOSTAT	VMPAGEX	VMPEND	VMPERCM	VMPERPND	VMPRGIL	VMPRGPND
	VMPSTAT	VMPSW	VMRSTAT	VMSHADT	VMSVCPND	VMTMOUTQ	VMTRBRIN	VMTRCTL	VMTREXT	VMTRPER	VMTRPRG	VMTTIME	VMV370R
	Y0	Y2	Y4	Y6									

Module	External References (Labels and Modules)												
DMKPRV	BRING	CHANID	CPCREG0	CPUID	CPUMCELL	CPUVERSN	C0	C1	DEFER	DMKDSPA	DMKDSPB	DMKDSPCH	DMKHVCAL
	DMKPERIL	DMKPRGSM	DMKPSAFP	DMKPSASP	DMKPTRAN	DMKTMRTN	DMKTRCPB	DMKTRCPV	DMKVATAB	DMKVATEX	DMKVATLA	DMKVATRN	DMKVIOEX
	ECBLOK	EXTCR0	EXTCR9	EXTMODE	EXTPERAD	EXTSHCR0	FFS	F15	F16	F240	F4	F5	F6
	F60	F7	INTPR	INTPRL	MNCLINST	MNCOSIM	PERGPRS	PERSALT	PROBMODE	PROPSW	PSA	RUNCR0	R0
	R1	R10	R11	R12	R13	R14	R15	R2	R3	R4	R5	R6	R7
	R8	R9	SWPFLAG	SWPKEY1	SWPSHR	TEMPSAVE	TRANMODE	TREXCR9	TREXINTC	TREXIN1	TREXNSI	TREXPERA	TREXT
	VCHBLOK	VCHBMX	VCHSEL	VCHTYPE	VMBLOK	VMCHSTRT	VMCHTBL	VMDSP	VMDSTAT	VMECEXT	VMESTAT	VMEXTCM	VMEXTPND
	VMEXWAIT	VMGPRS	VMINQ	VMINST	VMINVPAG	VMINVSEG	VMIOINT	VMNEWCR0	VMPEND	VMPERCM	VMPERPND	VMPRGIL	VMPSTAT
	VMPSW	VMPXINT	VMREAL	VMRSTAT	VMRUN	VMSEG	VMTRBRIN	VMTRCTL	VMTREXT	VMTRPER	VMTRPRV	VMVCR0	VMV370R
	WAIT												
DMKPSA	ACORETBL	APAGCP	ASYSOP	ASYSVM	BRING	BUSY	CLASGRAF	CLASTERM	CORFLAG	CORSHARE	CORTABLE	CPABEND	CPCREG0
	CPCREG8	CRESIMD	CSW	CUE	C0	C1	C8	DEFER	DFRET	DMKCFMBK	DMKCVTBH	DMKDMPDK	DMKDMPGR
	DMKDSPB	DMKDSPCH	DMKFREE	DMKFRET	DMKPRGRF	DMKPTRAN	DMKPTRUL	DMKQCNCL	DMKQCNWT	DMKRNHND	DMKSCHTQ	DMKSCNRD	DMKSTKIO
	DMKTMRVT	DMKTRCIT	DMKTRCPB	DMKTRCSV	DMKVERD	DMKVERO	EXOPSW	EXTMODE	FFS	F1	F15	F2	F240
	F4095	F60	F8	INTEX	INTEXF	INTSVC	INTSVCL	LOCK	NICBLOK	NICNAME	NICSIZE	NICUSER	NORET
	PROBMODE	PSASVCCT	QUANTUMR	RDEVBASE	RDEVBLOK	RDEVFLAG	RDEVHIO	RDEVNICL	RDEVTYPC	RDEVUSER	RUNPSW	RUNUSER	R0
	R1	R10	R11	R12	R13	R14	R15	R2	R3	R4	R8	SAVEAREA	SAVENEXT
	SAVERETN	SAVER12	SAVER13	SAVER2	SAVESIZE	SAVEWRK2	SM	SVCNPSW	SVCOPSW	SYSTEM	TIMER	TRACCURR	TRACEND
	TRACFLG1	TRACSTRT	TRAC01	TRAC02	TREXIN1	TREXT	TRQBBPNT	TRQBFPNT	TRQBLOK	TRQBVAL	VMADSTOP	VMBLOK	VMCPUTMR
	VMDISC	VMDSTAT	VMESTAT	VMEXTCM	VMEXWAIT	VMFPRS	VMGPRS	VMINST	VMMCR6	VMMICSVC	VMMSVC	VMOSTAT	VMPEND
	VMPERPND	VMPSW	VMQSEND	VMRSTAT	VMSEG	VMSHR	VMSYSOP	VMTERM	VMTLEVEL	VMTMOUTQ	VMTMRINT	VMTRBRIN	VMTRCTL
	VMTREXT	VMTRMID	VMTRSVC	VMTSEND	VMTTIME	WAIT	XPAGNUM	XRIGHT24	X2048BND	Y0	Y2	Y4	Y6
	ZEROES												
DMKPTR	ACORETBL	ARIODV	ASYSVM	AVMREAL	BALRSAVE	BALR0	BALR2	BRING	CORBPNT	CORCFLCK	CORCP	CORFLAG	CORFPNT
	CORFREE	CORICLCK	CORLCNT	CORPGPNT	CORRSV	CORSHARE	CORSWPNT	CORTABLE	CPEXADD	CPEXBLOK	CPEXFPNT	CPEXMISC	CPEXR0
	CPEXR13	CPEXR2	CPEXR7	CPEXR9	CPEXSIZE	CPSTAT	C1	DEFER	DMKCFMBK	DMKDSPCH	DMKDSPNP	DMKFREE	DMKFRET
	DMKFRETR	DMKPAGIO	DMKPAGQ	DMKPGTPG	DMKPGTPR	DMKQCNWT	DMKSCHDL	DMKSCHN1	DMKSCHN2	DMKSTKCP	DMKSYSOW	DMKSYSRM	DMKVMAPS
	FFS	F0	F1	F4	F4095	F4096	F8	IOERETN	LOCK	NORET	OWNDLIST	OWNDRDEV	PAGCORE
	PAGINVAL	PAGREF	PGREAD	PGWRITE	PSA	RDEVBLOK	RDEVTYPE	R0	R1	R10	R11	R12	R13
	R14	R15	R2	R3	R4	R5	R6	R7	R8	R9	SAVEAREA	SAVEREGS	SAVERETN
	SAVER0	SAVER1	SAVER11	SAVER12	SAVER13	SAVER2	SAVER3	SAVER7	SAVEWRK1	SAVEWRK2	SAVEWRK3	SAVEWRK5	SAVEWRK6
	SAVEWRK9	SWPALLOC	SWPCHG1	SWPCHG2	SWPCODE	SWPCYL	SWPDPAGE	SWPFLAG	SWPKEY1	SWPKEY2	SWPRECMP	SWPREF1	SWPREF2
	SWPSHR	SWPTRANS	SWPVPAGE	SYSTEM	TIMER	TYP2305	VMBLOK	VMESTAT	VMINVPAG	VMNDCNT	VMPAGES	VMPGREAD	VMPGRINQ
	VMPGWAIT	VMPGWRIT	VMPSTAT	VMRPAGE	VMRSTAT	VMSEG	VMSIZE	VMSTEALS	VMTIMER	VMTTIME	VMWCNT	XPAGNUM	XTNDLOCK
	ZEROES												

Module	External References (Labels and Modules)												
DMKQCN	ADSPCH	ALARM	ASYSOP	BALRSAVE	BLANKS	CLASGRAF	CLASSPEC	CLASTERM	CONADDR	CONCNT	CONCNTL	CONDATA	CONOUTPT
	CONPARM	CONPNT	CONRESP	CONRETN	CONRSV3	CONSPLT	CONSTAT	CONSYNC	CONTASK	CONTSIZE	CONTSKSZ	CONUSER	CPEXADD
	CPEXBLOK	CPEXREGS	CPEXR12	CPEXSIZE	DFRET	DMKCNSIC	DMKCVTBD	DMKCVTBH	DMKCVTDT	DMKDSPCH	DMKFREE	DMKFRET	DMKGRFIC
	DMKRGBIC	DMKRNHIC	DMKSCHDL	DMKSCHRT	DMKSCHST	DMKSCNRD	DMKSCNRN	DMKSTKCP	DMKSYSNM	DMKVSPVP	EDIT	F1	F2
	F4095	F8	INHIBIT	MNCLRESP	MNCOBRD	MNCOERD	MNCOWRIT	NICBLOK	NICLLEN	NICSIZE	NOAUTO	NORET	NOTIME
	OPERATOR	PRIORITY	PSA	RDEVAPLP	RDEVBLOK	RDEVCON	RDEVLLEN	RDEVNICL	RDEVTMCD	RDEVTYPC	RDEVTYPE	R0	R1
	R10	R11	R12	R13	R14	R15	R2	R3	R4	R5	R6	R7	R8
	R9	SAVEAREA	SAVER0	SAVER1	SAVER11	SAVER2	SAVER3	SAVEWRK1	SAVEWRK2	SAVEWRK3	SAVEWRK4	TEMPSAVE	TRQBIRA
	TRQBLOK	TRQBSIZE	TRQBUSER	TRQBVAL	TYPBSC	UCASE	VDEVBLOK	VDEVCSPL	VDEVFLAG	VDEVSFLG	VDEVTERM	VMBLOK	VMCF
	VMCFREAD	VMCFRUN	VMCFWAIT	VMDELAY	VMDISC	VMDVSTRT	VMGENIO	VMKILL	VMLOGOFF	VMLOGON	VMMCODE	VMMLEVEL	VMMSTMP
	VMMTEXT	VMOSTAT	VMQSTAT	VMRBSC	VMRSTAT	VMSYSOP	VMTERM	VMTRMID	VMTTIME	VMUSER	VMVIRCF	VMVTERM	
DMKRGA	ASYSVM	BLANKS	BRING	BSCAUSER	BSCBLOK	BSCCNT	BSCCOPY	BSCECCW1	BSCECCW2	BSCENQ	BSCETB	BSCFLAG	BSCFLAG1
	BSCIGN	BSCINDEX	BSCLOG	BSCOPIED	BSCPCCW1	BSCPCCW2	BSCPCCW3	BSCPCCW4	BSCRCVD	BSCREAD	BSCREGEN	BSCRESP	BSCRROBN
	BSCRSTRT	BSCRVI	BSCSCAN	BSCSCCW1	BSCSCCW2	BSCSCCW3	BSCSEL	BSCSEND	BSCSENSE	BSCSIZE	BSCSIZE1	BSCSPTR	BSCTMRQ
	BSCTSTRQ	BSCUCOPY	BSCUECCW	BUFCNT	BUFFER	BUFINLTH	BUFSIZE	CC	CD	CE	CLASTERM	CONACTV	CONADDR
	CONCCW1	CONCCW2	CONCCW3	CONCCW4	CONCNT	CONCNTL	CONDATA	CONDCNT	CONESCP	CONLABEL	CONPARM	CONPNT	CONRESP
	CONRETN	CONSTAT	CONTASK	CONTSIZE	CONTSKSZ	CONUSER	CPEXADD	CPEXBLOK	CPEXR0	CPEXSIZE	DE	DEFER	DMKBLDVM
	DMKCFMAT	DMKCFMBK	DMKCFMEN	DMKCNSED	DMKCVTBD	DMKCVTBH	DMKCVTDB	DMKCVTHB	DMKDSPCH	DMKERMSG	DMKFREE	DMKFRET	DMKIOERN
	DMKIOSQR	DMKPTRAN	DMKQCNCL	DMKQCNET	DMKQCNTO	DMKQCNWT	DMKRGBIC	DMKRGBMT	DMKRGBSN	DMKSCHRT	DMKSCHST	DMKSCNRD	DMKSCNRU
	DMKSTKCP	DMKTBLGR	DMKTBLUP	DMKTBMZI	EDIT	F0	F1	F2	F255	F256	F3	F4	F4095
	F5	F8	INHIBIT	IOBCAW	IOBCC3	IOBCP	IOBCSW	IOBFATAL	IOBFLAG	IOBIOER	IOBIRA	IOBLINK	IOBLOK
	IOBMISC	IOBMISC2	IOBRADD	IOBRCNT	IOBRSTRT	IOBSIZE	IOBSPEC	IOBSTAT	IOBUNSL	IOBUSER	IOERBLOK	IOEREXT	IOERSIZE
	LOGDROP	LOGHOLD	NICALRM	NICAPL	NICATRB	NICBLOK	NICCARD	NICCORD	NICCPNA	NICDIAG	NICDISA	NICDISB	NICENAB
	NICFLAG	NICFMT	NICHOLD	NICMORE	NICNAME	NICNTRL	NICPOLL	NICPROCN	NICQPNT	NICREAD	NICRSPL	NICRUNN	NICSELT
	NICSIO	NICSIZE	NICSTAT	NICTABF	NICTMCD	NICTRQ	NICTYPE	NICUSER	NIC3275	NORET	NOTIME	PSA	RDEVBLOK
	RDEVBSC	RDEVCON	RDEVDISA	RDEVDISB	RDEVENAB	RDEVFLAG	RDEVMAX	RDEVNICL	RDEVNRDY	RDEVPDLY	RDEVRSVD	RDEVSTAT	RDEVTYPC
	RDEVTYPE	RDEVWAII	R0	R1	R10	R11	R12	R13	R14	R15	R2	R3	R4
	R5	R6	R7	R8	R9	SAVEAREA	SAVER0	SAVER2	SILI	SYSTEM	TEMPR2	TEMPR3	TEMPR7
	TEMPSAVE	TRQBIRA	TRQBLOK	TRQBSIZE	TRQBUSER	TRQBVAL	TYPBSC	UCASE	UE	VCONCTL	VCONRBSZ	VCONRBUF	VCONRCNT
	VDEVBLOK	VDEVCON	VMBLOK	VMCF	VMCFWAIT	VMDVSTRT	VMGENIO	VMLOGOFF	VMLOGON	VMMCPENV	VMMLEVEL	VMMLINED	VMOSTAT
	VMPA2APL	VMPFUNC	VMPXINT	VMQSTAT	VMRSTAT	VMTERM	VMTLEND	VMTTIME	VMVTERM	XINTBLOK	XINTCODE	XINTNEXT	XINTSIZE
	XINTSORT	XTNDLOCK											
DMKRGB	ALARM	BRING	BSCAUSER	BSCBLOK	BSCFLAG	BSCLINE	BSCPCCW1	BSCPCCW2	BSCPCCW4	BSCRCVD	BSCREAD	BSCRESP	BSCRROBN
	BSCSCAN	BSCSCCW1	BSCSCCW2	BSCSCCW3	BSCSEL	BSCSIZE	BSCSIZE1	BSCSIZE2	BSCSPTR	BUFINLTH	CC	CD	CONADDR
	CONCCW1	CONCCW2	CONCCW3	CONCCW4	CONCNT	CONCNTL	CONDATA	CONESCP	CONLABEL	CONOUTPT	CONPARM	CONPNT	CONRESP
	CONRETN	CONRSV3	CONSTAT	CONSYNC	CONTASK	CONTSIZE	CONTSKSZ	CONUSER	CPEXADD	CPEXBLOK	CPEXSIZE	DEFER	DMKBOXBX
	DMKDSPCH	DMKFREE	DMKFRET	DMKIOSQR	DMKPTRAN	DMKQCNET	DMKRGAIN	DMKSCHRT	DMKSTKCP	DMKTBLGR	DMKTBMZO	F1	F256
	F4	F4095	INHIBIT	IOBCAW	IOBCP	IOBFLAG	IOBIOER	IOBIRA	IOBLOK	IOBMISC	IOBMISC2	IOBRCNT	IOBRSTRT
	IOBSIZE	IOBSPEC	IOBSTAT	IOBUSER	IOERBLOK	IOEREXT	IOERSIZE	LOGDROP	LOGHOLD	NICALRM	NICAPL	NICATRB	NICBLOK
	NICCORD	NICDIAG	NICDISA	NICDISB	NICFLAG	NICFMT	NICHOLD	NICMORE	NICNTRL	NICPOLL	NICPROCN	NICQPNT	NICREAD
	NICRUNN	NICSELT	NICSIO	NICSIZE	NICSTAT	NICTMCD	NICTRQ	NICUSER	PRIORITY	PSA	RDEVBLOK	RDEVBSC	RDEVCON
	RDEVDED	RDEVDISA	RDEVDISB	RDEVFLAG	RDEVMAX	RDEVNICL	RDEVNRDY	RDEVRSVD	RDEVSTAT	RDEVWAII	R0	R1	R10
	R11	R12	R13	R14	R15	R2	R3	R4	R5	R6	R7	R8	R9
	SAVEAREA	SAVER2	SILI	SYSTEM	TEMPR3	TEMPSAVE	TRQBLOK	VMBLOK	VMGENIO	VMLOGOFF	VMRSTAT	VMTLEND	VMTRMID
	VMTTIME												

Module	External References (Labels and Modules)												
DMKRNH	ABORT	ALARM	ASYSVM	ATTN	BALRSAVE	BLANKS	BUSOUT	BUSY	CACTDEV	CACTLIN	CACTLTR	CC	CCDESMD
	CDC	CHC	CKPBITS	CKPBKSZ	CKPBLOK	CKPNAME	CKPRMAX	CKPSIZE	CLASSPEC	CMDREJ	CNTLBTU	CONACTV	CONADDR
	CONCCW1	CONCCW2	CONCCW3	CONCNT	CONCNTL	CONCOMND	CONDATA	CONDCNT	CONDEST	CONESCP	CONEXTR	CONFLAG	CONOUTPT
	CONPARM	CONPNT	CONRESP	CONRETN	CONRTAG	CONRTRY	CONSPLT	CONSRID	CONSTAT	CONSYNC	CONSYSR	CONTACT	CONTASK
	CONTCMD	CONTSIZE	CONTSKSZ	CONUSER	CPEXADD	CPEXBLOK	CPEXSIZE	CRESCND	CRESERL	CRESIMD	CSETDSM	CTRMLTR	DE
	DFRET	DISCEOC	DISCNCT	DMKBLDVM	DMKCFMAT	DMKCFMBK	DMKCNSED	DMKCPVAE	DMKCVTBH	DMKCVTDT	DMKDSPCH	DMKERMSG	DMKFREE
	DMKFRET	DMKIOERN	DMKIOSQR	DMKNLDMP	DMKNLDR	DMKQCNCL	DMKQCNET	DMKQCNTO	DMKQCNWT	DMKRIORN	DMKSCNAU	DMKSCNRU	DMKSTKCP
	DMKVSPRT	EDIT	ERRMSG	F1	F16	F256	F4	F4095	F60	F8	IL	INHIBIT	INTREQ
	IOBCAW	IOBCC1	IOBCC3	IOBCP	IOBCSW	IOBFLAG	IOBIOER	IOBIRA	IOBLINK	IOBLOK	IOBMISC	IOBMISC2	IOBRADD
	IOBRCAW	IOBRCNT	IOBRSTRT	IOBSIZE	IOBSPEC	IOBSTAT	IOBUNSL	IOBUSER	IOERBLOK	IOERDATA	IOEREXT	IOERSIZE	IPLREQ
	LOGDROP	LOGHOLD	NICATOF	NICATTN	NICBLOK	NICCIBM	NICDED	NICDISA	NICDISB	NICENAB	NICEPMD	NICERLK	NICFLAG
	NICLINE	NICLTRC	NICMTA	NICNAME	NICNTRL	NICPSUP	NICQPNT	NICRCNT	NICSESN	NICSIZE	NICSTAT	NICTELE	NICTERM
	NICTYPE	NICUSER	NOAUTO	NORET	OPERATOR	PCI	PRGC	PRIORITY	PRTC	PSA	RDBUFLN	RDBUFNO	RDEVAUTO
	RDEVBLOK	RDEVBUSY	RDEVCKPT	RDEVCON	RDEVDED	RDEVDISA	RDEVFLAG	RDEVLCEP	RDEVLNCP	RDEVMAX	RDEVNCP	RDEVNICL	RDEVNRDY
	RDEVRCVY	RDEVRSVD	RDEVSCED	RDEVSLOW	RDEVSTAT	RDEVTBTU	RDEVTYPC	RDEVWAIT	READNRM	R0	R1	R10	R11
	R12	R13	R14	R15	R2	R3	R4	R5	R6	R7	R8	R9	SAVEAREA
	SAVER0	SAVER1	SAVER2	SILI	SYSTEM	TEMPSAVE	TRACCURR	TRACEND	TRACFLG2	TRACSTRT	TRAC11	TYP3705	UC
	UCASE	UE	VMBLOK	VMCFWAIT	VMLOGON	VMMCPENV	VMMLEVEL	VMRSTAT	VMTRMID	VMTTIME	VMUSER	WRITBRK	WRITEOT
	WRITNRM	XRIGHT16	ZEROES										
DMKRPA	ACORETBL	AVMREAL	BRING	CORBPNT	CORCFLCK	CORFLAG	CORFPNT	CORIOLCK	CORPGPNT	CORSWPNT	CORTABLE	CPEXADD	CPEXBLOK
	CPEXFPNT	CPEXR0	CPEXSIZE	DEFER	DMKFREE	DMKPAGIO	DMKPGTPR	DMKPGTSP	DMKPTRAN	DMKPTRFT	DMKPTRUL	DMKPTRWQ	DMKSCHDL
	FFS	F1	F4	IOERETN	LOCK	PAGCORE	PAGINVAL	PAGREF	PSA	R0	R1	R11	R13
	R14	R15	R2	R3	R5	R7	R9	SAVEAREA	SAVER1	SAVER2	SAVEWRK1	SWPCYL	SWPFLAG
	SWPRECMP	SWPSHR	SWPTRANS	SYSTEM	VMBLOK	VMESTAT	VMINVPAG	VMPAGES	VMPGWAIT	VMRSTAT	VMWCNT		
DMKRSE	ACNTBACK	ACNTBLOK	ATTN	BUSOUT	CC	CCC	CDC	CE	CHC	CLASURI	CLASURO	CMDREJ	CUE
	DATACHK	DE	DMKFREE	DMKFRET	DMKICEST	DMKMSWR	DMKRSP83	EQCHK	F1	F3	F4	F7	F8
	IFCC	INTREQ	IOBCAW	IOBCC1	IOBCC3	IOBCSW	IOBERP	IOBFATAL	IOBFLAG	IOBIOER	IOBLOK	IOBMISC2	IOBRCAW
	IOBRCNT	IOBRSTRT	IOBSTAT	IOERACT	IOERBLOK	IOERCEMD	IOERCSW	IOERDATA	IOERDEPD	IOERDERD	IOERECSW	IOERERP	IOERETRY
	IOEREXT	IOERFLG1	IOERFLG2	IOERFLG3	IOERIGN	IOERIND3	IOERINFO	IOERNUM	IOEROVFL	IOERPEND	IOERPNT	IOERREAD	IOERSIZE
	IOERXERP	PCI	PRGC	PRTC	PSA	RDEVACNT	RDEVBACK	RDEVBLOK	RDEVFLAG	RDEVIOER	RDEVNRDY	RDEVRSTR	RDEVSPL
	RDEVSTAT	RDEVTERM	RDEVTYPC	RDEVTYPE	R0	R1	R10	R11	R13	R14	R15	R2	R3
	R4	R5	R6	R7	R8	R9	SAVEAREA	SAVEWRK1	SILI	SM	TYPPUN	TYP1403	TYP1443
	TYP2501	TYP2520P	TYP2540P	TYP2540R	TYP3211	TYP3505	UC	VMBLOK	VMCF	VMOSTAT	XOBRCCW1	XOBRCCW2	XOBRCCW3
	XOBRCCW4	XOBREXT	XOBRFLAG	XOBRMIS1	XOBRMIS2	XOBRRT1	XOBRRT2	XOBRRT3	XOBRRT4	XOBRRT5	XOBRRT6	XOBRSIZE	XOBRSTAT
	XOBRT1	XOBRT2	XOBRT3	XOBR010	XOBR150	XOBR180	XOBR512	ZEROES					

Module	External References (Labels and Modules)												
DMKRSP	ALARM	BLANKS	BRING	BUFCNT	BUFFER	BUFNXT	BUFSIZE	CC	CCC	CDC	CE	CHGSFB	CLASURI
	CLASURO	CPEXADD	CPEXBLOK	CPEXSIZE	DE	DEFER	DMKACOPU	DMKCKSPL	DMKCSOSD	DMKCVTBD	DMKCVTBH	DMKCVTDT	DMKDSPCH
	DMKERMSG	DMKFREE	DMKFRET	DMKIOSQR	DMKOPRWT	DMKPGTSG	DMKPGTVG	DMKPGTVR	DMKPTRAN	DMKQCNWT	DMKRPAGT	DMKRPAPT	DMKRSERR
	DMKSCNFD	DMKSCNRD	DMKSCNRN	DMKSCNRU	DMKSEPSP	DMKSPLCR	DMKSPLDL	DMKSPLOR	DMKSTKCP	DMKSYSOC	DMKSYSOW	DMKSYSRM	DMKSYSTP
	DMKSYSWM	DMKTMRPT	DMKUDRFU	F24	F4	F4095	F4096	F8	IFCC	IL	IOBCAW	IOBCC1	IOBCP
	IOBCSW	IOBERP	IOBFATAL	IOBFLAG	IOBIOER	IOBIRA	IOBLOK	IOBMISC	IOBRADD	IOBRCAW	IOBRCNT	IOBRSTRT	IOBSIZE
	IOBSTAT	IOERBLOK	IOERCSW	IOERDATA	IOERDEPD	IOERDERD	IOERERP	IOEREXT	IOERFLG1	IOERSIZE	LOCK	NORET	OPERATOR
	PSA	RDEVACNT	RDEVBACK	RDEVBLOK	RDEVBUSY	RDEVCLAS	RDEVDED	RDEVDISA	RDEVDRAN	RDEVFLAG	RDEVIOER	RDEVLOAD	RDEVNRDY
	RDEVRSTR	RDEVSEP	RDEVSPAC	RDEVSPL	RDEVSTAT	RDEVTERM	RDEVTYPC	RDEVTYPE	RECBLOK	RECCYL	RECMAP	RECPNT	RECSIZE
	RECUSED	RSPDPAGE	RSPLCTL	RSPMISC	RSPRPAGE	RSPRSTRT	RSPSFBLK	RSPSIZE	RSPVPAGE	R0	R1	R10	R11
	R12	R13	R14	R15	R2	R3	R4	R5	R6	R7	R8	R9	SAVEAREA
	SAVEREGS	SAVER0	SFBCLAS	SFBCOPY	SFBFILID	SFBFLAG	SFBFLAG2	SFBFNAME	SFBFTYPE	SFBLAST	SFBLOK	SFBORIG	SFBPNT
	SFBRECER	SFBRECNO	SFBRECOK	SFBRECS	SFBREQUE	SFBRSTRT	SFBSHOLD	SFBSIZE	SFBSTART	SFBTICER	SFBTYPE	SFBUHOLD	SFBUSER
	SILI	SKIP	SPLINK	SPNXTPAG	SPPREPAG	SPRECNUM	SPRMISC	SPSIZE	SYSTEM	TYPPRT	TYPPUN	TYPRDR	TYP2540R
	UC	UE	VMBLOK										

Module	External References (Labels and Modules)												
DMKSAV	ALARM	CAW	CC	CE	CSW	DE	DMKCKP	DMKCKPRS	DMKCKPST	DMKCKPT	DMKCPICD	DMKCPINT	DMKCVTBH
	DMKOPRWT	DMKSYSNU	DMKSYSRS	DMKSYSTP	DMKSYSTZ	DMKSYSVL	EXNPSW	F1	F2	F3	F4	INTREQ	INTTIO
	IONPSW	IOOPSW	MCNPSW	PRNPSW	PSA	PSTARTSV	R0	R1	R10	R11	R12	R13	R14
	R15	R2	R3	R4	R5	R6	R7	R8	R9	SILI	SKIP	TEMPR2	TEMPR4
	TEMPSAVE	TYP2305	TYP2314	TYP3330	TYP3340	TYP3350							

Module	External References (Labels and Modules)												
DMKSCH	ACORETBL	AVMREAL	BALRSAVE	BALR11	BRING	CORBPNT	CORCFLCK	CORFLAG	CORFPNT	CORIOLCK	CORRSV	CORSHARE	CORTABLE
	DEFER	DMKDSPCH	DMKDSPNP	DMKFREE	DMKFRET	DMKMIDNT	DMKPTRAN	DMKPTRFL	DMKPTRRL	DMKPTRU1	ECBLOK	EXTCPTMR	EXTCPTRQ
	FFS	F0	F1	F10	F15	F3	F4	F5	IDLEWAIT	IONTWAIT	MNCLSCH	MNCOAEL	MNCOAQ
	MNCODQ	PAGCORE	PAGELOAD	PAGEWAIT	PAGINVAL	PAGREF	PROBTIME	PSA	R0	R1	R10	R11	R12
	R13	R14	R15	R2	R3	R4	R5	R6	R7	R8	R9	TEMPSAVE	TIMER
	TRACCURR	TRACEND	TRACFLG1	TRACSTRT	TRAC08	TRAC09	TRQBBPNT	TRQBFPNT	TRQBLOK	TRQBQUE	TRQBTOD	TRQBVAL	VMAEX
	VMAEXP	VMBLCK	VMCOMP	VMCPUTMR	VMCPWAIT	VMDROP1	VMDSP	VMDSTAT	VMECEXT	VMELIG	VMEPRIOR	VMHIPRI	VMINQ
	VMIOINT	VMLONGWT	VMLOPRI	VMNORUN	VMPAGES	VMPEND	VMPGREAD	VMPGRINQ	VMPRIDSP	VMPSTAT	VMPSWAIT	VMPXINT	VMQBPNT
	VMQFPNT	VMQLEVEL	VMQPRIOR	VMQSEND	VMQSTAT	VMQ1	VMRDINQ	VMRON	VMRPAGE	VMRPRIOR	VMRSTAT	VMRUN	VMSEG
	VMSTEALS	VMSTMPI	VMSTMPT	VMTIMER	VMTLEVEL	VMTMINQ	VMTMOUTQ	VMTMRINT	VMTODINQ	VMTRQBLK	VMTSEND	VMTTIME	VMUPRIOR
	VMVTIME	VMV370R	VMWSCHG	VMWSERNG	VMWSPROJ	XINTBLOK	XINTCODE	XINTNEXT	XINTPARM	XINTSIZE	XINTSORT	ZEROES	

Module	External References (Labels and Modules)												
DMKSCN	ARIOCH	ARIOCT	ARIOCU	ARIODC	ARIODV	ASYSVM	BALRSAVE	BALR1	BALR2	BALR3	BALR8	BLANKS	BUFFER
	BUFNXT	CLASDASD	CLASSPEC	CLASTERM	CLASURI	CLASURO	FFS	FTR2311B	FTR2311T	F0	F5	F7	PSA
	RCHADD	RCHBLOK	RCHCUTBL	RCUADD	RCUBLOK	RCUCHA	RCUDVTBL	RCUPRIME	RCUSUB	RCUTYPE	RDEVADD	RDEVBLOK	RDEVCUA
	RDEVDED	RDEVDISA	RDEVFLAG	RDEVLNKS	RDEVMOUT	RDEVSER	RDEVSIZE	RDEVSTAT	RDEVTYPC	R0	R1	R10	R11
	R14	R15	R2	R3	R4	R5	R6	R7	R8	R9	TYPCTCA	TYPIBM1	TYPPRT
	TYPPUN	TYPRDR	TYPTELE2	TYP2311	TYP2700	TYP3210	TYP3705	UDEVBLOK	UDEVFTR	UDEVRELN	VCHADD	VCHBLOK	VCHCUTBL
	VCUADD	VCUBLOK	VCUDVTBL	VDEVADD	VDEVBLOK	VDEVDED	VDEVFLAG	VDEVLINK	VDEVRDO	VDEVREAL	VDEVRELN	VDEVSIZE	VDEVSTAT
	VDEVTYPC	VDEVTYPE	VDEVUSER	VMBLOK	VMCHSTRT	VMCHTBL	VMCUSTRT	VMDVCNT	VMDVSTRT	VMLOGOFF	VMLOGON	VMPNT	VMRSTAT
	VMUSER	ZEROES											

Module	External References (Labels and Modules)												
DMKSEP	BRING	CC	DEFER	DMKBOXBX	DMKCPEID	DMKCVTBD	DMKCVTBH	DMKCVTDT	DMKDSPCH	DMKFRET	DMKIOSQR	DMKPGTVG	DMKPGTVR
	DMKPTRAN	DMKPTRUL	DMKSCNRD	IOBCAW	IOBCSW	IOBFATAL	IOBFLAG	IOBIRA	IOBLINK	IOBLOK	IOBMISC	IOBMISC2	IOBRSTRT
	IOBSIZE	IOBSTAT	LOCK	PSA	RDEVBLOK	RDEVFLAG	RDEVLOAD	RDEVSEP	RDEVTYPE	R0	R1	R10	R11
	R12	R13	R14	R15	R2	R3	R4	R5	R6	R7	R8	R9	SAVEAREA
	SAVEREGS	SAVER10	SAVER8	SAVEWRK1	SAVEWRK2	SAVEWRK5	SAVEWRK7	SAVEWRK8	SAVEWRK9	SFBCLAS	SFBDATE	SFBDIST	SFBFILID
	SFBFLAG2	SFBFNAME	SFBLOK	SFBORIG	SFBRECNO	SFBRSTRT	SFBTIME	SFBUSER	SILI	SKIP	SYSTEM	TYPPUN	UE
	VMBLOK												
DMKSEV	CCC	CCHCHNL	CCHCMDV	CCHCNTB	CCHCPU	CCHDAV	CCHDI	CCHINTFC	CCHLOG70	CCHREC	CCHSTG	CCHUSV	COMPFES
	COMPSEL	COMPSYS	CSW	FPS	F7	F8	HIOCCH	IFCC	IGBLAME	IGPRGFLG	IGTERMSQ	IGVALIDB	INTERCCH
	IOERBLOK	PSA	RTCODE1	RTCODE2	RTCODE3	RTCODE4	RTCODE5	RTCODE7	R0	R1	R12	R13	R14
	R15	R2	R3	R4	R9	SAVEAREA	SAVEWRK1	SAVEWRK9	TERMSYS	TIOCCH	XRIGHT16		
DMKSIX	CCHCHNL	CCHCMDV	CCHCNTB	CCHCPU	CCHDAV	CCHDI	CCHINTFC	CCHLOG60	CCHREC	CCHSTG	CCHUSV	COMPFES	COMPSEL
	CSW	FPS	F1	F7	F8	HIOCCH	IFCC	IGBLAME	IGPRGFLG	IGTERMSQ	IGVALIDB	IOERBLOK	PSA
	RTCODE1	RTCODE2	RTCODE3	RTCODE4	RTCODE5	R0	R1	R12	R13	R14	R15	R2	R3
	R4	R9	SAVEAREA	SAVEWRK1	SAVEWRK9	TERMSYS	TIOCCH	XRIGHT16					
DMKSNC	BRING	CCPADDR	CCPARM	CCPNAME	CCPPSIZE	CCPSIZE	DEFER	DMKERMSG	DMKPTRAN	DMKPTRUL	DMKRNTBL	DMKRPAPT	DMKSCNVS
	F1	F256	F4096	IOERETN	LOCK	NCPNAME	NCPPAGCT	NCPPNT	NCPSTART	NCPTBL	NCPVOL	PSA	RDEVBLOK
	RDEVCODE	RDEVFLAG	RDEVOWN	RDEVTYPE	R0	R1	R10	R11	R13	R2	R3	R4	R5
	R6	R7	R8	R9	SAVEAREA	SAVER0	SAVER2	SAVER6	SAVEWRK1	SAVEWRK2	SAVEWRK3	SAVEWRK4	SAVEWRK5
	SAVEWRK6	SAVEWRK8	SAVEWRK9	SYSTEM	TYP2314	TYP3330	TYP3350	VMBLOK					
DMKSPL	ACCTBLOK	ACCTDIST	ACCTUSER	ACORETBL	ADDSFB	ARIODV	ARIOPR	ARIOPU	ARSPPR	ARSPPU	ARSPRD	ASYSVM	BLANKS
	BRING	CC	CHGSFB	CLASURI	CPEXADD	CPEXBLOK	CPEXREGS	CPEXSIZE	DE	DEFER	DELSFB	DMKCKSPL	DMKCVTBD
	DMKCVTDT	DMKDRDDD	DMKDSPCH	DMKFREE	DMKFRET	DMKIOSQR	DMKPGTSD	DMKPGTSG	DMKPGTSR	DMKPGTVG	DMKPTRAN	DMKPTRLK	DMKPTRUL
	DMKQCNWT	DMKRPAGT	DMKRPAPT	DMKRSPDL	DMKRSPEX	DMKRSPHQ	DMKRSPID	DMKSCNAU	DMKSTKCP	DMKSTKIO	DMKSYSOC	DMKSYSOW	DMKUDRFU
	DMKUDRRD	DMKUDRRV	DMKVIOIN	FTR70MB	F0	F1	F2	F3	F4	IOBCAW	IOBCP	IOBCSW	IOBCYL
	IOBFATAL	IOBFLAG	IOBIRA	IOBLINK	IOBLCK	IOBMISC2	IOBRADD	IOBSIZE	IOBSTAT	IOBUSER	IOBVADD	LOCK	NORET
	OWNDLIST	OWNDRDEV	PCHCHN	PRTCHN	PSA	RDEVACNT	RDEVBLOK	RDEVCLAS	RDEVDED	RDEVDISA	RDEVDRAN	RDEVFLAG	RDEVFTR
	RDEVSPL	RDEVSTAT	RDEVTYPE	RDRCHN	RECBLCK	RECPNT	RECSIZE	RSPDPAGE	RSPLCTL	RSPRPAGE	RSPSFBLK	RSPSIZE	RSPVPAGE
	R0	R1	R10	R11	R12	R13	R14	R15	R2	R3	R4	R5	R6
	R7	R8	R9	SAVEAREA	SAVER11	SAVER7	SAVER8	SAVER9	SAVEWRK1	SAVEWRK2	SAVEWRK8	SFBCLAS	SFBCOPY
	SFBDATE	SFBDIST	SFBDUMP	SFBFILID	SFBFIRST	SFBFLAG	SFBFLAG2	SFBFNAME	SFBHOLD	SFBLAST	SFBLOK	SFBNOHLD	SFBORIG
	SFBPNT	SFBPURGE	SFBRECER	SFBRECS	SFBRECSZ	SFBREQUE	SFBRSTRT	SFBSHOLD	SFBSIZE	SFBSTART	SFBTIME	SFBTYPE	SFBUHOLD
	SFBUSER	SHQBLOK	SHQSHOLD	SHQUSER	SILI	SKIP	SPLINK	SPSIZE	SYSTEM	TYPPRT	TYPPUN	TYPRDR	TYP1052
	TYP2314	TYP3210	TYP3211	TYP3340	TYP3350	UDBFBLOK	UDBFSIZE	UDBFVADD	UDIRBLOK	UDIRDISP	UMACBLOK	UMACDIST	VCHADD
	VCHBLOK	VCHCUTBL	VCUADD	VCUBLOK	VCUDVTBL	VDEVADD	VDEVBLOK	VDEVCLAS	VDEVCOPY	VDEVCSW	VDEVEXTN	VDEVFOR	VDEVHOLD
	VDEVPEND	VDEVSFLG	VDEVSPL	VDEVSTAT	VDEVTYPC	VDEVTYPE	VDEVXFER	VMACOUNT	VMBLOK	VMCHSTRT	VMCHTBL	VMCUSTRT	VMDIST
	VMDVSTRT	VMMLEVEL	VMMSGON	VMTTIME	VMUSER	VSPLCTL	VSPSFBLK	VSPSIZE	VSPVPAGE	VSPXBLOK	VSPXTAG	VSPXTGLN	VSPXXUSR
	ZEROES												

Module	External References (Labels and Modules)												
DMKSSP	ARIOCH	ARIOCT	ARIOCU	ARIODV	ATTN	BUSY	CAW	CC	CD	CE	CLASDASD	CLASGRAF	CLASTAPE
	CLASTERM	CLASURI	CLASURO	CPUID	CPUVERSN	CSW	CUE	DE	DMKCPINT	DMKCVTBH	DMKCVTHB	DMKRIO	DMKRIOCH
	DMKRIOCN	DMKRIOCU	DMKRIODV	DMKRIOPR	DMKRIOPU	DMKRIORD	DMKSYSNU	FTRUCS	F4096	IONPSW	IOOPSW	MCNPSW	PRNPSW
	PSA	RCHBLOK	RCHCUTBL	RCHSIZE	RCUADD	RCUBLOK	RCUCHA	RCUCHB	RCUDVTBL	RCUSIZE	RCUTYPE	RDEVADD	RDEVBLOK
	RDEVCLAS	RDEVCUA	RDEVFTR	RDEVSIZE	RDEVTYPC	RDEVTYPE	R0	R1	R10	R11	R12	R13	R14
	R2	R3	R4	R5	R6	R7	R8	R9	SAVEAREA	SILI	SYSTEM	TYPPRT	TYPPUN
	TYP2314	TYP2540P	TYP2540R	TYP3066	TYP3210	TYP3277	TYP3330	TYP3340	TYP3350	UC	UE	XRIGHT16	
DMKSTK	CPEXBLOK	CPEXBPNT	CPEXFPNT	DMKDSPRQ	IOBBPNT	IOBFPNT	IOBLOK	PSA	R0	R1	R10	R14	R15
DMKSYM	DMKACO	DMKBLDRL	DMKBLDRT	DMKBLDVM	DMKBSCER	DMKCCHIS	DMKCCHRT	DMKCCH60	DMKCCWSB	DMKCCWTC	DMKCCWTR	DMKCDBDC	DMKCDBDI
	DMKCDBDM	DMKCDBDU	DMKCDSCP	DMKCDSTO	DMKCFDAD	DMKCFDLO	DMKCFGII	DMKCFGIP	DMKCFGSV	DMKCFMAT	DMKCFMBK	DMKCFMEN	DMKCFPRD
	DMKCFPRR	DMKCFSET	DMKCFTRM	DMKCKPT	DMKCNSED	DMKCNSEN	DMKCNSIC	DMKCNSIN	DMKCPBEX	DMKCPBNR	DMKCPBRS	DMKCPBRW	DMKCPBRY
	DMKCPBSR	DMKCPEID	DMKCPEND	DMKCPVAA	DMKCPVAC	DMKCPVAE	DMKCPVDS	DMKCPVEN	DMKCPSH	DMKCPVLK	DMKCPSRY	DMKCPSSH	DMKCPVUL
	DMKCQGEN	DMKCQPRV	DMKCQREY	DMKCQRFI	DMKCSOBS	DMKCSODR	DMKCSOFL	DMKCSOLD	DMKCSORP	DMKCSOSD	DMKCSOSP	DMKCSOST	DMKCSOVL
	DMKCSPCL	DMKCSPFR	DMKCSPHL	DMKCSPSP	DMKCSUCH	DMKCSUOR	DMKCSUPU	DMKCSUTR	DMKDASER	DMKDASRD	DMKDASSD	DMKDEFIN	DMKDGDDK
	DMKDIACP	DMKDIADR	DMKDIAL	DMKDIASM	DMKDMPDK	DMKDMPGR	DMKDMPRS	DMKDRDDD	DMKDRDER	DMKDRDMP	DMKDRDSY	DMKDSPAC	DMKDSPBC
	DMKDSPCC	DMKDSPCH	DMKDSPNP	DMKDSPQS	DMKDSPRQ	DMKEIG80	DMKEPSWD	DMKERMSG	DMKFREE	DMKFREHI	DMKFRELG	DMKFRELO	DMKFRELS
	DMKFRENP	DMKFRERS	DMKFRESV	DMKFRET	DMKFRETR	DMKGIOEX	DMKGRFEN	DMKGRFIC	DMKGRFIN	DMKHVCAL	DMKHVCDI	DMKIOEFM	DMKIOERR
	DMKIOF	DMKIOG	DMKIOSCT	DMKIOSHA	DMKIOSIN	DMKIOSQR	DMKIOSQV	DMKISMTR	DMKLNKIN	DMKLNKSB	DMKLOC	DMKLOGON	DMKLOGOP
	DMKMCCCL	DMKMCHAR	DMKMCHIN	DMKMCHMS	DMKMIDNT	DMKMONIO	DMKMONTH	DMKMSGEC	DMKMSGMS	DMKMSGWN	DMKMSWR	DMKNEMOP	DMKNETWK
	DMKNLDMP	DMKNLDR	DMKOPRWT	DMKPAGCC	DMKPAGIC	DMKPAGIO	DMKPAGPS	DMKPAGQ	DMKPAGSP	DMKPAGST	DMKPGS	DMKPGSPO	DMKPGSPP
	DMKPGTEN	DMKPGTPG	DMKPGTTM	DMKPGTTU	DMKPRGCT	DMKPRGC8	DMKPRGGR	DMKPRGIN	DMKPRGMC	DMKPRGRF	DMKPRGSM	DMKPRVLG	DMKPRVNC
	DMKPSADU	DMKPSAEX	DMKPSANS	DMKPSARG	DMKPSASV	DMKPTRAN	DMKPTRCT	DMKPTRFC	DMKPTRFF	DMKPTRPR	DMKPTRRC	DMKPTRRQ	DMKPTRSC
	DMKPTRSS	DMKPTRWQ	DMKQCNCL	DMKQCNET	DMKQCNRD	DMKQCNSY	DMKQCNTO	DMKQCNWT	DMKRGAIN	DMKRGBEN	DMKRGBIC	DMKRIOCH	DMKRIOCN
	DMKRIOCU	DMKRIODV	DMKRIOPR	DMKRIOPU	DMKRIORD	DMKRIORN	DMKRNHCT	DMKRNHIC	DMKRNHIN	DMKRNHND	DMKRNHTG	DMKRNHTR	DMKRPAGT
	DMKRPAPT	DMKRSERR	DMKRSPAC	DMKRSPCV	DMKRSPDL	DMKRSPER	DMKRSPEX	DMKRSPHQ	DMKRSPID	DMKRSPPR	DMKRSPPU	DMKRSPRD	DMKRSPUR
	DMKRSP83	DMKSCHAL	DMKSCHAP	DMKSCHAU	DMKSCHCP	DMKSCHCT	DMKSCHDL	DMKSCHIB	DMKSCHMD	DMKSCHN1	DMKSCHN2	DMKSCHPB	DMKSCHPD
	DMKSCHPG	DMKSCHPU	DMKSCHQ1	DMKSCHQ2	DMKSCHRL	DMKSCHRT	DMKSCHST	DMKSCHTQ	DMKSCHUB	DMKSCHW1	DMKSCHW2	DMKSCH80	DMKSEPHR
	DMKSEPSP	DMKSEV70	DMKSIX60	DMKSNCP	DMKSPLCR	DMKSPLCV	DMKSPLDL	DMKSPLDR	DMKSPLOR	DMKSPLOV	DMKSYSCS	DMKSYSLC	DMKSYSOC
	DMKSYSOP	DMKSYSOW	DMKSYSRM	DMKSYSRS	DMKSYSRV	DMKSYSVL	DMKSYSVM	DMKTAPER	DMKTDKGT	DMKTDKRL	DMKTMRTN	DMKTRACE	DMKTRCEX
	DMKTRMID	DMKUDRBV	DMKUDRDS	DMKUDRFD	DMKUDRFU	DMKUDRRD	DMKUDRRV	DMKUNTFR	DMKUNTIS	DMKUNTRN	DMKUNTRS	DMKUSODS	DMKUSOFF
	DMKUSOFL	DMKUSOFM	DMKUSOLG	DMKVATAB	DMKVATBC	DMKVATEX	DMKVATLA	DMKVATMD	DMKVATPX	DMKVATRN	DMKVATSX	DMKVCARD	DMKVCARS
	DMKVCASH	DMKVCAST	DMKVCATS	DMKVCHDC	DMKVCNEX	DMKVDBAT	DMKVDBDE	DMKVDREL	DMKVDSAT	DMKVDSDF	DMKVDSLK	DMKVERD	DMKVERO
	DMKVIOCT	DMKVIOCW	DMKVIOEX	DMKVIOIN	DMKVMA	DMKVMAPS	DMKVMASH	DMKVSPCO	DMKVSPCR	DMKVSPEX	DMKVSPRT	DMKVSPVP	DMKVSPWA
DMKTAP	CCC	CD	CDC	CHC	CUE	DE	DMKFREE	DMKFRET	DMKIOEST	DMKLOCKD	DMKLOCKQ	DMKMSWR	F1
	F15	F16	F2	F3	F4	F5	F6	F8	IDA	IFCC	IL	IOBCAW	IOBCP
	IOBCSW	IOBERP	IOBFATAL	IOBFLAG	IOBIOER	IOBLOK	IOBRCAW	IOBRCNT	IOBRSTRT	IOBSTAT	IOERACT	IOERADR	IOERBLOK
	IOERBSR	IOERCAN	IOERCLN	IOERCSW	IOERDATA	IOERDW	IOERERG	IOEREXT	IOERFLG1	IOERFLG2	IOERFLG3	IOERFSR	IOERIGNR
	IOERIND3	IOERIND4	IOERINFO	IOERLOC	IOERMSG	IOERMSW	IOERNUM	IOERORA	IOEROVFL	IOERPEND	IOERRBK	IOERREAD	IOERREW
	IOERSIZE	IOERSTRT	IOERSUPP	IOERVLD	IOERWRK	PRGC	PRTC	PSA	RDEVBLOK	RDEVIOER	RDEVNRDY	RDEVSTAT	RDEVTYPE
	R0	R1	R10	R13	R2	R3	R4	R5	R6	R7	R8	R9	SAVEAREA
	SAVEWRK2	SAVEWRK3	SILI	SKIP	TYP2401	TYP2415	TYP2420	TYP3410	TYP3420	UC	ZEROES		

Module	External References (Labels and Modules)												
DMKTDK	ALOCBLOK	ALOCCYL1	ALOCCYL2	ALOCMAP	ALOCPNT	CC	DMKDSPCH	DMKFREE	DMKFRET	DMKIOSQR	DMKPGTP0	DMKPGTP4	DMKPGTP5
	DMKPGTT0	DMKPGTT4	DMKPGTT5	DMKPGT4P	DMKPGT4T	DMKPGT5P	DMKPGT5T	FTR70MB	F255	F256	IOBCP	IOBCYL	IOBFLAG
	IOBLOK	IOBMISC	PSA	RDEVALLN	RDEVBLOK	RDEVFTR	RDEVPNT	RDEVTYPE	R0	R1	R10	R11	R12
	R13	R14	R15	R2	R3	R4	R5	R6	R7	R8	R9	SAVEAREA	SAVER0
	SAVER1	SAVER8	SAVEWRK2	SILI	TYP2314	TYP3330	TYP3340	TYP3350					
DMKTHI	ASYSVM	BLANKS	DFRET	DMKCVTBD	DMKCVTBH	DMKERMSG	DMKFREE	DMKFRET	DMKQCNWT	DMKSCHCO	DMKSCHCU	DMKSCHLI	DMKSCHSC
	DMKSCHS1	DMKSCHS2	DMKSCNFD	DMKSCNRD	DMKSCNVU	DMKTMRPT	F1	F3	F4	F60	F8	NORET	PSA
	RUNUSER	R0	R1	R10	R11	R13	R2	R3	R4	R5	R6	R7	R8
	R9	SAVEAREA	SAVER11	SAVEWRK1	SAVEWRK2	SAVEWRK3	SAVEWRK6	VDEVBLOK	VDEVREAL	VMACTDEV	VMBLOK	VMCLASSA	VMCLASSB
	VMCLASSC	VMCLASSD	VMCLASSE	VMCLASSF	VMCLASSG	VMCLEVEL	VMCRDS	VMDSTAT	VMELIG	VMEXWAIT	VMINQ	VMIOCNT	VMIOWAIT
	VMLINS	VMPAGES	VMPDISK	VMPDRUM	VMPGREAD	VMPGWAIT	VMPGWRIT	VMPNCH	VMPNT	VMPSWAIT	VMQLEVEL	VMQ1	VMRSTAT
	VMRUN	VMSTKO	VMTTIME	VMUSER	VMWSPROJ								
DMKTMR	BALR2	BRING	CPCREG0	C0	C1	DEFER	DMKDSPCH	DMKFREE	DMKFRET	DMKPRGSM	DMKPSAFP	DMKPSASP	DMKPTRAN
	DMKSCHN1	DMKSCHN2	DMKSCHRT	DMKSCHST	DMKSTKIO	DMKVATEX	DMKVATRN	ECBLOK	EXTCCTRQ	EXTCPTMR	EXTCPTRQ	EXTCR9	EXTPERAD
	EXTSHCR0	F4	F4095	F5	F60	F7	F8	PERSALT	PSA	R0	R1	R10	R11
	R12	R14	R15	R2	R3	R4	R5	R6	R7	R8	R9	TRANMODE	TREXCR9
	TREXPERA	TREXT	TRQBFPNT	TRQBLOK	TRQEQUE	TRQBTOD	TRQBVAL	VMBLOK	VMCPUTMR	VMDSP	VMDSTAT	VMECEXT	VMESTAT
	VMEXTCM	VMEXWAIT	VMGPRS	VMINQ	VMINST	VMINVPAG	VMPEND	VMPERCM	VMPERPND	VMPRGIL	VMPSTAT	VMPSW	VMPXINT
	VMQLEVEL	VMQ1	VMRSTAT	VMRUN	VMSEG	VMTLEVEL	VMTMOUTQ	VMTMRINT	VMTRCTL	VMTREXT	VMTRPER	VMVTIME	VMV370R
	XINTBLCK	XINTNEXT	XINTPARM	XINTSIZE	XINTSORT	ZEROES							
DMKTRA	CSW	C1	DMKERMSG	DMKFREE	DMKFRET	DMKLOCKD	DMKLOCKQ	DMKQCNWT	DMKSCNFD	DMKTRCIT	DMKTRCPB	FFS	F3
	F8	NORET	PSA	R0	R1	R11	R12	R13	R14	R15	R2	R3	R4
	R5	R9	SAVEAREA	SAVER2	SAVEWRK1	SAVEWRK2	SAVEWRK7	TREXANSI	TREXBRAN	TREXCCW	TREXCSW	TREXCTL	TREXINST
	TREXIN1	TREXPRNT	TREXRUNF	TREXSIZE	TREXT	TREXTERM	VMBLOK	VMCFWAIT	VMEXWAIT	VMPSW	VMRSTAT	VMSEG	VMTRBRIN
	VMTRCTL	VMTREX	VMTREXT	VMTRINT	VMTRIO	VMTRPER	VMTRPRG	VMTRPRV	VMTRSIO	VMTRSVC	WAIT	XRIGHT16	
DMKTRC	APTRAN	BLANKS	BRING	CAW	CLASDASD	CLASGRAF	CLASSPEC	CLASTERM	CLASURI	CLASURO	CPCREG0	CSW	C0
	C1	DEFER	DMKCCWSE	DMKCFMBK	DMKCVTBH	DMKFRET	DMKLOCKD	DMKLOCKQ	DMKNEMOP	DMKPSARR	DMKPSARS	DMKPSARX	DMKPSASC
	DMKPSASP	DMKQCNWT	DMKSCNRD	DMKSCNRN	DMKSCNVN	DMKSCNVU	DMKSYSRM	DMKVATRN	DMKVMACF	DMKVSPRT	ECBLOK	EXTCR0	EXTMASK
	EXTMODE	EXTSHCR0	FFS	F1	F15	F16	F2	F240	F3	F4	F60	F8	IDA
	INTSVCL	IOBCAW	IOBCSW	IOBLOK	IOBRADD	IOBSTAT	IOBVADD	IOMASK	NCRET	PERMODE	PSA	RCWCCW	RCWGEN
	RCWINVL	RCWPNT	RCWRCNT	RCWTASK	RCWVCAW	RCWVCNT	R0	R1	R10	R11	R12	R13	R14
	R15	R2	R3	R4	R5	R6	R7	R8	R9	SAVEAREA	SAVER0	SAVER1	SAVER2
	SAVER4	SAVER5	SAVEWRK1	SAVEWRK2	SAVEWRK3	SAVEWRK4	SAVEWRK5	SAVEWRK6	SAVEWRK7	SAVEWRK8	SAVEWRK9	SVCNPSW	SVCOPSW
	TRANMODE	TREXANSI	TREXBRAN	TREXBUFF	TREXCCW	TREXCSW	TREXCTL1	TREXCTL2	TREXFLAG	TREXINST	TREXIN1	TREXIN2	TREXLCNT
	TREXNSI	TREXPRNT	TREXRUNF	TREXSIZE	TREXSVC1	TREXSVC2	TREXT	TREXTERM	TREXVAT	VDEVBLOK	VDEVCSW	VDEVDED	VDEVREAL
	VDEVSTAT	VDEVTYPC	VMBLOK	VMCFWAIT	VMECEXT	VMESTAT	VMEXTCM	VMEXWAIT	VMGPRS	VMINST	VMINVPAG	VMLOGOFF	VMPEND
	VMPERPND	VMPSTAT	VMPSW	VMRSTAT	VMSEG	VMSTOR	VMTRBRIN	VMTRCTL	VMTREX	VMTREXT	VMTRINT	VMTRIO	VMTRPRG
	VMTRPRV	VMTRSIO	VMTRSVC	VMVCR0	VMV370R	WAIT	X2048BND	ZEROES					

Module	External References (Labels and Modules)												
DMKTRM	F7	PSA	RDEVATOF	RDEVBLOK	RDEVCORR	RDEVFLAG	RDEVIDNT	RDEVPTTC	RDEVTFLG	RDEVTMCD	R0	R1	R13
	R2	R3	R4	R5	R8	SAVEAREA							
DMKUCB	CC	SILI											
DMKUCS	CC	SILI											
DMKUDR	ACORETBL	ALARM	ARIODV	ASYSLC	BLANKS	BRING	CC	CORBPNT	CORFPNT	CORPGPNT	CORTABLE	DEFER	DMKDSPCH
	DMKFREE	DMKFRET	DMKIOSQR	DMKLOCKD	DMKLOCKQ	DMKPGTVG	DMKPGTVR	DMKPTRAN	DMKPTRFT	DMKQCNWT	DMKRPAGT	DMKSYSOC	DMKSYSOW
	DMKSYSPL	DMKSYSUD	F1	F256	F4096	F8	IOBCAW	IOBCP	IOBFATAL	IOBFLAG	IOBIRA	IOBLOK	IOBMISC
	IOBMISC2	IOBSIZE	IOBSTAT	IOBUSER	NOADD	NORET	OPERATOR	OWNDLIST	OWNDRDEV	OWNDVSER	PAGINVAL	PSA	R0
	R1	R10	R11	R12	R13	R14	R15	R2	R3	R4	R5	R6	R7
	R8	R9	SAVEAREA	SAVER0	SAVER2	SAVEWRK2	SILI	SYSLOCS	SYSTEM	UDBFBLOK	UDBFDASD	UDBFVADD	UDBFWORK
	UDEVADD	UDEVBLOK	UDEVDASD	UDEVDISP	UDEVSIZE	UDIRBLOK	UDIRDASD	UDIRDISP	UDIRSIZE	UDIRUSER	UMACBLOK	UMACDASD	UMACDISP
	VMBLOK	VMESTAT	VMEXTCM	VMPSW	ZEROES								
DMKUNT	ACORETBL	BALRSAVE	CCC	CD	CDC	CHC	CORPGPNT	CORTABLE	DMKDSPCH	DMKFRET	DMKPTRFT	DMKPTRUL	DMKSTKIO
	DMKSYSRM	FFS	F0	F1	F15	F16	F240	F4	F7	F9	IDA	IFCC	IOBCAW
	IOBCC3	IOBCSW	IOBFLAG	IOBIRA	IOBLCK	IOBMISC	IOBRES	IOBSTAT	IOERBLOK	IOERCYLR	IOERDATA	IOERFLG2	IOERLEN
	PRGC	PRTC	PSA	RCWADDR	RCWCCNT	RCWCCW	RCWCNT	RCWCOMND	RCWCTL	RCWFLAG	RCWGEN	RCWHMR	RCWIO
	RCWPNT	RCWRCNT	RCWSHR	RCWTASK	RCWVCAW	RCW2311	RDEVBLOK	RDEVMDL	R0	R1	R10	R11	R12
	R13	R14	R15	R2	R3	R4	R5	R6	R7	R8	R9	SAVEAREA	SAVEWRK1
	SAVEWRK2	SAVEWRK3	SAVEWRK5	SAVEWRK6	SAVEWRK9	SKIP	TYP2305	TYP3330	TYP3340	TYP3350	UC	VDEVBLOK	VDEVCSW
	VDEVFLAG	VDEVREAL	VDEVRELN	VDEVTYPE	VDEV231B	VMBLOK	XPAGNUM	X2048BND	ZEROES				
DMKUSO	ACCTLENG	ADSPCH	ARSPPR	ARSPPU	ARSPRD	ASYSLC	ASYSOP	ASYSVM	BLANKS	CLASGRAF	CLASSPEC	CLASTERM	CPEXADD
	CPEXBLOK	CPEXR0	CPEXR11	CPEXR12	CPEXSIZE	DELPAGES	DELSEGS	DMKACOFF	DMKACOTM	DMKBLDRL	DMKCFPRR	DMKCVTBD	DMKCVTBH
	DMKCVTDT	DMKDSPCH	DMKERMSG	DMKFREE	DMKFRELO	DMKFRENP	DMKFRERS	DMKFRET	DMKLOCKD	DMKLOCKQ	DMKPERT	DMKPGSPO	DMKPGTP5
	DMKPTRRL	DMKPTRRU	DMKQCNWT	DMKSCHAU	DMKSCHDL	DMKSCHRT	DMKSCNAU	DMKSCNFD	DMKSCNRD	DMKSCNRN	DMKSCNVU	DMKSTKCP	DMKSYSDW
	DMKSYSNM	DMKSYSTI	DMKTRCND	DMKVATBC	DMKVDREL	DMKVMAPS	DMKVMASH	DMKVSPWA	ECBLOK	EXTCCTRQ	EXTCPTRQ	EXTSIZE	FFS
	F15	F8	LASTUSER	LOGDROP	LOGHCLD	MICSIZE	NORET	OPERATOR	PRIORITY	PSA	RDEVBLOK	RDEVPAGE	RDEVPNT
	RDEVRECS	RDEVTYPC	RDEVTYPE	RECSIZE	RUNUSER	R0	R1	R10	R11	R13	R14	R15	R2
	R3	R4	R5	R6	R7	R8	R9	SAVEAREA	SAVERETN	SAVER11	SAVEWRK1	SAVEWRK2	SAVEWRK3
	SFBFLAG	SFBINUSE	SFBLOK	SFBSIZE	SYSLOCS	TREXSIZE	TRQBFPNT	TRQBLOK	TRQBSIZE	TYPESC	TYP3705	VCHBLOK	VCHSIZE
	VCUBLOK	VCUDVTBL	VCUSIZE	VDEVADD	VDEVBLOK	VDEVSIZE	VMACCOUN	VMACNT	VMACOUNT	VMAEXP	VMBLOK	VMBSIZE	VMCHCNT
	VMCHSTRT	VMCHTBL	VMCOMND	VMCUCNT	VMDELAY	VMDISC	VMDVCNT	VMDVSTRT	VMECEXT	VMFSTAT	VMKILL	VMLOGOFF	VMLOGON
	VMMICRO	VMMLEVEL	VMMSGON	VMOSTAT	VMPFUNC	VMPNT	VMPSTAT	VMPSW	VMQLEVEL	VMRPAGE	VMRSTAT	VMSEG	VMSHR
	VMSIZE	VMSYSOP	VMTERM	VMTRCTL	VMTREXT	VMTRMID	VMTRPER	VMTRQBLK	VMTTIME	VMUSER	VMVTERM	VMV370R	VMWNGON
	WAIT	ZEROES											

Module	External References (Labels and Modules)												
DMKVAT	BALRSAVE	BALR12	BALR13	BALR14	BRING	CPCREG0	C1	DEFER	DMKDSPCH	DMKFREE	DMKFRET	DMKPERIL	DMKPRGSM
	DMKPTRAN	DMKPTRRW	ECBLOK	EXTARCH	EXTCOPY	EXTCR0	EXTCR1	EXTSEGLN	EXTSHCR0	EXTSHCR1	EXTSHLEN	EXTSHSEG	EXTSTOLD
	EXTVSEGS	F4	PGADDR	PGBLOK	PGBSIZE	PGPNT	PSA	R0	R1	R10	R11	R12	R13
	R14	R15	R2	R3	R4	R5	R6	R7	R8	R9	SAVEAREA	SAVEREGS	SAVERETN
	SAVER0	SAVER1	SAVER12	SAVER13	SAVER2	SAVER3	TREXADD	VMBADCR0	VMBLOK	VMECEXT	VMESTAT	VMEXWAIT	VMINVPAG
	VMINVSEG	VMIOPND	VMNEWCR0	VMPAGEX	VMPEND	VMPERPND	VMPGPND	VMPGPNT	VMPSTAT	VMPSW	VMRSTAT	VMSEG	VMSHADT
	VMTRCTL	VMTRPER	XRIGHT16										
DMKVCA	ADSPCH	ATTN	BALRSAVE	BALR14	BALR15	BALR2	BALR3	BALR9	BLANKS	BUSY	CC	CD	CE
	CHBATTN	CHBCENT	CHBCNTL	CHBEOFL	CHBHIO	CHBMNOP	CHBM370	CHBRDBK	CHBREAD	CHBREST	CHBSIZE	CHBWAIT	CHBWEOF
	CHBWRIT	CHXBLOK	CHXCMDB	CHXCMDT	CHXCNCT	CHXDATN	CHXFLAG	CHXIDAW	CHXNCCW	CHXCTHR	CHXPKEY	CHXRCNT	CHXSTAT
	CHXYADD	CHYBLOK	CHYCMDB	CHYCMDT	CHYCNCT	CHYDATN	CHYFLAG	CHYIDAW	CHYNCCW	CHYRCNT	CHYSTAT	CHYXADD	CPEX
	CPEXADD	CPEXBLOK	CPEXFPNT	CPEXR0	CPEXR12	CPEXSIZE	CPWAIT	DE	DMKCVTBH	DMKDIASM	DMKDSPCH	DMKFREE	DMKFRET
	DMKQCNWT	DMKSCHDL	DMKSCNVU	DMKSTKCP	DMKSTKIO	DMKSYSRM	DMKTRCSI	DMKVIOIN	FFS	FREESAVE	F1	F2	F240
	IDA	IL	INTREQ	IOBCAW	IOBCC1	IOBCC3	IOBCSW	IOBFLAG	IOBIOER	IOBIRA	IOBLINK	IOBLOK	IOBRES
	IOBRSTRT	IOBSIZE	IOBSTAT	IOBUSER	IOBVADD	IOERBLOK	IOERCCW	IOERCSW	IOERDATA	IOERLEN	IOERSIZE	NORET	PCI
	PCIF	PRGC	PRTC	PSA	RCWADDR	RCWCCW	RCWCNT	RCWCOMND	RCWCTL	RCWFLAG	RCWINVL	RUNUSER	R0
	R1	R10	R11	R12	R13	R14	R15	R2	R3	R4	R5	R6	R7
	R8	R9	SAVEAREA	SAVERETN	SAVER0	SAVER1	SAVER10	SAVER11	SAVER12	SAVER5	SAVEWRK1	SAVEWRK2	SAVEWRK3
	SAVEWRK4	SAVEWRK6	SAVEWRK9	SILI	SKIP	TEMPR0	TEMPR2	TEMPR3	TEMPR4	TEMPR5	TYPCTCA	UC	UE
	VDEVBLOK	VDEVCCW1	VDEVINTS	VDEVIOCT	VDEVNRDY	VDEVREAL	VDEVSTAT	VMBLOK	VMDVSTRT	VMIOWAIT	VMLOGOFF	VMRSTAT	VMRUN
	VMTRCTL	VMTRSIO	VMTTIME	VMUSER	ZEROES								
DMKVCH	ARIOCU	ARIODV	ASYSOP	BRING	CLASDASD	CLASGRAF	CLASSPEC	CLASTAPE	CLASTERM	CLASURI	CLASURO	DEFER	DMKCVTBH
	DMKERMSG	DMKFREE	DMKFRET	DMKPTRAN	DMKQCNWT	DMKSCNRD	DMKSCNRU	DMKSCNVU	DMKVDREL	DMKVDSAT	FFS	NORET	OPERATOR
	PSA	RCHBLOK	RCHCUTBL	RCHDISA	RCHSTAT	RCUBLOK	RCUDISA	RCUDVTBL	RCUSTAT	RDEVATT	RDEVBLOK	RDEVBUSY	RDEVDED
	RDEVDISA	RDEVDRAN	RDEVENAB	RDEVFLAG	RDEVCWN	RDEVRCVY	RDEVRSVD	RDEVSCED	RDEVSPL	RDEVSTAT	RDEVSYS	RDEVTYPC	RDEVTYPE
	R0	R1	R11	R13	R14	R15	R2	R3	R4	R5	R6	R7	R8
	R9	SAVEAREA	SAVER10	SAVER11	SAVER2	SAVEWRK1	SAVEWRK2	SYSTEM	TYP3705	VCHADD	VCHBLOK	VCHCUTBL	VCHDED
	VCHSTAT	VCUADD	VCUBLOK	VCUDVTBL	VDEVADD	VDEVBLOK	VMBLOK	VMCHTBL	VMTTIME	VMUSER			
DMKVCN	ALARM	BLANKS	BRING	BUFCNT	BUFIN	BUFNXT	CC	CD	CE	CLASGRAF	CLASTERM	CMDREJ	CSW
	DE	DEFER	DMKCFMAT	DMKCFMBK	DMKCFMEN	DMKDSPCH	DMKFREE	DMKFRET	DMKPSACC	DMKPSASC	DMKPTRAN	DMKQCNRD	DMKQCNWT
	DMKSCNVU	DMKTBLUP	DMKVIOMK	DMKVMAPS	EDIT	F1	F256	F3	F4	F7	F8	IDA	IL
	INTREQ	NOAUTO	NORET	NOTIME	PCI	PCIF	PRGC	PRIORITY	PRTC	PSA	RDEVBLOK	RDEVTYPC	RDEVTYPE
	R0	R1	R10	R11	R12	R13	R14	R15	R2	R3	R4	R5	R6
	R7	R8	R9	SILI	SKIP	TYPBSC	TYP3210	UC	UE	VCHADD	VCHBLOK	VCHCUINT	VCONADDR
	VCONBFSZ	VCONBUF	VCONCAW	VCONCCW	VCONCNT	VCONCOMD	VCONCTL	VCONFLAG	VCONIDAP	VCONRBSZ	VCONREUF	VCONRCNT	VCONRSV4
	VCONWBSZ	VCONWBUF	VCONWCNT	VCUADD	VCUBLOK	VCUCEPND	VCUDVINT	VCUSHRD	VCUSTAT	VCUTYPE	VDEVADD	VDEVATTN	VDEVAUCR
	VDEVBLOK	VDEVBUSY	VDEVCCW1	VDEVCFLG	VDEVCHAN	VDEVCHBS	VDEVCON	VDEVCSPL	VDEVCSW	VDEVFLAG	VDEVINTS	VDEVIOCT	VDEVKEY
	VDEVNRDY	VDEVPEND	VDEVSFLG	VDEVSNSE	VDEVSTAT	VDEVTERM	VDEVTIC	VDEVTRAN	VDEVTYPC	VDEVTYPE	VDEVVCF	VMBLOK	VMCF
	VMCHSTRT	VMCUSTRT	VMDISC	VMDSTAT	VMDVSTRT	VMESTAT	VMEXTCM	VMEXWAIT	VMGENIO	VMIDLE	VMIOINT	VMIOPND	VMLOGOFF
	VMMLEVEL	VMMLINED	VMMSTMP	VMOSTAT	VMPEND	VMPRIDSP	VMPSW	VMQSTAT	VMRBSC	VMRSTAT	VMTERM	VMTIO	XRIGHT16

```
Module   External References (Labels and Modules)

DMKVDB   ALOCBLOK ALOCCYL1 ALOCCYL2 ALOCMAP  ALOCMAX  ALOCPNT  ALOCUSED ARIODV   ASYSVM   BALR1    BALR14   BALR6    BLANKS
         CC       CLASDASD CLASSPEC CLASTAPE CPEXBLOK CPEXR0   CPEXR13  CPEXSIZE DFRET    DMKCVTBD DMKCVTBH DMKCVTHB DMKDSPCH
         DMKERMSG DMKFREE  DMKFRET  DMKIOSQR DMKLOCKD DMKLOCKQ DMKPGTP0 DMKPGTP4 DMKPGTP5 DMKPGTTM DMKPGTT0 DMKPGTT4 DMKPGTT5
         DMKPGT4P DMKPGT4T DMKPGT5P DMKPGT5T DMKPGT90 DMKQCNWT DMKSCNAU DMKSCNFD DMKSCNRD DMKSCNRN DMKSCNRU DMKSCNVN DMKSCNVS
         DMKSCNVU DMKSYSOC DMKSYSOW DMKVCHDC DMKVDREL DMKVDSAT FFS      FTRRPS   FTR35MB  FTR70MB  F0       F1       F10
         F3       F4       F6       F7       F8       F9       IOBCAW   IOBCC3   IOBCP    IOBFATAL IOBFLAG  IOBIRA   IOBLOK
         IOBMISC  IOBMISC2 IOBSIZE  IOBSPEC  IOBSTAT  IOBTIO   IOBUSER  NORET    OPERATOR OWNDLIST OWNDPREF OWNDRDEV OWNDVSER
         PSA      RDEVADD  RDEVALLN RDEVATT  RDEVBLOK RDEVCODE RDEVDED  RDEVDISA RDEVFLAG RDEVFTR  RDEVLNKS RDEVMDL  RDEVMOUT
         RDEVOWN  RDEVPNT  RDEVPREF RDEVSER  RDEVSTAT RDEVSYS  RDEVTYPC RDEVTYPE RDEVUSER R0       R1       R10      R11
         R12      R13      R14      R15      R2       R3       R4       R5       R6       R7       R8       R9       SAVEAREA
         SAVER11  SAVER2   SAVEWRK1 SAVEWRK2 SAVEWRK3 SAVEWRK4 SAVEWRK5 SAVEWRK6 SAVEWRK9 SILI     SKIP     TYPCTCA  TYP2305
         TYP2314  TYP3330  TYP3340  TYP3350  UDEVADD  UDEVBLOK UDEVMODE UDEVW    VCHADD   VCHBLOK  VCHCUTBL VCHDED   VCHSTAT
         VCUADD   VCUBLOK  VCUDVTBL VDEVADD  VDEVBLOK VDEVCATT VDEVDED  VDEVSTAT VDEVTYPC VMBLOK   VMCHTBL  VMCLASSB VMCLEVEL
         VMMIMSG  VMMLVL2  VMOSTAT  VMSYSOP  VMTTIME  VMUSER   ZEROES

DMKVDR   ASYSVM   CLASDASD CLASSPEC CLASTAPE CLASTERM CLASURI  CLASURO  DMKACODV DMKCFPRD DMKCVTBH DMKFREE  DMKFRET  DMKIOSQR
         DMKIOSRW DMKQCNWT DMKSCNRD DMKSCNRN DMKTDKRL DMKVCARS DMKVSPCO DMKVSPCR FFS      F1       IOBCAW   IOBFLAG  IOBIRA
         IOBLOK   IOBRELCU IOBSIZE  IOBUSER  NORET    OPERATOR PSA      RDEVADD  RDEVATT  RDEVBLOK RDEVDED  RDEVFLAG RDEVLNKS
         RDEVMOUT RDEVSTAT RDEVSYS  RDEVTMAT RDEVTYPC RDEVUSER R0       R1       R10      R11      R13      R15      R2
         R3       R4       R6       R8       SAVEAREA SAVER8   SAVEWRK2 SAVEWRK3 SAVEWRK4 SAVEWRK6 SAVEWRK9 SILI     TYPCTCA
         TYP1052  TYP2305  TYP3211  VCONBFSZ VCONBUF  VCONCTL  VCONRBSZ VCONRBUF VCONSIZE VCONWBSZ VCONWBUF VDEVBLOK VDEVBND
         VDEVCATT VDEVCON  VDEVDED  VDEVEXTN VDEVFCBK VDEVFLAG VDEVLINK VDEVREAL VDEVRELN VDEVSPL  VDEVSTAT VDEVTDSK VDEVTYPC
         VDEVTYPE VFCBSIZE VMBLOK   VMDVSTRT VMUSER   VMVTERM  VSPXBLOK VSPXLEN  ZEROES

DMKVDS   BALR1    BLANKS   CLASDASD CLASGRAF CLASSPEC CLASTAPE CLASTERM CLASURI  CLASURO  DMKCVTBH DMKERMSG DMKFREE  DMKFRET
         DMKSCNRD DMKSCNRU DMKSCNVU DMKSYSCK DMKTDKGT FFS      FTRRSRL  F8       NICSIZE  PSA      RDEVATT  RDEVBLOK RDEVDED
         RDEVDISA RDEVDRAN RDEVENAB RDEVEPLN RDEVFLAG RDEVFTR  RDEVLNCP RDEVLNKS RDEVMAX  RDEVMOUT RDEVNICL RDEVOWN  RDEVRCVY
         RDEVRSVD RDEVSPL  RDEVSTAT RDEVSYS  RDEVTMAT RDEVTYPC RDEVTYPE RDEVUSER R0       R1       R10      R11      R12
         R13      R14      R15      R2       R3       R4       R5       R6       R7       R8       R9       SAVEAREA SAVER1
         SAVER2   SAVER8   SAVEWRK1 SAVEWRK2 SAVEWRK3 SAVEWRK4 SAVEWRK6 SAVEWRK7 SAVEWRK9 TYPBSC   TYPCTCA  TYP1052  TYP2305
         TYP2311  TYP3210  TYP3277  TYP3705  UDEVADD  UDEVBLOK UDEVCLAS UDEVFTR  UDEVMODE UDEVNCYL UDEVRELN UDEVSTAT UDEVTDSK
         UDEVTYPC UDEVTYPE UDEV3158 VCHADD   VCHBLOK  VCHBMX   VCHCUTBL VCHSEL   VCHSIZE  VCHTYPE  VCONSIZE VCUADD   VCUBLOK
         VCUCTCA  VCUDVTBL VCUSHRD  VCUSIZE  VCUTYPE  VDEVADD  VDEVBLOK VDEVBND  VDEVCLAS VDEVCON  VDEVCOPY VDEVCSPL VDEVDED
         VDEVEOF  VDEVFLAG VDEVLINK VDEVNRDY VDEVRDC  VDEVREAL VDEVRELN VDEVRSRL VDEVSFLG VDEVSIZE VDEVSTAT VDEVTDSK VDEVTERM
         VDEVTMAT VDEVTYPC VDEVTYPE VDEVUSER VMBLOK   VMCHCNT  VMCHSTRT VMCHTBL  VMCUCNT  VMCUSTRT VMDVCNT  VMDVSTRT VMFBMX
         VMFSTAT  VMOSTAT  VMSYSOP  VMTERM   VMVTERM  ZEROES

DMKVER   ADSPCH   ALARM    BRING    CLASDASD CPUID    DDRCUA1  DDRCUA2  DDRKEYN  DDRREC   DEFER    DMKCVTBH DMKFREE  DMKFRET
         DMKIOEVR DMKPTRAN DMKQCNWT DMKSCNRD DMKSCNVU DMKVATRN EXTMODE  FTR2311B FTR2311T FTR70MB  F1       F24      F256
         F4       F4095    F7       F8       MDRCUA1  MDRKEYN  MDRREC   MIHCUA1  MIHKEYN  MIHREC   MIHVOL   NORET    OBRCPIDN
         OBRCUA   OBRCUAIN OBRCUAPR OBRHAN   OBRKEYN  OBRLSKN  OBRPGMN  OBRRECN  OBRSENSN OBRSWSN  OBRVOLN  OBR33SNS OPERATOR
         PSA      RDEVBLOK RDEVDED  RDEVFTR  RDEVSER  RDEVSTAT RDEVTYPC RDEVTYPE R0       R1       R10      R11      R13
         R14      R15      R2       R3       R4       R5       R6       R7       R8       R9       SAVEAREA SAVERETN SAVER12
         SAVEWRK1 SAVEWRK2 SAVEWRK3 SAVEWRK4 SAVEWRK5 SAVEWRK7 SAVEWRK9 TRANMODE TYP2305  TYP2314  TYP3330  TYP3340  TYP3350
         VDEVBLOK VDEVDED  VDEVREAL VDEVRELN VDEVSTAT VDEVTYPC VDEVTYPE VMBLOK   VMEXWAIT VMGPRS   VMPSW    VMRSTAT  VMSTOR
         VMUSER
```

Module	External References (Labels and Modules)												
DMKVIO	ATTN	AVMREAL	BLKMPX	BRING	BUSY	CAW	CE	CHBM370	CHXBLOK	CHXFLAG	CLASDASD	CLASGRAF	CLASSPEC
	CLASTERM	CLASURI	CLASURO	CSW	CUE	DE	DEFER	DMKCCWTR	DMKDSPCH	DMKFREE	DMKFRET	DMKIOSQV	DMKPTRAN
	DMKPTRUL	DMKSCHDL	DMKSCNVU	DMKSTKIO	DMKTRCSI	DMKTRCSW	DMKTRCWT	DMKUNTFR	DMKUNTRN	DMKVCASH	DMKVCAST	DMKVCATS	DMKVCNEX
	DMKVSPEX	DMKVSPTO	FTR35MB	FTR70MB	F1	F240	F4095	F8	IL	INTREQ	IOBCAW	IOBCC2	IOBCC3
	IOBCSW	IOBFATAL	IOBFLAG	IOBHIO	IOBICER	IOBIRA	IOBLINK	IOBLOK	IOBMISC	IOBMISC2	IOBRADD	IOBRCAW	IOBRELCU
	IOBSIOF	IOBSIZE	IOBSPEC	IOBSTAT	IOBTIO	IOBUNSL	IOBUSER	IOBVADD	IOBWRAP	IOERBLOK	IOERCSW	IOERDATA	IOEREXT
	IOERSIZE	PCI	PSA	RDEVAIOB	RDEVBLOK	RDEVFTR	RDEVMDL	R0	R1	R10	R11	R12	R13
	R14	R15	R2	R3	R4	R5	R6	R7	R8	R9	SM	TEMPSAVE	TRACBEF
	TRACCURR	TRACEND	TRACFLG2	TRACSTRT	TRACOD	TREXCSW	TREXCTL2	TREXT	TYPCTCA	TYP2314	TYP3210	TYP3330	TYP3340
	UC	UE	VCHADD	VCHBLOK	VCHBMX	VCHBUSY	VCHCEDEV	VCHCEPND	VCHCUINT	VCHCUTBL	VCHSEL	VCHSTAT	VCHTYPE
	VCUACTV	VCUADD	VCUBLOK	VCUBUSY	VCUCEPND	VCUCHBSY	VCUCTCA	VCUCUEPN	VCUDVINT	VCUDVTBL	VCUINTS	VCUSHRD	VCUSTAT
	VCUTYPE	VDEVADD	VDEVBLOK	VDEVBND	VDEVBUSY	VDEVCHAN	VDEVCHBS	VDEVCSW	VDEVCUE	VDEVDED	VDEVDIAL	VDEVENAB	VDEVFLAG
	VDEVINTS	VDEVIOB	VDEVIOER	VDEVNRDY	VDEVPEND	VDEVPOST	VDEVRDO	VDEVREAL	VDEVSAS	VDEVSPL	VDEVSTAT	VDEVTYPC	VDEVTYPE
	VDEVUC	VMACTDEV	VMBLOK	VMCHSTRT	VMCUSTRT	VMDSTAT	VMDVSTRT	VMECEXT	VMESTAT	VMEXTCM	VMEXWAIT	VMGPRS	VMIDLE
	VMINST	VMIOACTV	VMIOCNT	VMIOINT	VMIOPND	VMIOWAIT	VMNOTRAN	VMPEND	VMPRIDSP	VMPSTAT	VMPSW	VMQSTAT	VMRSTAT
	VMSIZE	VMTIO	VMTRBRIN	VMTRCTL	VMTREXT	VMTRIO	VMTRSIO	VMVCR0	VMV370R	XTNDLOCK			
DMKVMA	ACORETBL	ASYSVM	BALRSAVE	BALR2	BRING	CORFLAG	CORPGPNT	CORSHARE	CORSWPNT	CORTABLE	CPEXADD	CPEXBLOK	CPEXR0
	CPEXR2	CPEXR3	CPEXSIZE	DEFER	DMKCFMBK	DMKCVTBH	DMKDSPCH	DMKDSPNP	DMKERMSG	DMKFREE	DMKFRET	DMKPTRAN	DMKPTRSC
	DMKSTKCP	F1	F2	F4095	F4096	F8	LASTUSER	PAGCORE	PAGINVAL	PAGSHR	PAGSWP	PAGTABLE	PSA
	RUNUSER	R0	R1	R10	R11	R12	R13	R14	R15	R2	R3	R4	R5
	R6	R7	R8	R9	SAVEAREA	SAVER11	SAVER2	SAVEWRK1	SAVEWRK2	SAVEWRK3	SAVEWRK5	SAVEWRK6	SAVEWRK7
	SAVEWRK8	SAVEWRK9	SEGPAGE	SEGPLEN	SEGTABLE	SHRBPNT	SHRFPNT	SHRNAME	SHRSEGCT	SHRSEGNM	SHRTABLE	SHRTSIZE	SHRUSECT
	SWPALLOC	SWPFLAG	SWPPAG	SWPRECMP	SWPSHR	SWPTABLE	SWPTRANS	SWPVM	SWPVPAGE	VMABLOK	VMAFPNT	VMANAME	VMASHRBK
	VMASIZE	VMASSIST	VMBLOK	VMIDLE	VMOSTAT	VMPAGES	VMRSTAT	VMSEG	VMSHR	VMSHRSYS	VMTTIME	XPAGNUM	
DMKVMI	ATTN	BUSY	CAW	CC	CD	CE	CLASDASD	CLASSPEC	CLASTAPE	CLASURI	CSW	DE	EXTMODE
	IL	INTTIO	IPLCCW1	IPLPSW	PSA	R0	R1	R10	R11	R12	R13	R14	R15
	R2	R3	R4	R5	R6	R9	SILI	SKIP	SM	TYPCTCA	TYPRDR	TYPUNSUP	TYP2401
	TYP2415	TYP2420	TYP2501	TYP2540R	UC	UE	VMMCODE	VMMTEXT					

Module	External References (Labels and Modules)												
DMKVSP	ADDSFB	ARSPBD	BLANKS	BRING	CC	CD	CE	CHGSFB	CLASURI	CLASURO	CMDREJ	CPEXADD	CPEXBLOK
	CPEXFPNT	CPEXR1	CPEXR11	CPEXR8	CPEXSIZE	CSW	DATACHK	DE	DEFER	DMKBOXHR	DMKCKSPL	DMKCVTBH	DMKCVTDT
	DMKDSPCH	DMKERMSG	DMKFREE	DMKFRET	DMKPGTSG	DMKPGTVG	DMKPGTVR	DMKPSACC	DMKPSASC	DMKPTRAN	DMKPTRUL	DMKRPAGT	DMKRPAPT
	DMKSCNVD	DMKSCNVU	DMKSPLCV	DMKSPLDL	DMKSPLOV	DMKSTKCP	DMKTMRPT	DMKVIOMK	DMKVMAPS	FFS	F0	F1	F4
	F4095	F4096	F8	IDA	IL	INTREQ	LOCK	OPNSFB	PCI	PCIF	PRGC	PRTC	PRTCHN
	PSA	RECBLOK	RECCYL	RECMAP	RECPNT	RECSIZE	RECUSED	R0	R1	R10	R11	R12	R13
	R14	R15	R2	R3	R4	R5	R6	R7	R8	R9	SAVEAREA	SAVER0	SAVER1
	SAVER2	SAVER8	SAVEWRK2	SAVEWRK6	SFBCLAS	SFBDUMP	SFBEOF	SFBFILID	SFBFLAG	SFBFLAG2	SFBHOLD	SFBINUSE	SFBLAST
	SFBLOK	SFBMISC1	SFBNOHLD	SFBOPEN	SFBPNT	SFBRECER	SFBRECNO	SFBRECS	SFBRECSZ	SFBSTART	SFBTIME	SFBTYPE	SFBUHOLD
	SFBUSER	SILI	SKIP	SPLINK	SPNXTPAG	SPPREPAG	SPRECNUM	SPSIZE	SYSTEM	TEMPR0	TEMPR1	TYPPRT	TYPPUN
	TYPTIMER	TYP3210	TYP3211	TYP3505	UC	UE	VCHADD	VCHBLOK	VCHBMX	VCHCEDEV	VCHCEPND	VCHCUINT	VCHCUTBL
	VCHSEL	VCHSTAT	VCHTYPE	VCUADD	VCUBLCK	VCUDVINT	VCUDVTBL	VDEVADD	VDEVBLOK	VDEVBUSY	VDEVCCW1	VDEVCFCL	VDEVCHAN
	VDEVCHES	VDEVCLAS	VDEVCONT	VDEVCSPL	VDEVCSW	VDEVDED	VDEVDIAG	VDEVEOF	VDEVFCBK	VDEVFEED	VDEVFLAG	VDEVHOLD	VDEVINTS
	VDEVIOCT	VDEVKEY	VDEVNRDY	VDEVPEND	VDEVPURG	VDEVSFLG	VDEVSNSE	VDEVSPL	VDEVSTAT	VDEVSVC	VDEVTYPC	VDEVTYPE	VDEVUNIT
	VFCBBLOK	VFCBCHL	VFCBCNT	VFCBEOF	VFCBFLAG	VFCBLOAD	VFCBNDEX	VFCBSIZE	VMBLOK	VMCHSTRT	VMCHTBL	VMCRDS	VMCUSTRT
	VMDVSTRT	VMESTAT	VMEXTCM	VMEXWAIT	VMINST	VMIOINT	VMIOPND	VMLINS	VMOSTAT	VMPEND	VMPNCH	VMPSW	VMRSTAT
	VMSYSOP	VMTTIME	VMUSER	VSPBUFBK	VSPBUFSZ	VSPCAW	VSPCCW	VSPDPAGE	VSPIDACT	VSPIDAL	VSPIDASW	VSPIDAW2	VSPLCTL
	VSPMISC	VSPNEXT	VSPRECNO	VSPSFBLK	VSPSIZE	VSPVPAGE	ZEROES						
DMKWRM	ACNTBLOK	ACNTCCW	ACNTDATA	ACNTNEXT	ACNTSIZE	ADDSFB	ALARM	ARIODV	BRING	CC	CHGSHQ	CKPBLOK	CKPNAME
	CKPRMAX	CKPSIZE	CLASSPEC	CLASTERM	CPID	DEFER	DMKCKSIN	DMKCKSPL	DMKCKSWM	DMKCVTBD	DMKERMSG	DMKFREE	DMKPGTVG
	DMKPGTVR	DMKPTRAN	DMKQCNWT	DMKRPAGT	DMKRPAPT	DMKRSPAC	DMKRSPCV	DMKRSPDL	DMKRSPHQ	DMKRSPID	DMKRSPPR	DMKRSPPU	DMKRSPRD
	DMKSCNRU	DMKSYSDT	DMKSYSLG	DMKSYSOW	DMKSYSWM	FFS	F256	F8	LOCK	NICBLOK	NICDISA	NICENAB	NICFLAG
	NICLGRP	NICSIZE	NICSTAT	NICTERM	NICTYPE	OPERATOR	OWNDLIST	OWNDRDEV	PSA	RDEVALLN	RDEVAUTO	RDEVBLOK	RDEVCKPT
	RDEVCLAS	RDEVCODE	RDEVDISA	RDEVDRAN	RDEVENAB	RDEVFLAG	RDEVMAX	RDEVNCP	RDEVNICL	RDEVRECS	RDEVSEP	RDEVSER	RDEVSPL
	RDEVSTAT	RDEVTYPC	RDEVTYPE	RECBLOK	RECCYL	RECPNT	RECSIZE	R0	R1	R10	R12	R13	R14
	R15	R2	R3	R4	R5	R6	R7	R8	R9	SAVEAREA	SAVER2	SAVEWRK1	SAVEWRK2
	SAVEWRK3	SAVEWRK4	SAVEWRK6	SAVEWRK7	SFBDATE	SFBEOF	SFBFILID	SFBFLAG	SFBFLAG2	SFBINUSE	SFBLOK	SFBOPEN	SFBPNT
	SFBRECER	SFBRECS	SFBRSTRT	SFBSIZE	SHQBLCK	SHQBSIZE	SILI	STARTIME	SYSIPLDV	SYSTEM	TYPBSC	TYP2305	TYP3330
	TYP3340	TYP3350	TYP3705	ZEROES									

Label	Count	References											
ABORT	000002	DMKNLD	DMKRNH										
ACCTACNC	000003	DMKHVD											
ACCTBLOK	000003	DMKHVD	DMKSPL										
ACCTDIST	000002	DMKHVD	DMKSPL										
ACCTLENG	000004	DMKHVD	DMKUSO										
ACCTUSER	000002	DMKHVD	DMKSPL										
ACNTBACK	000007	DMKACO	DMKRSE										
ACNTBLCK	000021	DMKACO	DMKCKP	DMKHVD	DMKRSE	DMKWRM							
ACNTCCW	000009	DMKACO	DMKCKP	DMKWRM									
ACNTCODE	000001	DMKHVD											
ACNTDATA	000014	DMKACO	DMKCKP	DMKHVD	DMKWRM								
ACNTNEXT	000014	DMKACO	DMKCKP	DMKWRM									
ACNTNUM	000001	DMKHVD											
ACNTSIZE	000008	DMKACO	DMKHVD	DMKWRM									
ACNTUSER	000001	DMKHVD											
ACORETBL	000068	DMKACO	DMKBLD	DMKCCW	DMKCDS	DMKCFS	DMKCPI	DMKCPV	DMKCSO	DMKDGD	DMKDMP	DMKEDM	DMKFRE
		DMKMCC	DMKMCH	DMKPAG	DMKPGS	DMKPSA	DMKPTR	DMKRPA	DMKSCH	DMKSPL	DMKUDR	DMKUNT	DMKVMA
ACTSFB	000005	DMKCKS											
ADDSFB	000006	DMKCKS	DMKNLD	DMKSPL	DMKVSP	DMKWRM							
ADSPCH	000005	DMKIOS	DMKQCN	DMKUSO	DMKVCA	DMKVER							
AFREE	000007	DMKFRE											
ALARM	000054	DMKCCH	DMKCKP	DMKCNS	DMKCPI	DMKCQP	DMKDAS	DMKDMP	DMKERM	DMKGRF	DMKMCH	DMKMID	DMKMSG
		DMKMSW	DMKOPR	DMKPAG	DMKPGT	DMKQCN	DMKRGB	DMKRNH	DMKRSP	DMKSAV	DMKUDR	DMKVCN	DMKVER
		DMKWRM											
ALOCBLOK	000012	DMKCPI	DMKMON	DMKPGT	DMKTDK	DMKVDB							
ALOCCYL1	000006	DMKCPI	DMKTDK	DMKVDB									
ALOCCYL2	000005	DMKCPI	DMKTDK	DMKVDB									
ALOCMAP	000012	DMKCPI	DMKPGT	DMKTDK	DMKVDB								
ALOCMAX	000015	DMKCPI	DMKMON	DMKPGT	DMKVDB								
ALOCPNT	000004	DMKCPI	DMKTDK	DMKVDB									
ALOCUSED	000010	DMKCPI	DMKMON	DMKPGT	DMKVDB								
APAGCP	000006	DMKCPI	DMKPSA										
APTRAN	000003	DMKCSO	DMKTRC										
APTRLK	000001	DMKCCW											
ARIOCC	000001	DMKCKP											
ARIOCH	000009	DMKCKP	DMKCPI	DMKCPS	DMKCPV	DMKCQP	DMKIOG	DMKMON	DMKSCN	DMKSSP			
ARIOCT	000010	DMKCKP	DMKCPI	DMKCPS	DMKCPV	DMKCQP	DMKEDM	DMKIOG	DMKMON	DMKSCN	DMKSSP		
ARIOCU	000012	DMKCCH	DMKCKP	DMKCPI	DMKCPS	DMKCPV	DMKCQP	DMKDIA	DMKMON	DMKNES	DMKSCN	DMKSSP	DMKVCH
ARIODC	000002	DMKLOG	DMKSCN										

Label	Count	References											
ARIODV	000045	DMKACO	DMKCCH	DMKCKP	DMKCKS	DMKCPI	DMKCPS	DMKCPV	DMKCQP	DMKCQR	DMKCSO	DMKDIA	DMKDMP
		DMKDRD	DMKGRF	DMKLOG	DMKMON	DMKNES	DMKNET	DMKPAG	DMKPGT	DMKPTR	DMKSCN	DMKSPL	DMKSSP
		DMKUDR	DMKVCH	DMKVDB	DMKWRM								
ARIOPR	000004	DMKCKP	DMKCSO	DMKSPL									
ARIOPU	000008	DMKACO	DMKCKP	DMKCSO	DMKSPL								
ARIORD	000004	DMKCKP	DMKCSO										
ARSPAC	000003	DMKACO											
ARSPPR	000011	DMKCKP	DMKCKS	DMKCQG	DMKCQR	DMKCSP	DMKCSU	DMKEDM	DMKSPL	DMKUSO			
ARSPPU	000009	DMKCKS	DMKCQG	DMKCQR	DMKCSP	DMKCSU	DMKSPL	DMKUSO					
ARSPRD	000025	DMKCKS	DMKCQG	DMKCQR	DMKCSP	DMKCST	DMKCSU	DMKDMP	DMKDRD	DMKNLD	DMKSPL	DMKUSO	DMKVSP
ASYSLC	000022	DMKACO	DMKBLD	DMKCFS	DMKCFT	DMKCKP	DMKLOC	DMKLOG	DMKUDR	DMKUSO			
ASYSOP	000014	DMKCPI	DMKCPS	DMKLOG	DMKMSG	DMKMSW	DMKPSA	DMKQCN	DMKUSO	DMKVCH			
ASYSVM	000098	DMKACO	DMKBLD	DMKCFG	DMKCKP	DMKCNS	DMKCPB	DMKCPI	DMKCPS	DMKCPV	DMKCQP	DMKCSO	DMKDAS
		DMKDIA	DMKDRD	DMKDSP	DMKEDM	DMKFRE	DMKGRF	DMKIOS	DMKLOG	DMKMCC	DMKMCH	DMKMID	DMKMON
		DMKNES	DMKNET	DMKNLD	DMKPGS	DMKPGT	DMKPSA	DMKPTR	DMKRGA	DMKRNH	DMKSCN	DMKSPL	DMKTHI
		DMKUSO	DMKVDB	DMKVDR	DMKVMA								
ATTN	000055	DMKCFC	DMKCFM	DMKCKP	DMKCNS	DMKDDR	DMKDIR	DMKDMP	DMKDSP	DMKFMT	DMKGRF	DMKIOS	DMKRNH
		DMKRSE	DMKSSP	DMKVCA	DMKVIO	DMKVMI							
AVMREAL	000025	DMKBLD	DMKCFG	DMKCFP	DMKCFS	DMKCPV	DMKFRE	DMKMCH	DMKPGS	DMKPTR	DMKRPA	DMKSCH	DMKVIO
BALRSAVE	000077	DMKCCW	DMKCFM	DMKCNS	DMKCPI	DMKCPV	DMKCSO	DMKCVT	DMKDIA	DMKFRE	DMKLOC	DMKPGT	DMKPTR
		DMKQCN	DMKRNH	DMKSCH	DMKSCN	DMKUNT	DMKVAT	DMKVCA	DMKVMA				
BALR0	000005	DMKCPI	DMKPGT	DMKPTR									
BALR1	000021	DMKCPI	DMKCVT	DMKDIA	DMKPGT	DMKSCN	DMKVDB	DMKVDS					
BALR11	000001	DMKSCH											
BALR12	000001	DMKVAT											
BALR13	000001	DMKVAT											
BALR14	000008	DMKCPI	DMKLOC	DMKVAT	DMKVCA	DMKVDB							
BALR15	000001	DMKVCA											
BALR2	000027	DMKCCW	DMKCPI	DMKCVT	DMKDMP	DMKPTR	DMKSCN	DMKTMR	DMKVCA	DMKVMA			
BALR3	000007	DMKCCW	DMKCNS	DMKSCN	DMKVCA								
BALR6	000005	DMKCNS	DMKCPI	DMKVDB									
BALR8	000005	DMKCPI	DMKPGT	DMKSCN									
BALR9	000004	DMKCNS	DMKVCA										
BLANKS	000121	DMKCDS	DMKCFC	DMKCFD	DMKCFM	DMKCFS	DMKCNS	DMKCPB	DMKCQG	DMKCQP	DMKCQR	DMKCSO	DMKCSP
		DMKCSU	DMKDIA	DMKERM	DMKGRF	DMKHVC	DMKLNK	DMKLOG	DMKMCC	DMKMSG	DMKNES	DMKNET	DMKNLD
		DMKQCN	DMKRGA	DMKRNH	DMKRSP	DMKSCN	DMKSPL	DMKTHI	DMKTRC	DMKUDR	DMKUSO	DMKVCA	DMKVCN
		DMKVDB	DMKVDS	DMKVSP									
BLKMPX	000002	DMKCPI	DMKVIO										
BRING	000134	DMKCCW	DMKCDB	DMKCDS	DMKCFD	DMKCFG	DMKCKS	DMKCNS	DMKCPB	DMKCPI	DMKCPV	DMKCSO	DMKCST
		DMKDGD	DMKDRD	DMKDSP	DMKERM	DMKGIO	DMKGRF	DMKHVC	DMKHVD	DMKIOF	DMKIOG	DMKMCC	DMKNLD
		DMKPRG	DMKPRV	DMKPSA	DMKPTR	DMKRGA	DMKRGB	DMKRPA	DMKRSP	DMKSCH	DMKSEP	DMKSNC	DMKSPL
		DMKTMR	DMKTRC	DMKUDR	DMKVAT	DMKVCH	DMKVCN	DMKVER	DMKVIO	DMKVMA	DMKVSP	DMKWRM	
BSCAUSER	000004	DMKRGA	DMKRGB										
BSCBLOK	000005	DMKBSC	DMKRGA	DMKRGB									
BSCCNT	000009	DMKRGA											

Label	Count	References											
BSCCOPY	000009	DMKRGA											
BSCECCW1	000004	DMKRGA											
BSCECCW2	000004	DMKRGA											
BSCENQ	000004	DMKRGA											
BSCETB	000005	DMKRGA											
BSCFLAG	000040	DMKRGA	DMKRGB										
BSCFLAG1	000008	DMKRGA											
BSCIGN	000004	DMKRGA											
BSCINDEX	000006	DMKRGA											
BSCLINE	000002	DMKRGB											
BSCLOG	000005	DMKRGA											
BSCOPIED	000004	DMKRGA											
BSCPCCW1	000004	DMKRGA	DMKRGB										
BSCPCCW2	000002	DMKRGA	DMKRGB										
BSCPCCW3	000003	DMKRGA											
BSCPCCW4	000006	DMKRGA	DMKRGB										
BSCRCVD	000008	DMKRGA	DMKRGB										
BSCREAD	000024	DMKBSC	DMKRGA	DMKRGB									
BSCREGEN	000003	DMKRGA											
BSCRESP	000028	DMKBSC	DMKRGA	DMKRGB									
BSCRROBN	000006	DMKRGA	DMKRGB										
BSCRSTRT	000002	DMKRGA											
BSCRVI	000003	DMKRGA											
BSCSCAN	000007	DMKRGA	DMKRGB										
BSCSCCW1	000006	DMKRGA	DMKRGB										
BSCSCCW2	000005	DMKRGA	DMKRGB										
BSCSCCW3	000003	DMKRGA	DMKRGB										
BSCSEL	000012	DMKRGA	DMKRGB										
BSCSEND	000005	DMKRGA											
BSCSENSE	000015	DMKRGA											
BSCSIZE	000002	DMKRGA	DMKRGB										
BSCSIZE1	000004	DMKRGA	DMKRGB										
BSCSIZE2	000001	DMKRGB											
BSCSPTR	000015	DMKRGA	DMKRGB										
BSCTMRQ	000006	DMKRGA											
BSCTSTRQ	000002	DMKRGA											
BSCUCOPY	000006	DMKRGA											
BSCUECCW	000003	DMKRGA											
BUFCNT	000027	DMKCFM	DMKCFS	DMKCPI	DMKCST	DMKERM	DMKGRF	DMKLOG	DMKRGA	DMKRSP	DMKVCN		
BUFFER	000107	DMKCDB	DMKCFG	DMKCFM	DMKCFS	DMKCPI	DMKCSO	DMKCSP	DMKCST	DMKCSU	DMKERM	DMKGRF	DMKLNK
		DMKLOG	DMKMSG	DMKRGA	DMKRSP	DMKSCN							
BUFIN	000003	DMKCPI	DMKVCN										

Label	Count	References
BUFINLTH	000018	DMKCFM DMKERM DMKGRF DMKLNK DMKRGA DMKRGB
BUFNXT	000026	DMKCDB DMKCFG DMKCFM DMKCFS DMKCPI DMKCSO DMKCSU DMKLNK DMKLOG DMKMSG DMKRSP DMKSCN DMKVCN
BUFSIZE	000026	DMKCFM DMKCPI DMKERM DMKGRF DMKLNK DMKLOG DMKRGA DMKRSP
BUSOUT	000005	DMKRNH DMKRSE
BUSY	000044	DMKCKP DMKCNS DMKCPI DMKDDR DMKDIR DMKDMP DMKFMT DMKIOS DMKPSA DMKRNH DMKSSP DMKVCA DMKVIO DMKVMI
CACTDEV	000002	DMKRNH
CACTLIN	000002	DMKNET DMKRNH
CACTLTR	000002	DMKNES DMKRNH
CAW	000073	DMKCCH DMKCKP DMKCNS DMKCPI DMKDMP DMKFMT DMKIOS DMKOPR DMKSAV DMKSSP DMKTRC DMKVIO DMKVMI
CC	000845	DMKACO DMKBSC DMKCCW DMKCKP DMKCNS DMKCPI DMKCSO DMKDAS DMKDDR DMKDGD DMKDIA DMKDIR DMKDMP DMKFMT DMKGRF DMKIOS DMKMCC DMKMON DMKNLD DMKOPR DMKPAG DMKRGA DMKRGB DMKRNH DMKRSE DMKRSP DMKSAV DMKSEP DMKSPL DMKSSP DMKTDK DMKUCB DMKUCS DMKUDR DMKVCA DMKVCN DMKVDB DMKVMI DMKVSP DMKWRM
CCC	000041	DMKBSC DMKCCH DMKCNS DMKCPI DMKDAS DMKEIG DMKGRF DMKHVC DMKIOE DMKIOS DMKMSW DMKRSE DMKRSP DMKSEV DMKTAP DMKUNT
CCCPUID	000001	DMKCCH
CCDESMD	000003	DMKDIA DMKRNH
CCDEVTYP	000001	DMKCCH
CCHADDR	000001	DMKCCH
CCHANID	000006	DMKCCH
CCHCAV	000001	DMKCCH
CCHCHNL	000012	DMKSEV DMKSIX
CCHCMDV	000010	DMKEIG DMKSEV DMKSIX
CCHCNTB	000005	DMKSEV DMKSIX
CCHCPU	000003	DMKSEV DMKSIX
CCHCUA	000002	DMKCCH
CCHDAV	000010	DMKEIG DMKSEV DMKSIX
CCHDI	000003	DMKEIG DMKSEV DMKSIX
CCHHIO	000002	DMKCCH
CCHINTB	000001	DMKCCH
CCHINTFC	000007	DMKSEV DMKSIX
CCHLOG45	000002	DMKCCH
CCHLOG60	000001	DMKSIX
CCHLOG70	000001	DMKSEV
CCHLOG80	000002	DMKCCH DMKEIG
CCHRCV	000003	DMKCCH DMKEIG
CCHREC	000005	DMKCCH DMKEIG DMKSEV DMKSIX
CCHSIOB	000002	DMKCCH
CCHSIZE	000002	DMKCCH

Label	Count	References											
CCHSIZE1	000002	DMKCCH											
CCHSNSB	000001	DMKCCH											
CCHSTG	000004	DMKSEV	DMKSIX										
CCHTIO	000002	DMKCCH											
CCHUSV	000005	DMKEIG	DMKSEV	DMKSIX									
CCPADDR	000001	DMKSNC											
CCPARM	000004	DMKNLD	DMKSNC										
CCPENTRY	000001	DMKNLD											
CCPMAXID	000001	DMKNLD											
CCPNAME	000003	DMKNLD	DMKSNC										
CCPPSIZE	000005	DMKNLD	DMKSNC										
CCPRESID	000002	DMKNLD											
CCPROGID	000003	DMKCCH											
CCPRSTAT	000001	DMKNLD											
CCPRSTEP	000001	DMKNLD											
CCPRSTYP	000001	DMKNLD											
CCPSIZE	000003	DMKNLD	DMKSNC										
CCPTEP	000002	DMKNLD											
CCPTPEP	000001	DMKNLD											
CCPTYPE	000003	DMKNLD											
CCRECTYP	000002	DMKCCH											
CD	000099	DMKCCW	DMKCNS	DMKDAS	DMKDDR	DMKDGD	DMKDIA	DMKDIR	DMKFMT	DMKGRF	DMKISM	DMKOPR	DMKRGA
		DMKRGB	DMKSSP	DMKTAP	DMKUNT	DMKVCA	DMKVCN	DMKVMI	DMKVSP				
CDC	000038	DMKBSC	DMKCCH	DMKCNS	DMKDAS	DMKGRF	DMKHVC	DMKIOE	DMKIOF	DMKIOS	DMKMSW	DMKNLD	DMKRNH
		DMKRSE	DMKRSP	DMKTAP	DMKUNT								
CDCTLIN	000001	DMKNET											
CDISPLY	000001	DMKNES											
CE	000077	DMKCKP	DMKCNS	DMKCPI	DMKDDR	DMKDIA	DMKDIR	DMKDMP	DMKFMT	DMKGRF	DMKHVC	DMKIOS	DMKRGA
		DMKRSE	DMKRSP	DMKSAV	DMKSSP	DMKVCA	DMKVCN	DMKVIO	DMKVMI	DMKVSP			
CFSTOP	000006	DMKCPS	DMKMCC	DMKMON									
CHANID	000003	DMKIOG	DMKPRV										
CHBATTN	000013	DMKVCA											
CHBCENT	000003	DMKVCA											
CHBCNTL	000002	DMKVCA											
CHBEOFL	000014	DMKVCA											
CHBHIO	000015	DMKVCA											
CHBMNOP	000005	DMKVCA											
CHBM370	000020	DMKVCA	DMKVIO										
CHBRDBK	000007	DMKVCA											
CHBREAD	000008	DMKVCA											
CHBREST	000012	DMKVCA											
CHBSIZE	000003	DMKDIA	DMKVCA										

Label	Count	References
CHBWAIT	000014	DMKCFP DMKVCA
CHBWEOF	000002	DMKVCA
CHBWRIT	000009	DMKVCA
CHC	000014	DMKBSC DMKCNS DMKGRF DMKHVC DMKIOS DMKRNH DMKRSE DMKTAP DMKUNT
CHGRDV	000002	DMKCSO
CHGSFB	000012	DMKCKS DMKCSP DMKCSU DMKDMP DMKRSP DMKSPL DMKVSP
CHGSHQ	000004	DMKCSP DMKWRM
CHXBLOK	000013	DMKCFP DMKCQG DMKDIA DMKVCA DMKVIO
CHXCMDB	000010	DMKVCA
CHXCMDT	000014	DMKVCA
CHXCNCT	000009	DMKCFP DMKVCA
CHXDATN	000005	DMKVCA
CHXFLAG	000057	DMKCFP DMKVCA DMKVIO
CHXIDAW	000004	DMKVCA
CHXNCCW	000012	DMKVCA
CHXOTHR	000009	DMKCQG DMKDIA DMKVCA
CHXPKEY	000005	DMKVCA
CHXRCNT	000010	DMKVCA
CHXSTAT	000020	DMKVCA
CHXYADD	000007	DMKCQG DMKDIA DMKVCA
CHYBLOK	000005	DMKDIA DMKVCA
CHYCMDB	000001	DMKVCA
CHYCMDT	000003	DMKVCA
CHYCNCT	000004	DMKVCA
CHYDATN	000006	DMKVCA
CHYFLAG	000032	DMKVCA
CHYIDAW	000001	DMKVCA
CHYNCCW	000004	DMKVCA
CHYOTHR	000001	DMKDIA
CHYRCNT	000005	DMKVCA
CHYSTAT	000003	DMKVCA
CHYXADD	000005	DMKDIA DMKVCA
CKCMASK	000001	DMKCPI
CKPBITS	000003	DMKRNH
CKPBKSZ	000001	DMKRNH
CKPBLOK	000004	DMKRNH DMKWRM
CKPNAME	000003	DMKRNH DMKWRM
CKPRMAX	000002	DMKRNH DMKWRM
CKPSIZE	000003	DMKRNH DMKWRM
CLASDASD	000108	DMKACO DMKCCW DMKCKP DMKCPI DMKCPS DMKCPV DMKCQG DMKCQP DMKCQR DMKDDR DMKDEF DMKDGD DMKDIR DMKDMP DMKEDM DMKGIC DMKIOC DMKIOE DMKIOF DMKIOS DMKLNK DMKLOG DMKMON DMKMSW DMKSCN DMKSSP DMKTRC DMKVCH DMKVDB DMKVDR DMKVDS DMKVER DMKVIO DMKVMI

Label	Count	References											
CLASGRAF	000060	DMKCCW	DMKCFM	DMKCFP	DMKCFT	DMKCKP	DMKCPI	DMKCPS	DMKCPV	DMKCQG	DMKCQP	DMKDEF	DMKDIA
		DMKDIR	DMKEDM	DMKGRF	DMKHVC	DMKHVD	DMKIOE	DMKIOF	DMKIOS	DMKOPR	DMKPSA	DMKQCN	DMKSSP
		DMKTRC	DMKUSO	DMKVCH	DMKVCN	DMKVDS	DMKVIO						
CLASSPEC	000061	DMKBLD	DMKCCW	DMKCFP	DMKCFT	DMKCKP	DMKCPB	DMKCQG	DMKCQP	DMKCQR	DMKDEF	DMKDIA	DMKDIR
		DMKHVD	DMKIOE	DMKIOF	DMKIOS	DMKLOG	DMKNES	DMKNET	DMKNLD	DMKQCN	DMKRNH	DMKSCN	DMKTRC
		DMKUSO	DMKVCH	DMKVDB	DMKVDR	DMKVDS	DMKVIO	DMKVMI	DMKWRM				
CLASTAPE	000064	DMKCCW	DMKCFS	DMKCKP	DMKCPB	DMKCPI	DMKCPS	DMKCQG	DMKCQP	DMKCQR	DMKDDR	DMKDMP	
		DMKGIO	DMKIOE	DMKIOF	DMKIOS	DMKMCC	DMKMON	DMKSSP	DMKVCH	DMKVDB	DMKVDR	DMKVDS	DMKVMI
CLASTERM	000138	DMKBLD	DMKCCW	DMKCFM	DMKCFP	DMKCFT	DMKCKP	DMKCNS	DMKCPB	DMKCPI	DMKCPS	DMKCPV	DMKCQG
		DMKCQP	DMKCQR	DMKCSP	DMKCST	DMKDDR	DMKDEF	DMKDIA	DMKDIR	DMKEDM	DMKHVC	DMKHVD	DMKIOC
		DMKIOE	DMKIOF	DMKIOS	DMKLOG	DMKNES	DMKNET	DMKPSA	DMKQCN	DMKRGA	DMKSCN	DMKSSP	DMKTRC
		DMKUSO	DMKVCH	DMKVCN	DMKVDR	DMKVDS	DMKVIO	DMKWRM					
CLASURI	000076	DMKCCW	DMKCFP	DMKCKP	DMKCPB	DMKCPS	DMKCQG	DMKCQP	DMKCSO	DMKCSP	DMKCST	DMKCSU	
		DMKDEF	DMKDIR	DMKDRD	DMKEDM	DMKHVD	DMKIOF	DMKIOS	DMKRSE	DMKRSP	DMKSCN	DMKSPL	DMKSSP
		DMKTRC	DMKVCH	DMKVDR	DMKVDS	DMKVIO	DMKVMI	DMKVSP					
CLASURO	000058	DMKCCW	DMKCFP	DMKCFS	DMKCKP	DMKCPB	DMKCPS	DMKCQG	DMKCQP	DMKCSO	DMKCSP	DMKCST	
		DMKDEF	DMKDIR	DMKDMP	DMKEDM	DMKHVD	DMKIOF	DMKIOS	DMKRSE	DMKRSP	DMKSCN	DMKSSP	DMKTRC
		DMKVCH	DMKVDR	DMKVDS	DMKVIC	DMKVSP							
CMDREJ	000010	DMKCNS	DMKDIA	DMKRNH	DMKRSE	DMKVCN	DMKVSP						
CNTLBTU	000005	DMKRNH											
COMPFES	000009	DMKEIG	DMKSEV	DMKSIX									
COMPSEL	000015	DMKEIG	DMKSEV	DMKSIX									
COMPSYS	000006	DMKCCH	DMKEIG	DMKSEV									
CONACTV	000028	DMKCNS	DMKGRF	DMKRGA	DMKRNH								
CONADDR	000032	DMKCNS	DMKGRF	DMKMON	DMKQCN	DMKRGA	DMKRGB	DMKRNH					
CONCCW1	000083	DMKCNS	DMKGRF	DMKRGA	DMKRGB	DMKRNH							
CONCCW2	000037	DMKCNS	DMKGRF	DMKRGA	DMKRGB	DMKRNH							
CONCCW3	000034	DMKCNS	DMKDIA	DMKIOE	DMKNES	DMKNET	DMKRGA	DMKRGB	DMKRNH				
CONCCW4	000030	DMKCNS	DMKGRF	DMKRGA	DMKRGB								
CONCNT	000069	DMKCNS	DMKGRF	DMKMON	DMKQCN	DMKRGA	DMKRGB	DMKRNH					
CONCNTL	000024	DMKCNS	DMKQCN	DMKRGA	DMKRGB	DMKRNH							
CONCOMND	000008	DMKCNS	DMKRNH										
CONDATA	000081	DMKCNS	DMKDIA	DMKGRF	DMKIOE	DMKNES	DMKQCN	DMKRGA	DMKRGB	DMKRNH			
CONDCNT	000031	DMKDIA	DMKIOE	DMKRGA	DMKRNH								
CONDEST	000004	DMKRNH											
CONESCP	000021	DMKCNS	DMKGRF	DMKRGA	DMKRGB	DMKRNH							
CONEXTR	000001	DMKRNH											
CONFLAG	000006	DMKCNS	DMKRNH										
CONLABEL	000029	DMKRGA	DMKRGB										
CONOUTPT	000031	DMKCNS	DMKGRF	DMKQCN	DMKRGB	DMKRNH							
CONPARM	000109	DMKCNS	DMKGRF	DMKQCN	DMKRGA	DMKRGB	DMKRNH						
CONPNT	000091	DMKCNS	DMKEDM	DMKGRF	DMKQCN	DMKRGA	DMKRGB	DMKRNH					

Label	Count	References											
CONRESP	000017	DMKCNS	DMKGRF	DMKQCN	DMKRGA	DMKRGB	DMKRNH						
CONRETN	000030	DMKCNS	DMKGRF	DMKQCN	DMKRGA	DMKRGB	DMKRNH						
CONRSV3	000022	DMKGRF	DMKQCN	DMKRGB									
CONRTAG	000003	DMKRNH											
CONRTRY	000012	DMKCNS	DMKRNH										
CONSPLT	000014	DMKCNS	DMKQCN	DMKRNH									
CONSRID	000013	DMKRNH											
CONSTAT	000126	DMKCNS	DMKGRF	DMKQCN	DMKRGA	DMKRGB	DMKRNH						
CONSYNC	000005	DMKCNS	DMKGRF	DMKQCN	DMKRGB	DMKRNH							
CONSYSR	000040	DMKDIA	DMKNES	DMKNET	DMKRNH								
CONTACT	000003	DMKNET	DMKRNH										
CONTASK	000119	DMKCNS	DMKEDM	DMKGRF	DMKMON	DMKNES	DMKQCN	DMKRGA	DMKRGB	DMKRNH			
CONTCMD	000027	DMKRNH											
CONTSIZE	000036	DMKCNS	DMKGRF	DMKQCN	DMKRGA	DMKRGB	DMKRNH						
CONTSKSZ	000016	DMKCNS	DMKEDM	DMKGRF	DMKQCN	DMKRGA	DMKRGB	DMKRNH					
CONUSER	000012	DMKCNS	DMKQCN	DMKRGA	DMKRGB	DMKRNH							
CORBPNT	000020	DMKMCH	DMKPGS	DMKPTR	DMKRPA	DMKSCH	DMKUDR						
CORCFLCK	000017	DMKBLD	DMKCPI	DMKCPV	DMKPGS	DMKPTR	DMKRPA	DMKSCH					
CORCP	000010	DMKCPI	DMKDMP	DMKEDM	DMKMCC	DMKMON	DMKPTR						
CORDISA	000002	DMKEDM	DMKMCH										
CORFLAG	000056	DMKBLD	DMKCCW	DMKCDS	DMKCFS	DMKCPI	DMKCPV	DMKDMP	DMKEDM	DMKMCC	DMKMCH	DMKMON	DMKPGS
		DMKPSA	DMKPTR	DMKRPA	DMKSCH	DMKVMA							
CORFLUSH	000001	DMKEDM											
CORFPNT	000052	DMKBLD	DMKCPI	DMKCPV	DMKDMP	DMKEDM	DMKMCH	DMKMON	DMKPGS	DMKPTR	DMKRPA	DMKSCH	DMKUDR
CORFREE	000003	DMKCPI	DMKEDM	DMKPTR									
CORIOLCK	000012	DMKMCH	DMKPGS	DMKPTR	DMKRPA	DMKSCH							
CORLCNT	000008	DMKBLD	DMKPTR										
CORPGPNT	000034	DMKBLD	DMKCDS	DMKDGD	DMKFRE	DMKMCH	DMKPGS	DMKPTR	DMKRPA	DMKUDR	DMKUNT	DMKVMA	
CORRSV	000008	DMKCFS	DMKPGS	DMKPTR	DMKSCH								
CORSHARE	000014	DMKCCW	DMKCDS	DMKEDM	DMKPGS	DMKPSA	DMKPTR	DMKSCH	DMKVMA				
CORSWPNT	000019	DMKBLD	DMKCCW	DMKCDS	DMKCPI	DMKDGD	DMKEDM	DMKMCH	DMKPTR	DMKRPA	DMKVMA		
CORTABLE	000087	DMKBLD	DMKCCW	DMKCDS	DMKCFS	DMKCPI	DMKCPV	DMKDGD	DMKDMP	DMKEDM	DMKFRE	DMKMCC	DMKMCH
		DMKMON	DMKPAG	DMKPGS	DMKPSA	DMKPTR	DMKRPA	DMKSCH	DMKUDR	DMKUNT	DMKVMA		
CPABEND	000009	DMKDMP	DMKEDM	DMKPRG	DMKPSA								
CPCREG0	000018	DMKCKP	DMKCPI	DMKDSP	DMKIOS	DMKMCH	DMKPRG	DMKPRV	DMKPSA	DMKTMR	DMKTRC	DMKVAT	
CPCREG8	000019	DMKCPS	DMKIOS	DMKMCC	DMKMCN	DMKPRG	DMKPSA						
CPEX	000009	DMKDSP	DMKVCA										
CPEXADD	000045	DMKACO	DMKCDS	DMKCFM	DMKCPV	DMKDIA	DMKDSP	DMKGRF	DMKIOE	DMKIOS	DMKLOC	DMKMCH	DMKMON
		DMKPAG	DMKPGT	DMKPTR	DMKQCN	DMKRGA	DMKRGB	DMKRNH	DMKRPA	DMKRSP	DMKSPL	DMKUSO	DMKVCA
		DMKVMA	DMKVSP										

Label	Count	References
CPEXBLCK	000088	DMKACO DMKCDS DMKCFM DMKCPS DMKCPV DMKDIA DMKDSP DMKGRF DMKIOE DMKIOF DMKIOS DMKLOC
		DMKMCH DMKMON DMKPAG DMKPGT DMKPTR DMKQCN DMKRGA DMKRGB DMKRNH DMKRPA DMKRSP DMKSPL
		DMKSTK DMKUSO DMKVCA DMKVDB DMKVMA DMKVSP
CPEXBPNT	000006	DMKDSP DMKPAG DMKSTK
CPEXFPNT	000033	DMKCDS DMKDSP DMKIOE DMKIOF DMKLOC DMKPAG DMKPTR DMKRPA DMKSTK DMKVCA DMKVSP
CPEXMISC	000006	DMKPAG DMKPTR
CPEXREGS	000009	DMKCFM DMKCPV DMKDSP DMKIOE DMKIOF DMKLOC DMKQCN DMKSPL
CPEXR0	000025	DMKCDS DMKCPS DMKGRF DMKMON DMKPAG DMKPTR DMKRGA DMKRPA DMKUSO DMKVCA DMKVDB DMKVMA
CPEXR1	000001	DMKVSP
CPEXR10	000001	DMKDSP
CPEXR11	000005	DMKDSP DMKPAG DMKUSO DMKVSP
CPEXR12	000004	DMKCPV DMKQCN DMKUSO DMKVCA
CPEXR13	000004	DMKIOS DMKPTR DMKVDB
CPEXR14	000001	DMKCDS
CPEXR15	000001	DMKCDS
CPEXR2	000005	DMKPTR DMKVMA
CPEXR3	000001	DMKVMA
CPEXR5	000003	DMKCDS DMKPAG
CPEXR6	000005	DMKIOF
CPEXR7	000003	DMKCDS DMKPAG DMKPTR
CPEXR8	000001	DMKVSP
CPEXR9	000002	DMKPTR
CPEXSIZE	000057	DMKACO DMKCDS DMKCFM DMKCPI DMKCPS DMKCPV DMKDIA DMKDSP DMKGRF DMKIOE DMKIOF DMKIOG
		DMKIOS DMKLOC DMKMCC DMKMCH DMKMON DMKPGT DMKPTR DMKQCN DMKRGA DMKRGB DMKRNH DMKRPA
		DMKRSP DMKSPL DMKUSO DMKVCA DMKVDB DMKVMA DMKVSP
CPID	000031	DMKCCH DMKCKP DMKCNS DMKCPI DMKCPS DMKCVT DMKDMP DMKGRF DMKMCH DMKPGT DMKWRM
CPMICAVL	000007	DMKCFS DMKCPI DMKLOG
CPMICON	000009	DMKCFS DMKCPI DMKCQR DMKDSP
CPRUN	000002	DMKDSP
CPSHRLK	000007	DMKCCW DMKDGD DMKDSP
CPSTAT	000001	DMKPTR
CPSTATUS	000012	DMKCPI DMKDSP DMKIOS
CPSTAT2	000019	DMKCCW DMKCFS DMKCPI DMKCQR DMKDGD DMKDSP DMKLOG
CPUID	000028	DMKCCH DMKCPI DMKHVC DMKIOE DMKIOF DMKIOG DMKMCH DMKMON DMKOPR DMKPRV DMKSSP DMKVER
CPULOG	000001	DMKCPI
CPUMCELL	000003	DMKHVD DMKIOG DMKPRV
CPUMODEL	000002	DMKCPI DMKIOG
CPUVERSN	000011	DMKCPI DMKHVD DMKIOF DMKIOG DMKMCH DMKOPR DMKPRV DMKSSP
CPWAIT	000008	DMKCPI DMKDSP DMKIOS DMKVCA
CRESCND	000001	DMKRNH
CRESDQ	000001	DMKDIA
CRESERL	000002	DMKRNH

Label	Count	References											
CRESIMD	000005	DMKDIA	DMKNET	DMKPSA	DMKRNH								
CSETDSM	000002	DMKDIA	DMKRNH										
CSW	000269	DMKCCH	DMKCKP	DMKCNS	DMKCPI	DMKDGD	DMKDMP	DMKDSP	DMKEIG	DMKFMT	DMKGIO	DMKIOS	DMKOPR
		DMKPSA	DMKSAV	DMKSEV	DMKSIX	DMKSSP	DMKTRA	DMKTRC	DMKVCN	DMKVIO	DMKVMI	DMKVSP	
CSWLMEP	000002	DMKDIA	DMKNES										
CSWLNCP	000002	DMKDIA	DMKNES										
CTRMLTR	000003	DMKDIA	DMKNES	DMKRNH									
CUE	000039	DMKCFP	DMKCKP	DMKCPI	DMKDDR	DMKDIR	DMKDMP	DMKDSP	DMKFMT	DMKIOS	DMKMON	DMKNLD	DMKPSA
		DMKRSE	DMKSSP	DMKTAP	DMKVIC								
C0	000034	DMKCKP	DMKCPI	DMKDMP	DMKDSP	DMKIOS	DMKMCH	DMKPRG	DMKPRV	DMKPSA	DMKTMR	DMKTRC	
C1	000049	DMKCCW	DMKCDB	DMKCPI	DMKCSO	DMKDSP	DMKPRV	DMKPSA	DMKPTR	DMKTMR	DMKTRA	DMKTRC	DMKVAT
C11	000001	DMKDSP											
C13	000003	DMKDSP	DMKMCH										
C14	000003	DMKCFG	DMKCPI	DMKDMP									
C15	000003	DMKCFG	DMKDMP										
C2	000013	DMKCFG	DMKCKP	DMKDMP	DMKFRE								
C3	000003	DMKCKP	DMKMCH										
C4	000003	DMKDSP											
C5	000002	DMKDSP											
C6	000009	DMKCPI	DMKDSP										
C7	000006	DMKCCH	DMKDSP	DMKMCH									
C8	000022	DMKCPS	DMKIOS	DMKMCC	DMKMON	DMKPRG	DMKPSA						
C9	000001	DMKDSP											
DAMAGRPT	000002	DMKCPI	DMKDMP										
DASDCL	000006	DMKMCC	DMKMON										
DATACHK	000006	DMKCNS	DMKRSE	DMKVSP									
DATE	000027	DMKCPI	DMKCVT	DMKDMP	DMKEDM	DMKMID							
DDRCUA1	000002	DMKVER											
DDRCUA2	000002	DMKVER											
DDRKEYN	000001	DMKVER											
DDRREC	000001	DMKVER											
DE	000096	DMKACO	DMKCKP	DMKCNS	DMKCPB	DMKCPI	DMKCPS	DMKCSO	DMKCSP	DMKCSU	DMKDDR	DMKDIA	DMKDIR
		DMKDMP	DMKFMT	DMKGRF	DMKHVC	DMKIOS	DMKMON	DMKRGA	DMKRNH	DMKRSE	DMKRSP	DMKSAV	DMKSPL
		DMKSSP	DMKTAP	DMKVCA	DMKVCN	DMKVIO	DMKVMI	DMKVSP					
DEFER	000112	DMKCCW	DMKCDB	DMKCDS	DMKCFD	DMKCFG	DMKCKS	DMKCNS	DMKCPB	DMKCPI	DMKCPV	DMKCSO	DMKDGD
		DMKDRD	DMKDSP	DMKERM	DMKGIO	DMKGRF	DMKHVC	DMKHVD	DMKIOF	DMKIOG	DMKMCC	DMKNLD	DMKPGS
		DMKPRG	DMKPRV	DMKPSA	DMKPTR	DMKRGA	DMKRGB	DMKRPA	DMKRSP	DMKSCH	DMKSEP	DMKSNC	DMKSPL
		DMKTMR	DMKTRC	DMKUDR	DMKVAT	DMKVCH	DMKVCN	DMKVER	DMKVIO	DMKVMA	DMKVSP	DMKWRM	
DELPAGES	000007	DMKBLD	DMKCFP	DMKDEF	DMKPGS	DMKUSO							
DELSEGS	000005	DMKBLD	DMKDEF	DMKUSO									
DELSFB	000005	DMKCKS	DMKCSP	DMKSPL									
DEVCARD	000002	DMKACO	DMKCKP										

Label	Count	References										
DEVCCH	000005	DMKCCH										
DFRET	000016	DMKACO	DMKCPS	DMKCQR	DMKDAS	DMKDIA	DMKERM	DMKPSA	DMKQCN	DMKRNH	DMKTHI	DMKVDB
DISCEOC	000002	DMKRNH										
DISCNCT	000001	DMKRNH										
DMKACO	000001	DMKSYM										
DMKACODV	000004	DMKCPV	DMKDIA	DMKVDR								
DMKACOFF	000003	DMKCPV	DMKUSO									
DMKACON	000001	DMKLOG										
DMKACOPU	000001	DMKRSP										
DMKACOQU	000001	DMKHVD										
DMKACOTM	000003	DMKCPV	DMKCQR	DMKUSO								
DMKBLDEC	000002	DMKCFS	DMKLOG									
DMKBLDRL	000005	DMKBLD	DMKCFP	DMKDEF	DMKPGS	DMKSYM	DMKUSO					
DMKBLDRT	000008	DMKCFG	DMKCFP	DMKCPI	DMKDEF	DMKLOG	DMKPGS	DMKSYM				
DMKBLDVM	000007	DMKCNS	DMKDIA	DMKGRF	DMKLOG	DMKRGA	DMKRNH	DMKSYM				
DMKBOXBX	000003	DMKGRF	DMKRGB	DMKSEP								
DMKBOXHR	000001	DMKVSP										
DMKBSCER	000002	DMKIOS	DMKSYM									
DMKCCHCF	000001	DMKIOG										
DMKCCHIS	000002	DMKIOS	DMKSYM									
DMKCCHMX	000001	DMKIOG										
DMKCCHNT	000001	DMKIOS										
DMKCCHRT	000002	DMKIOE	DMKSYM									
DMKCCHSZ	000001	DMKIOG										
DMKCCH60	000002	DMKIOG	DMKSYM									
DMKCCWSB	000002	DMKSYM	DMKTRC									
DMKCCWTC	000003	DMKHVC	DMKSYM									
DMKCCWTR	000004	DMKGIO	DMKHVC	DMKSYM	DMKVIO							
DMKCDBDC	000002	DMKCFC	DMKSYM									
DMKCDBDI	000002	DMKCFC	DMKSYM									
DMKCDBDM	000002	DMKCFC	DMKSYM									
DMKCDBDU	000002	DMKCFC	DMKSYM									
DMKCDSCP	000002	DMKCFC	DMKSYM									
DMKCDSTO	000002	DMKCFC	DMKSYM									
DMKCFCMD	000001	DMKCFM										
DMKCFDAD	000002	DMKCFC	DMKSYM									
DMKCFDLO	000002	DMKCFC	DMKSYM									
DMKCFGCL	000001	DMKHVC										
DMKCFGII	000002	DMKLOG	DMKSYM									
DMKCFGIP	000002	DMKCFC	DMKSYM									
DMKCFGSV	000002	DMKCFC	DMKSYM									
DMKCFMAT	000010	DMKCFC	DMKCNS	DMKGRF	DMKRGA	DMKRNH	DMKSYM	DMKVCN				

Label	Count	References
DMKCPMBK	000018	DMKCNS DMKCPV DMKDSP DMKGRF DMKHVC DMKIOE DMKMCH DMKPRG DMKPSA DMKPTR DMKRGA DMKRNH DMKSYM DMKTRC DMKVCN DMKVMA
DMKCPMEN	000006	DMKCPI DMKGRF DMKHVC DMKRGA DMKSYM DMKVCN
DMKCPMWU	000001	DMKCFC
DMKCPPRD	000006	DMKCPB DMKCPV DMKDEF DMKDIA DMKSYM DMKVDR
DMKCPPRI	000001	DMKNLD
DMKCPPRR	000011	DMKCFG DMKCFS DMKCPB DMKDEF DMKIOE DMKMCH DMKSYM DMKUSO
DMKCPSET	000002	DMKCFC DMKSYM
DMKCPTRM	000002	DMKCFC DMKSYM
DMKCKP	000006	DMKCPI DMKPGT DMKSAV
DMKCKPRS	000001	DMKSAV
DMKCKPST	000001	DMKSAV
DMKCKPT	000002	DMKSAV DMKSYM
DMKCKSIN	000001	DMKWRM
DMKCKSPL	000018	DMKCSO DMKCSP DMKCSU DMKNLD DMKRSP DMKSPL DMKVSP DMKWRM
DMKCKSWM	000001	DMKWRM
DMKCNSED	000006	DMKGRF DMKRGA DMKRNH DMKSYM
DMKCNSEN	000003	DMKCPI DMKCPV DMKSYM
DMKCNSIC	000002	DMKQCN DMKSYM
DMKCNSIN	000002	DMKIOS DMKSYM
DMKCPBEX	000002	DMKCFC DMKSYM
DMKCPBNR	000002	DMKCFC DMKSYM
DMKCPBRS	000002	DMKCFC DMKSYM
DMKCPBRW	000002	DMKCFC DMKSYM
DMKCPBRY	000002	DMKCFC DMKSYM
DMKCPBSR	000002	DMKCFC DMKSYM
DMKCPE	000002	DMKFRE DMKLD00E
DMKCPEID	000005	DMKCPI DMKHVD DMKMON DMKSEP DMKSYM
DMKCPEND	000003	DMKCPI DMKSYM
DMKCPICD	000001	DMKSAV
DMKCPIEM	000002	DMKCNS DMKGRF
DMKCPINT	000002	DMKSAV DMKSSP
DMKCPVAA	000002	DMKHVD DMKSYM
DMKCPVAC	000002	DMKCFC DMKSYM
DMKCPVAE	000003	DMKCPI DMKRNH DMKSYM
DMKCPVDS	000002	DMKCFC DMKSYM
DMKCPVEN	000002	DMKCFC DMKSYM
DMKCPVH	000002	DMKCFC DMKSYM
DMKCPVLK	000002	DMKCFC DMKSYM
DMKCPVRY	000002	DMKCFC DMKSYM
DMKCPVSH	000002	DMKCFC DMKSYM
DMKCPVUL	000002	DMKCFC DMKSYM

Label	Count	References											
DMKCQGEN	000002	DMKCFC	DMKSYM										
DMKCQPRV	000002	DMKCFC	DMKSYM										
DMKCQREY	000002	DMKCFC	DMKSYM										
DMKCQRFI	000003	DMKCPI	DMKLOG	DMKSYM									
DMKCSOBS	000002	DMKCFC	DMKSYM										
DMKCSODR	000002	DMKCFC	DMKSYM										
DMKCSOFL	000002	DMKCFC	DMKSYM										
DMKCSOLD	000002	DMKCFC	DMKSYM										
DMKCSORP	000002	DMKCFC	DMKSYM										
DMKCSOSD	000005	DMKCPI	DMKCSP	DMKCSU	DMKRSP	DMKSYM							
DMKCSOSP	000002	DMKCFC	DMKSYM										
DMKCSOST	000002	DMKCFC	DMKSYM										
DMKCSOVL	000002	DMKCFC	DMKSYM										
DMKCSPCL	000002	DMKCFC	DMKSYM										
DMKCSPFR	000002	DMKCFC	DMKSYM										
DMKCSPHL	000002	DMKCFC	DMKSYM										
DMKCSPSP	000002	DMKCFC	DMKSYM										
DMKCSTAG	000001	DMKCFC											
DMKCSUCH	000002	DMKCFC	DMKSYM										
DMKCSUCR	000002	DMKCFC	DMKSYM										
DMKCSUPU	000002	DMKCFC	DMKSYM										
DMKCSUTR	000002	DMKCFC	DMKSYM										
DMKCVTBD	000060	DMKCDB	DMKCKS	DMKCPI	DMKCPV	DMKCQG	DMKCQP	DMKCQR	DMKCSU	DMKDEF	DMKDIA	DMKERM	DMKGRF
		DMKLNK	DMKLOG	DMKQCN	DMKRGA	DMKRSP	DMKSEP	DMKSPL	DMKTHI	DMKUSO	DMKVDB	DMKWRM	
DMKCVTBH	000205	DMKACO	DMKBLD	DMKCCH	DMKCDB	DMKCDS	DMKCFD	DMKCFG	DMKCFS	DMKCFT	DMKCNS	DMKCPB	DMKCPI
		DMKCPV	DMKCQG	DMKCQP	DMKCQR	DMKCSO	DMKCST	DMKDAS	DMKDEF	DMKDIA	DMKDSP	DMKLNK	DMKLOG
		DMKMSW	DMKNES	DMKNET	DMKNLD	DMKPAG	DMKPSA	DMKQCN	DMKRGA	DMKRNH	DMKRSP	DMKSAV	DMKSEP
		DMKSSP	DMKTHI	DMKTRC	DMKUSO	DMKVCA	DMKVCH	DMKVDB	DMKVDR	DMKVDS	DMKVER	DMKVMA	DMKVSP
DMKCVTDB	000032	DMKCDB	DMKCDS	DMKCFC	DMKCFG	DMKCFS	DMKCFT	DMKCQG	DMKCQR	DMKCSO	DMKCSP	DMKCST	DMKCSU
		DMKDEF	DMKGRF	DMKHVD	DMKMCC	DMKMSG	DMKNES	DMKRGA					
DMKCVTDT	000016	DMKCFS	DMKCPI	DMKCQR	DMKHVC	DMKLOG	DMKMID	DMKMON	DMKMSG	DMKNLD	DMKQCN	DMKRNH	DMKRSP
		DMKSEP	DMKSPL	DMKUSO	DMKVSP								
DMKCVTFP	000001	DMKCDB											
DMKCVTHB	000053	DMKCDB	DMKCDS	DMKCFC	DMKCFD	DMKCFG	DMKCFS	DMKCPB	DMKCPV	DMKCQG	DMKCQP	DMKCSO	DMKCSP
		DMKCST	DMKDEF	DMKDIA	DMKGRF	DMKLNK	DMKMCC	DMKNES	DMKNET	DMKNLD	DMKRGA	DMKSSP	DMKVDB
DMKDASER	000002	DMKIOS	DMKSYM										
DMKDASRD	000003	DMKCPV	DMKIOS	DMKSYM									
DMKDASSD	000003	DMKCCW	DMKCPV	DMKSYM									
DMKDEFIN	000002	DMKCFC	DMKSYM										
DMKDGDDK	000002	DMKHVC	DMKSYM										
DMKDIACP	000002	DMKCFC	DMKSYM										
DMKDIADR	000003	DMKCFP	DMKSYM										

Label	Count	References											
DMKDIAL	000002	DMKCFC	DMKSYM										
DMKDIASM	000003	DMKCCW	DMKSYM	DMKVCA									
DMKDMPAU	000002	DMKCFS	DMKCPI										
DMKDMPDK	000003	DMKPRG	DMKPSA	DMKSYM									
DMKDMPDV	000003	DMKCFS	DMKCPI	DMKCQR									
DMKDMPGR	000003	DMKPRG	DMKPSA	DMKSYM									
DMKDMPRC	000002	DMKCPI											
DMKDMPRS	000003	DMKCPV	DMKMCH	DMKSYM									
DMKDMPSF	000001	DMKCPI											
DMKDMPSW	000002	DMKCFS	DMKCQR										
DMKDMPTR	000001	DMKCDB											
DMKDRDDD	000002	DMKSPL	DMKSYM										
DMKDRDER	000002	DMKHVD	DMKSYM										
DMKDRDMP	000002	DMKHVD	DMKSYM										
DMKDRDSY	000002	DMKHVD	DMKSYM										
DMKDSPA	000001	DMKPRV											
DMKDSPAC	000002	DMKMON	DMKSYM										
DMKDSPB	000004	DMKCFM	DMKPRG	DMKPRV	DMKPSA								
DMKDSPBC	000002	DMKMON	DMKSYM										
DMKDSPCC	000002	DMKMON	DMKSYM										
DMKDSPCH	000093	DMKACO	DMKCFM	DMKCFP	DMKCNS	DMKCPB	DMKCPI	DMKCPV	DMKCSO	DMKDAS	DMKDGD	DMKDIA	DMKGIO
		DMKGRF	DMKHVC	DMKIOE	DMKIOS	DMKLOC	DMKMCH	DMKMON	DMKNLD	DMKPAG	DMKPGT	DMKPRG	DMKPRV
		DMKPSA	DMKPTR	DMKQCN	DMKRGA	DMKRGB	DMKRNH	DMKRSP	DMKSCH	DMKSEP	DMKSPL	DMKSYM	DMKTDK
		DMKTMR	DMKUDR	DMKUNT	DMKUSO	DMKVAT	DMKVCA	DMKVCN	DMKVDB	DMKVIO	DMKVMA	DMKVSP	
DMKDSPCK	000001	DMKMON											
DMKDSPIT	000001	DMKMON											
DMKDSPNP	000019	DMKCFS	DMKCPI	DMKCPV	DMKFRE	DMKMON	DMKPGS	DMKPTR	DMKSCH	DMKSYM	DMKVMA		
DMKDSPPT	000001	DMKMON											
DMKDSPQS	000001	DMKSYM											
DMKDSPRQ	000003	DMKSTK	DMKSYM										
DMKEIG80	000002	DMKIOG	DMKSYM										
DMKEMA00	000001	DMKERM											
DMKEMB00	000001	DMKERM											
DMKEPSWD	000004	DMKLNK	DMKLOG	DMKSYM									
DMKERMSG	000056	DMKACO	DMKBLD	DMKCDB	DMKCDS	DMKCFC	DMKCFD	DMKCFG	DMKCFS	DMKCFT	DMKCKS	DMKCNS	DMKCPB
		DMKCPV	DMKCQG	DMKCQP	DMKCQR	DMKCSO	DMKCSP	DMKCST	DMKCSU	DMKDEF	DMKDIA	DMKIOF	DMKIOG
		DMKLNK	DMKLOG	DMKMCC	DMKMCH	DMKMID	DMKMON	DMKMSG	DMKNES	DMKNET	DMKNLD	DMKRGA	DMKRNH
		DMKRSP	DMKSNC	DMKSYM	DMKTHI	DMKTRA	DMKUSO	DMKVCH	DMKVDB	DMKVDS	DMKVMA	DMKVSP	DMKWRM
DMKFCBLD	000002	DMKCSO											

Label	Count	References											
DMKFREE	000320	DMKACO	DMKBLD	DMKBSC	DMKCCH	DMKCCW	DMKCDB	DMKCDS	DMKCFC	DMKCFD	DMKCFG	DMKCFM	DMKCFP
		DMKCFS	DMKCKS	DMKCNS	DMKCPB	DMKCPI	DMKCPV	DMKCQG	DMKCQP	DMKCQR	DMKCSO	DMKCSP	DMKCST
		DMKCSU	DMKDAS	DMKDEF	DMKDGD	DMKDIA	DMKDRD	DMKDSP	DMKERM	DMKGIO	DMKGRF	DMKHVC	DMKHVD
		DMKIOE	DMKIOF	DMKIOG	DMKIOS	DMKISM	DMKLNK	DMKLOC	DMKLOG	DMKMCC	DMKMCH	DMKMON	DMKMSG
		DMKMSW	DMKNES	DMKNET	DMKNLD	DMKPAG	DMKPGT	DMKPSA	DMKPTR	DMKQCN	DMKRGA	DMKRGB	DMKRNH
		DMKRPA	DMKRSE	DMKRSP	DMKSCH	DMKSPL	DMKSYM	DMKTAP	DMKTDK	DMKTHI	DMKTMR	DMKTRA	DMKUDR
		DMKUSO	DMKVAT	DMKVCA	DMKVCH	DMKVCN	DMKVDB	DMKVDR	DMKVDS	DMKVER	DMKVIO	DMKVMA	DMKVSP
		DMKWRM											
DMKFREHI	000002	DMKCPI	DMKSYM										
DMKFRELG	000002	DMKCPI	DMKSYM										
DMKFRELO	000003	DMKCPI	DMKSYM	DMKUSO									
DMKFRELS	000001	DMKSYM											
DMKFRENP	000002	DMKSYM	DMKUSO										
DMKFRERS	000002	DMKSYM	DMKUSO										
DMKFRESV	000002	DMKCPI	DMKSYM										
DMKFRET	000285	DMKACO	DMKBLD	DMKBSC	DMKCCH	DMKCCW	DMKCDB	DMKCFD	DMKCFG	DMKCFM	DMKCFP	DMKCFS	DMKCKS
		DMKCNS	DMKCPB	DMKCPI	DMKCPV	DMKCQG	DMKCQP	DMKCQR	DMKCSO	DMKCSP	DMKCST	DMKCSU	DMKDAS
		DMKDEF	DMKDGD	DMKDIA	DMKDRD	DMKDSP	DMKERM	DMKGIO	DMKGRF	DMKHVC	DMKHVD	DMKIOE	DMKIOF
		DMKIOS	DMKLNK	DMKLOC	DMKLOG	DMKMCC	DMKMON	DMKMSG	DMKMSW	DMKNES	DMKNET	DMKNLD	DMKPAG
		DMKPGS	DMKPGT	DMKPSA	DMKPTR	DMKQCN	DMKRGA	DMKRGB	DMKRNH	DMKRSE	DMKRSP	DMKSCH	DMKSEP
		DMKSPL	DMKSYM	DMKTAP	DMKTDK	DMKTHI	DMKTMR	DMKTRA	DMKTRC	DMKUDR	DMKUNT	DMKUSO	DMKVAT
		DMKVCA	DMKVCH	DMKVCN	DMKVDB	DMKVDR	DMKVDS	DMKVER	DMKVIO	DMKVMA	DMKVSP		
DMKFRETR	000004	DMKCPI	DMKPTR	DMKSYM									
DMKGIOEX	000002	DMKHVC	DMKSYM										
DMKGRFEN	000002	DMKCPV	DMKSYM										
DMKGRFIC	000002	DMKQCN	DMKSYM										
DMKGRFIN	000002	DMKIOS	DMKSYM										
DMKHVCAL	000002	DMKPRV	DMKSYM										
DMKHVCDI	000002	DMKMON	DMKSYM										
DMKHVCPC	000001	DMKDRD											
DMKHVDAL	000001	DMKHVC											
DMKIOCVT	000001	DMKIOF											
DMKIOECC	000001	DMKCCH											
DMKIOECQ	000002	DMKIOF											
DMKIOEES	000005	DMKIOF	DMKIOG										
DMKIOEFL	000001	DMKCPI											
DMKIOEFM	000002	DMKHVD	DMKSYM										
DMKIOEIQ	000003	DMKIOF											
DMKIOEIR	000001	DMKCFS											
DMKIOEMC	000001	DMKMCH											
DMKIOEMP	000002	DMKIOF	DMKIOG										
DMKIOEMQ	000002	DMKIOF											

Label	Count	References											
DMKIOEMS	000005	DMKIOF	DMKIOG										
DMKIOEMX	000002	DMKIOF	DMKIOG										
DMKIOENI	000002	DMKIOF	DMKIOG										
DMKIOENQ	000001	DMKIOF											
DMKIOECP	000002	DMKIOF	DMKIOG										
DMKIOERN	000002	DMKRGA	DMKRNH										
DMKIOERP	000001	DMKIOF											
DMKIOERQ	000002	DMKIOF											
DMKIOERR	000004	DMKCNS	DMKGRF	DMKIOS	DMKSYM								
DMKIOESD	000001	DMKDAS											
DMKIOESQ	000001	DMKIOF											
DMKIOESR	000004	DMKCPV	DMKNES	DMKNET									
DMKIOEST	000006	DMKBSC	DMKCNS	DMKDAS	DMKGRF	DMKRSE	DMKTAP						
DMKIOEVQ	000001	DMKIOF											
DMKIOEVR	000001	DMKVER											
DMKIOF	000001	DMKSYM											
DMKIOFC1	000001	DMKIOE											
DMKIOFIN	000002	DMKIOE											
DMKIOFM1	000001	DMKIOE											
DMKIOFOB	000003	DMKIOE											
DMKIOFST	000001	DMKIOE											
DMKIOFVR	000001	DMKIOE											
DMKIOG	000001	DMKSYM											
DMKIOGF1	000001	DMKIOE											
DMKIOGF2	000001	DMKIOE											
DMKIOSCT	000002	DMKMON	DMKSYM										
DMKIOSER	000001	DMKDSP											
DMKIOSHA	000004	DMKCFP	DMKCPV	DMKDIA	DMKSYM								
DMKIOSIN	000002	DMKCPI	DMKSYM										
DMKIOSQR	000029	DMKACO	DMKCFP	DMKCNS	DMKCPB	DMKCPI	DMKCPV	DMKCSO	DMKDAS	DMKDIA	DMKGRF	DMKMON	DMKNLD
		DMKPAG	DMKRGA	DMKRGB	DMKRNH	DMKRSP	DMKSEP	DMKSPL	DMKSYM	DMKTDK	DMKUDR	DMKVDB	DMKVDR
DMKIOSQV	000005	DMKDGD	DMKGIO	DMKSYM	DMKVIC								
DMKIOSRC	000001	DMKDSP											
DMKIOSRW	000002	DMKCPB	DMKVDR										
DMKISMTR	000002	DMKCCW	DMKSYM										
DMKLNKIN	000002	DMKCFC	DMKSYM										
DMKLNKSB	000002	DMKLOG	DMKSYM										
DMKLOC	000001	DMKSYM											
DMKLOCK	000002	DMKLNK											
DMKLOCKD	000014	DMKCFP	DMKCKS	DMKDEF	DMKLNK	DMKTAP	DMKTRA	DMKTRC	DMKUDR	DMKUSO	DMKVDB		
DMKLOCKQ	000011	DMKCFP	DMKCKS	DMKDEF	DMKTAP	DMKTRA	DMKTRC	DMKUDR	DMKUSO	DMKVDB			
DMKLOCKT	000001	DMKCKS											

Label	Count	References							
DMKLOGA	000001	DMKCFC							
DMKLOGON	000003	DMKCFC	DMKSYM						
DMKLOGCP	000002	DMKCPI	DMKSYM						
DMKMCCCL	000002	DMKCFC	DMKSYM						
DMKMCHAR	000005	DMKCCH	DMKCFS	DMKIOG	DMKSYM				
DMKMCHBL	000001	DMKIOG							
DMKMCHIN	000002	DMKCPI	DMKSYM						
DMKMCHMS	000002	DMKCFS	DMKSYM						
DMKMCHRD	000001	DMKIOG							
DMKMIDNT	000002	DMKSCH	DMKSYM						
DMKMONIO	000001	DMKSYM							
DMKMONMI	000001	DMKMCC							
DMKMONSH	000002	DMKCPV	DMKMCC						
DMKMONTH	000002	DMKMCC	DMKSYM						
DMKMONTI	000001	DMKMCC							
DMKMSGEC	000002	DMKCFC	DMKSYM						
DMKMSGMS	000003	DMKCFC	DMKSYM						
DMKMSGWN	000003	DMKCFC	DMKSYM						
DMKMSWR	000007	DMKBSC	DMKCNS	DMKDAS	DMKGRF	DMKRSE	DMKSYM	DMKTAP	
DMKNEMOP	000004	DMKSYM	DMKTRC						
DMKNESDS	000001	DMKNET							
DMKNESEP	000001	DMKNET							
DMKNESHD	000001	DMKNET							
DMKNESPL	000001	DMKNET							
DMKNESTR	000001	DMKNET							
DMKNESWN	000001	DMKNET							
DMKNETAE	000001	DMKCPI							
DMKNETWK	000002	DMKCFC	DMKSYM						
DMKNLDMP	000003	DMKNET	DMKRNH	DMKSYM					
DMKNLDR	000004	DMKCPI	DMKNET	DMKRNH	DMKSYM				
DMKOPRWT	000008	DMKCCH	DMKCKP	DMKDMP	DMKMCH	DMKPAG	DMKRSP	DMKSAV	DMKSYM
DMKPAGCC	000002	DMKMON	DMKSYM						
DMKPAGHI	000001	DMKCPI							
DMKPAGIC	000001	DMKSYM							
DMKPAGIO	000007	DMKCDS	DMKPTR	DMKRPA	DMKSYM				
DMKPAGLO	000001	DMKCPI							
DMKPAGFS	000002	DMKMON	DMKSYM						
DMKPAGQ	000002	DMKPTR	DMKSYM						
DMKPAGQR	000001	DMKCQR							
DMKPAGSP	000001	DMKSYM							
DMKPAGST	000002	DMKCPI	DMKSYM						
DMKPERIL	000004	DMKPRG	DMKPRV	DMKVAT					

Label	Count	References											
DMKPERT	000003	DMKCFP	DMKDSP	DMKUSO									
DMKPGS	000001	DMKSYM											
DMKPGSFS	000001	DMKCFG											
DMKPGSPO	000007	DMKCFG	DMKCFP	DMKCPB	DMKDEF	DMKMCH	DMKSYM	DMKUSO					
DMKPGSPP	000004	DMKCFG	DMKCFP	DMKCPV	DMKSYM								
DMKPGSPS	000002	DMKCFG											
DMKPGSSS	000001	DMKHVC											
DMKPGTBN	000002	DMKCPI	DMKSYM										
DMKPGTCG	000001	DMKNLD											
DMKPGTPG	000003	DMKCPI	DMKPTR	DMKSYM									
DMKPGTPR	000003	DMKPGS	DMKPTR	DMKRPA									
DMKPGTP0	000003	DMKCPI	DMKTDK	DMKVDB									
DMKPGTP4	000003	DMKCPI	DMKTDK	DMKVDB									
DMKPGTP5	000004	DMKCPI	DMKTDK	DMKUSO	DMKVDB								
DMKPGTSD	000004	DMKDRD	DMKNLD	DMKSPL									
DMKPGTSG	000006	DMKRSP	DMKSPL	DMKVSP									
DMKPGTSP	000001	DMKRPA											
DMKPGTSR	000001	DMKSPL											
DMKPGTTM	000004	DMKCPI	DMKSYM	DMKVDB									
DMKPGTTU	000003	DMKCKS	DMKCPI	DMKSYM									
DMKPGTT0	000004	DMKCPI	DMKTDK	DMKVDB									
DMKPGTT4	000004	DMKCPI	DMKTDK	DMKVDB									
DMKPGTT5	000004	DMKCPI	DMKTDK	DMKVDB									
DMKPGTVG	000022	DMKCFG	DMKCKS	DMKCST	DMKDRD	DMKIOF	DMKIOG	DMKNLD	DMKRSP	DMKSEP	DMKSPL	DMKUDR	DMKVSP
		DMKWRM											
DMKPGTVR	000020	DMKCFG	DMKCKS	DMKCST	DMKDRD	DMKIOF	DMKIOG	DMKNLD	DMKRSP	DMKSEP	DMKUDR	DMKVSP	DMKWRM
DMKPGT4P	000003	DMKCPI	DMKTDK	DMKVDB									
DMKPGT4T	000004	DMKCPI	DMKTDK	DMKVDB									
DMKPGT5P	000003	DMKCPI	DMKTDK	DMKVDB									
DMKPGT5T	000004	DMKCPI	DMKTDK	DMKVDB									
DMKPGT90	000002	DMKCPI	DMKVDB										
DMKPRGCT	000002	DMKMON	DMKSYM										
DMKPRGC8	000012	DMKMCC	DMKMON	DMKSYM									
DMKPRGGR	000007	DMKMON	DMKSYM										
DMKPRGIN	000002	DMKCPI	DMKSYM										
DMKPRGMC	000012	DMKCPV	DMKDMP	DMKMCC	DMKMON	DMKSYM							
DMKPRGMI	000002	DMKMCC	DMKMON										
DMKPRGRF	000002	DMKPSA	DMKSYM										
DMKPRGSM	000008	DMKHVC	DMKPRV	DMKSYM	DMKTMR	DMKVAT							
DMKPRGTI	000006	DMKMCC	DMKMON										
DMKPRVCD	000001	DMKMON											
DMKPRVCE	000001	DMKMON											

Label	Count	References											
DMKPRVCH	000001	DMKMON											
DMKPRVCP	000001	DMKMON											
DMKPRVCS	000001	DMKMON											
DMKPRVCT	000001	DMKMON											
DMKPRVDI	000001	DMKMON											
DMKPRVEK	000001	DMKMON											
DMKPRVEP	000001	DMKMON											
DMKPRVIK	000001	DMKMON											
DMKPRVIP	000001	DMKMON											
DMKPRVLC	000001	DMKMON											
DMKPRVLG	000002	DMKPRG	DMKSYM										
DMKPRVLP	000001	DMKMON											
DMKPRVLR	000001	DMKMON											
DMKPRVMN	000001	DMKMON											
DMKPRVMO	000001	DMKMON											
DMKPRVMS	000001	DMKMON											
DMKPRVNC	000002	DMKMON	DMKSYM										
DMKPRVPB	000001	DMKMON											
DMKPRVPE	000001	DMKMON											
DMKPRVPT	000001	DMKMON											
DMKPRVRR	000001	DMKMON											
DMKPRVTC	000001	DMKMON											
DMKPRVTE	000001	DMKMON											
DMKPSA	000001	DMKLD00E											
DMKPSACC	000016	DMKCDS	DMKDGD	DMKVCN	DMKVSP								
DMKPSADU	000002	DMKCPI	DMKSYM										
DMKPSAEX	000002	DMKCPI	DMKSYM										
DMKPSAFP	000006	DMKPRV	DMKTMR										
DMKPSAHI	000001	DMKCPI											
DMKPSALO	000001	DMKCPI											
DMKPSANS	000002	DMKCPI	DMKSYM										
DMKPSANX	000001	DMKMON											
DMKPSARG	000001	DMKSYM											
DMKPSARR	000001	DMKTRC											
DMKPSARS	000003	DMKTRC											
DMKPSARX	000002	DMKTRC											
DMKPSASC	000013	DMKCDS	DMKCFD	DMKDGD	DMKTRC	DMKVCN	DMKVSP						
DMKPSASP	000012	DMKDRD	DMKHVC	DMKHVD	DMKPRV	DMKTMR	DMKTRC						
DMKPSASV	000002	DMKCPI	DMKSYM										
DMKPTRAN	000111	DMKCCW	DMKCDB	DMKCDS	DMKCFD	DMKCFG	DMKCKS	DMKCNS	DMKCPB	DMKCPI	DMKCPV	DMKDGD	DMKDRD
		DMKDSP	DMKERM	DMKGIO	DMKGRF	DMKHVC	DMKHVD	DMKIOF	DMKIOG	DMKISM	DMKMCC	DMKNLD	DMKPGS
		DMKPRG	DMKPRV	DMKPSA	DMKRGA	DMKRGB	DMKRPA	DMKRSP	DMKSCH	DMKSEP	DMKSNC	DMKSPL	DMKSYM
		DMKTMR	DMKUDR	DMKVAT	DMKVCB	DMKVCN	DMKVER	DMKVIO	DMKVMA	DMKVSP	DMKWRM		

Label	Count	References											
DMKPTRCS	000001	DMKMON											
DMKPTRCT	000001	DMKSYM											
DMKPTRFA	000001	DMKCPI											
DMKPTRFC	000002	DMKMON	DMKSYM										
DMKPTRFD	000001	DMKDSP											
DMKPTRFE	000001	DMKDSP											
DMKPTRFF	000004	DMKCQR	DMKMON	DMKPAG	DMKSYM								
DMKPTRFL	000001	DMKSCH											
DMKPTRFN	000002	DMKCPI	DMKMON										
DMKPTRFP	000001	DMKDSP											
DMKPTRFR	000005	DMKCCW	DMKDGD	DMKFRE	DMKMCC								
DMKPTRFT	000010	DMKDGD	DMKFRE	DMKMCH	DMKMON	DMKPGS	DMKRPA	DMKUDR	DMKUNT				
DMKPTRF0	000001	DMKMON											
DMKPTRF1	000001	DMKCPI											
DMKPTRLK	000005	DMKACO	DMKCPI	DMKCPV	DMKSPL								
DMKPTRPR	000002	DMKMON	DMKSYM										
DMKPTRPW	000003	DMKCFP	DMKPGS	DMKVAT									
DMKPTRRC	000005	DMKCFS	DMKDSP	DMKMON	DMKPGS	DMKSYM							
DMKPTRRF	000001	DMKMON											
DMKPTRRL	000003	DMKCFS	DMKSCH	DMKUSO									
DMKPTRRQ	000002	DMKPAG	DMKSYM										
DMKPTRRT	000001	DMKDSP											
DMKPTRRU	000002	DMKCFS	DMKUSO										
DMKPTRSC	000005	DMKMON	DMKPGS	DMKSYM	DMKVMA								
DMKPTRSS	000004	DMKCQR	DMKMON	DMKPAG	DMKSYM								
DMKPTRSW	000001	DMKMON											
DMKPTRUL	000036	DMKACO	DMKCCW	DMKCFG	DMKCKS	DMKCPI	DMKCPV	DMKCSO	DMKDGD	DMKIOF	DMKISM	DMKMON	DMKNLD
		DMKPSA	DMKRPA	DMKSEP	DMKSNC	DMKSPL	DMKUNT	DMKVIO	DMKVSP				
DMKPTRU1	000002	DMKCPI	DMKSCH										
DMKPTRWQ	000004	DMKCDS	DMKPAG	DMKRPA	DMKSYM								
DMKQCNCL	000017	DMKCNS	DMKDIA	DMKGRF	DMKNES	DMKNLD	DMKPSA	DMKRGA	DMKRNH	DMKSYM			
DMKQCNET	000012	DMKCNS	DMKGRF	DMKRGA	DMKRGB	DMKRNH	DMKSYM						
DMKQCNRD	000014	DMKCFG	DMKCFM	DMKCFS	DMKCPI	DMKLNK	DMKMSG	DMKMSW	DMKNLD	DMKSYM	DMKVCN		
DMKQCNSY	000004	DMKCFP	DMKCKS	DMKLOG	DMKSYM								
DMKQCNTO	000008	DMKCNS	DMKGRF	DMKNES	DMKNLD	DMKRGA	DMKRNH	DMKSYM					
DMKQCNWT	000174	DMKACO	DMKBLD	DMKCCH	DMKCDB	DMKCDS	DMKCFC	DMKCFD	DMKCFG	DMKCFM	DMKCFS	DMKCPB	DMKCPI
		DMKCPV	DMKCQG	DMKCQP	DMKCQR	DMKCSO	DMKCSU	DMKDAS	DMKDEF	DMKDIA	DMKDSP	DMKERM	DMKGRF
		DMKIOE	DMKLNK	DMKLOG	DMKMCC	DMKMCH	DMKMID	DMKMSG	DMKMSW	DMKNES	DMKNET	DMKNLD	DMKPGT
		DMKPRG	DMKPSA	DMKPTR	DMKRGA	DMKRNH	DMKRSP	DMKSPL	DMKSYM	DMKTHI	DMKTRA	DMKTRC	DMKUDR
		DMKUSO	DMKVCA	DMKVCH	DMKVCN	DMKVDB	DMKVDR	DMKVER	DMKWRM				
DMKRGAIN	000003	DMKIOS	DMKRGB	DMKSYM									
DMKRGBEN	000004	DMKNES	DMKNET	DMKSYM									

Label	Count	References											
DMKRGBIC	000004	DMKQCN	DMKRGA	DMKSYM									
DMKRGBMT	000001	DMKRGA											
DMKRGBSN	000001	DMKRGA											
DMKRIO	000002	DMKSSP											
DMKRIOCH	000002	DMKSSP	DMKSYM										
DMKRIOCN	000006	DMKCPI	DMKGRF	DMKOPR	DMKSSP	DMKSYM							
DMKRIOCU	000004	DMKSSP	DMKSYM										
DMKRIODV	000006	DMKCPR	DMKSSP	DMKSYM									
DMKRIOPR	000004	DMKCQR	DMKDMP	DMKSSP	DMKSYM								
DMKRIOPU	000002	DMKSSP	DMKSYM										
DMKRIORD	000002	DMKSSP	DMKSYM										
DMKRIORN	000014	DMKBLD	DMKCPI	DMKCQP	DMKDIA	DMKNES	DMKNET	DMKRNH	DMKSYM				
DMKRNHCT	000001	DMKSYM											
DMKRNHIC	000002	DMKQCN	DMKSYM										
DMKRNHIN	000003	DMKIOS	DMKNLD	DMKSYM									
DMKRNHND	000015	DMKDIA	DMKNES	DMKNET	DMKPSA	DMKSYM							
DMKRNHTG	000001	DMKSYM											
DMKRNHTR	000004	DMKNES	DMKSYM										
DMKRNTBL	000002	DMKNLD	DMKSNC										
DMKRPAGT	000043	DMKCFG	DMKCKS	DMKCST	DMKDRD	DMKHVD	DMKIOF	DMKIOG	DMKNLD	DMKRSP	DMKSPL	DMKSYM	DMKUDR
		DMKVSP	DMKWRM										
DMKRPAPT	000024	DMKCFG	DMKCKS	DMKCPI	DMKCST	DMKIOF	DMKIOG	DMKNLD	DMKRSP	DMKSNC	DMKSPL	DMKSYM	DMKVSP
		DMKWRM											
DMKRSERR	000002	DMKRSP	DMKSYM										
DMKRSPAC	000003	DMKCKP	DMKSYM	DMKWRM									
DMKRSPCV	000005	DMKCKP	DMKSYM	DMKWRM									
DMKRSPDL	000009	DMKCKP	DMKSPL	DMKSYM	DMKWRM								
DMKRSPER	000002	DMKIOS	DMKSYM										
DMKRSPEX	000005	DMKACO	DMKCSO	DMKIOS	DMKSPL	DMKSYM							
DMKRSPHQ	000009	DMKCKP	DMKCKS	DMKCQR	DMKCSP	DMKSPL	DMKSYM	DMKWRM					
DMKRSPID	000008	DMKCKP	DMKCKS	DMKDMP	DMKNLD	DMKSPL	DMKSYM	DMKWRM					
DMKRSPPR	000005	DMKCKP	DMKCQR	DMKSYM	DMKWRM								
DMKRSPPU	000004	DMKCKP	DMKCQR	DMKSYM	DMKWRM								
DMKRSPRD	000005	DMKCKP	DMKCQR	DMKSYM	DMKWRM								
DMKRSPUR	000002	DMKCQP	DMKSYM										
DMKRSP83	000002	DMKRSE	DMKSYM										
DMKSAV	000004	DMKCKP	DMKCPI										
DMKSAVRS	000001	DMKCKP											
DMKSCHAL	000002	DMKMON	DMKSYM										
DMKSCHAP	000002	DMKCFS	DMKSYM										
DMKSCHAU	000004	DMKCFS	DMKSYM	DMKUSO									
DMKSCHCO	000001	DMKTHI											

Label	Count	References											
DMKSCHCP	000002	DMKBLD	DMKSYM										
DMKSCHCT	000002	DMKMON	DMKSYM										
DMKSCHCU	000001	DMKTHI											
DMKSCHDL	000014	DMKDSP	DMKIOS	DMKLOG	DMKPTR	DMKQCN	DMKRPA	DMKSYM	DMKUSO	DMKVCA	DMKVIO		
DMKSCHIB	000001	DMKSYM											
DMKSCHLI	000002	DMKCPI	DMKTHI										
DMKSCHMD	000002	DMKCPI	DMKSYM										
DMKSCHN1	000006	DMKDSP	DMKMON	DMKPTR	DMKSYM	DMKTMR							
DMKSCHN2	000005	DMKDSP	DMKMON	DMKPTR	DMKSYM	DMKTMR							
DMKSCHPB	000001	DMKSYM											
DMKSCHPD	000001	DMKSYM											
DMKSCHPG	000003	DMKCFS	DMKCQR	DMKSYM									
DMKSCHPU	000003	DMKMON	DMKSYM										
DMKSCHQ1	000003	DMKCPI	DMKMON	DMKSYM									
DMKSCHQ2	000002	DMKCPI	DMKSYM										
DMKSCHRL	000002	DMKDSP	DMKSYM										
DMKSCHRT	000016	DMKCFC	DMKCFM	DMKCFP	DMKCFS	DMKGRF	DMKLOG	DMKMCC	DMKMON	DMKQCN	DMKRGA	DMKRGB	DMKSYM
		DMKTMR	DMKUSO										
DMKSCHSC	000001	DMKTHI											
DMKSCHST	000011	DMKCFC	DMKCPI	DMKGRF	DMKMCC	DMKMID	DMKMON	DMKQCN	DMKRGA	DMKSYM	DMKTMR		
DMKSCHS1	000001	DMKTHI											
DMKSCHS2	000001	DMKTHI											
DMKSCHTI	000001	DMKCPI											
DMKSCHTQ	000002	DMKPSA	DMKSYM										
DMKSCHUB	000001	DMKSYM											
DMKSCHW1	000003	DMKMON	DMKSYM										
DMKSCHW2	000003	DMKMON	DMKSYM										
DMKSCH80	000003	DMKCFS	DMKLOG	DMKSYM									
DMKSCNAU	000021	DMKCFD	DMKCFS	DMKCPV	DMKCQG	DMKCQP	DMKCQR	DMKCSU	DMKDIA	DMKLNK	DMKLOG	DMKMSG	DMKRNH
		DMKSPL	DMKUSO	DMKVDB									
DMKSCNFD	000178	DMKCDB	DMKCDS	DMKCFC	DMKCFD	DMKCFG	DMKCFM	DMKCFS	DMKCFT	DMKCPB	DMKCPV	DMKCQG	DMKCQP
		DMKCQR	DMKCSO	DMKCSP	DMKCST	DMKCSU	DMKDEF	DMKDIA	DMKLNK	DMKLOG	DMKMCC	DMKMCH	DMKMSG
		DMKNES	DMKNET	DMKNLD	DMKRSP	DMKTHI	DMKTRA	DMKUSO	DMKVDB				
DMKSCNLI	000001	DMKLNK											
DMKSCNRD	000048	DMKBLD	DMKCFS	DMKCKS	DMKCNS	DMKCPI	DMKCPV	DMKCQG	DMKCQP	DMKCQR	DMKDIA	DMKDMP	DMKGRF
		DMKLOG	DMKNES	DMKNET	DMKNLD	DMKPAG	DMKPSA	DMKQCN	DMKRGA	DMKRSP	DMKSEP	DMKTHI	DMKTRC
		DMKUSO	DMKVCH	DMKVDB	DMKVDR	DMKVDS	DMKVER						
DMKSCNRN	000015	DMKCPV	DMKCQG	DMKCQP	DMKDIA	DMKMSW	DMKQCN	DMKRSP	DMKTRC	DMKUSO	DMKVDB	DMKVDR	
DMKSCNRU	000048	DMKCCH	DMKCFD	DMKCFS	DMKCKS	DMKCNS	DMKCPI	DMKCPV	DMKCQP	DMKCSO	DMKDAS	DMKDIA	DMKDMP
		DMKGRF	DMKHVD	DMKIOG	DMKIOS	DMKLOG	DMKMCC	DMKNES	DMKNLD	DMKRGA	DMKRNH	DMKRSP	DMKVCH
		DMKVDB	DMKVDS	DMKWRM									
DMKSCNVD	000011	DMKCFT	DMKCPV	DMKCQP	DMKCST	DMKDEF	DMKDIA	DMKHVD	DMKLOG	DMKVSP			

Label	Count	References											
DMKSCNVN	000009	DMKCPV	DMKCQG	DMKCST	DMKDEF	DMKLNK	DMKLOG	DMKTRC	DMKVDB				
DMKSCNVS	000011	DMKCFG	DMKCPI	DMKLNK	DMKNLD	DMKSNC	DMKVDB						
DMKSCNVU	000063	DMKCFD	DMKCFG	DMKCFP	DMKCPB	DMKCPV	DMKCQG	DMKCQP	DMKCSO	DMKCSP	DMKCST	DMKDEF	DMKDGD
		DMKDIA	DMKDRD	DMKDSP	DMKGIO	DMKHVC	DMKHVD	DMKLNK	DMKLOG	DMKNLD	DMKTHI	DMKTRC	DMKUSO
		DMKVCA	DMKVCH	DMKVCN	DMKVDB	DMKVDS	DMKVER	DMKVIO	DMKVSP				
DMKSEPBR	000001	DMKSYM											
DMKSEPSP	000002	DMKRSP	DMKSYM										
DMKSEV70	000002	DMKIOG	DMKSYM										
DMKSIX60	000002	DMKIOG	DMKSYM										
DMKSLC		DMKBLD											
DMKSNCP	000002	DMKHVD	DMKSYM										
DMKSNTBL	000001	DMKCFG											
DMKSPLCR	000002	DMKRSP	DMKSYM										
DMKSPLCV	000002	DMKSYM	DMKVSP										
DMKSPLDL	000007	DMKCSO	DMKCSU	DMKRSP	DMKSYM	DMKVSP							
DMKSPLDR	000001	DMKSYM											
DMKSPLOR	000002	DMKRSP	DMKSYM										
DMKSPLOV	000002	DMKSYM	DMKVSP										
DMKSTKCP	000043	DMKACO	DMKCFM	DMKCFP	DMKCPV	DMKDIA	DMKGRF	DMKIOE	DMKIOF	DMKIOS	DMKLOC	DMKMCH	DMKMON
		DMKPAG	DMKPGT	DMKPTR	DMKQCN	DMKRGA	DMKRGB	DMKRNH	DMKRSP	DMKSPL	DMKUSO	DMKVCA	DMKVMA
		DMKVSP											
DMKSTKIO	000024	DMKACO	DMKCFP	DMKCPV	DMKCSO	DMKCSP	DMKCSU	DMKDIA	DMKIOS	DMKNLD	DMKPSA	DMKSPL	DMKTMR
		DMKUNT	DMKVCA	DMKVIO									
DMKSYM	000002	DMKCPI	DMKDRD										
DMKSYMTB	000001	DMKCPI											
DMKSYSCD	000001	DMKCFT											
DMKSYSCH	000003	DMKCKS	DMKDMP										
DMKSYSCK	000020	DMKACO	DMKCCH	DMKCKP	DMKDIA	DMKDMP	DMKLOG	DMKMCH	DMKVDS				
DMKSYSCN	000006	DMKCKS											
DMKSYSCS	000001	DMKSYM											
DMKSYSDT	000004	DMKCFS	DMKCKP	DMKLOG	DMKWRM								
DMKSYSDU	000002	DMKCPI	DMKNLD										
DMKSYSDW	000009	DMKCFS	DMKCPI	DMKCQR	DMKLOG	DMKMID	DMKUSO						
DMKSYSER	000003	DMKHVD	DMKIOG										
DMKSYSES	000001	DMKCFT											
DMKSYSLB	000002	DMKLOC											
DMKSYSLC	000001	DMKSYM											
DMKSYSLD	000001	DMKCFT											
DMKSYSLE	000002	DMKBLD	DMKCFT										
DMKSYSLG	000006	DMKCFS	DMKCKP	DMKCQR	DMKLOG	DMKWRM							
DMKSYSLL	000002	DMKBLD											
DMKSYSLW	000003	DMKCFS	DMKLOG										
DMKSYSMA	000001	DMKLOG											

Label	Count	References
DMKSYSMU	000004	DMKLOG
DMKSYSND	000004	DMKCQR DMKDIA DMKMON
DMKSYSNM	000014	DMKCQR DMKGRF DMKLOG DMKMON DMKQCN DMKUSO
DMKSYSNU	000003	DMKCPI DMKSAV DMKSSP
DMKSYSOC	000010	DMKCKP DMKCPI DMKMON DMKRSP DMKSPL DMKSYM DMKUDR DMKVDB
DMKSYSOP	000001	DMKSYM
DMKSYSOW	000022	DMKCKP DMKCKS DMKCPI DMKDRD DMKMON DMKPAG DMKPGT DMKPTR DMKRSP DMKSPL DMKSYM DMKUDR DMKVDB DMKWRM
DMKSYSPL	000003	DMKUDR
DMKSYSRM	000029	DMKCCH DMKCCW DMKCDB DMKCDS DMKCKP DMKCPI DMKCQP DMKDIA DMKDMP DMKERM DMKFRE DMKHVD DMKPTR DMKRSP DMKSYM DMKTRC DMKUNT DMKVCA
DMKSYSRS	000002	DMKSAV DMKSYM
DMKSYSRV	000009	DMKCFS DMKCPI DMKCQP DMKDMP DMKSYM
DMKSYSTI	000006	DMKCPI DMKCQR DMKLOG DMKMID DMKUSO
DMKSYSTM	000002	DMKCFS DMKLOG
DMKSYSTP	000003	DMKCKP DMKRSP DMKSAV
DMKSYSTZ	000004	DMKCPI DMKIOE DMKSAV
DMKSYSUD	000004	DMKCPI DMKUDR
DMKSYSVL	000004	DMKCPI DMKSAV DMKSYM
DMKSYSVM	000001	DMKSYM
DMKSYSWM	000003	DMKCKP DMKRSP DMKWRM
DMKTAPER	000002	DMKIOS DMKSYM
DMKTBLCI	000001	DMKCNS
DMKTBLCO	000001	DMKCNS
DMKTBLGR	000004	DMKGRF DMKRGA DMKRGB
DMKTBLPI	000001	DMKCNS
DMKTBLPO	000001	DMKCNS
DMKTBLTI	000001	DMKCNS
DMKTBLTO	000001	DMKCNS
DMKTBLUP	000004	DMKCNS DMKGRF DMKRGA DMKVCN
DMKTBMMI	000001	DMKCNS
DMKTBMMO	000001	DMKCNS
DMKTBMNI	000001	DMKCNS
DMKTBMNO	000001	DMKCNS
DMKTBMZI	000002	DMKGRF DMKRGA
DMKTBMZO	000002	DMKGRF DMKRGB
DMKTDKGT	000002	DMKSYM DMKVDS
DMKTDKRL	000002	DMKSYM DMKVDR
DMKTHIEN	000001	DMKCFC
DMKTMRCK	000001	DMKBLD
DMKTMRPT	000007	DMKACO DMKCKP DMKHVC DMKRSP DMKTHI DMKVSP
DMKTMRTN	000002	DMKPRV DMKSYM

Label	Count	References								
DMKTMRVT	000001	DMKPSA								
DMKTRACE	000002	DMKCFC	DMKSYM							
DMKTRCEX	000002	DMKDSP	DMKSYM							
DMKTRCIO	000001	DMKDSP								
DMKTRCIT	000005	DMKCDS	DMKCFC	DMKDSP	DMKPSA	DMKTRA				
DMKTRCND	000001	DMKUSO								
DMKTRCPB	000008	DMKCDS	DMKCFC	DMKCFP	DMKPRV	DMKPSA	DMKTRA			
DMKTRCPG	000002	DMKDSP	DMKPRG							
DMKTRCPV	000001	DMKPRV								
DMKTRCSI	000004	DMKIOS	DMKVCA	DMKVIO						
DMKTRCSV	000001	DMKPSA								
DMKTRCSW	000001	DMKVIO								
DMKTRCWT	000001	DMKVIO								
DMKTRMID	000002	DMKCNS	DMKSYM							
DMKUCBLD	000001	DMKCSO								
DMKUCSLD	000001	DMKCSO								
DMKUDRBV	000002	DMKCPI	DMKSYM							
DMKUDRDS	000002	DMKHVD	DMKSYM							
DMKUDRFD	000003	DMKLNK	DMKSYM							
DMKUDRFU	000012	DMKCSP	DMKCSU	DMKDEF	DMKHVD	DMKLNK	DMKLOG	DMKRSP	DMKSPL	DMKSYM
DMKUDRRD	000006	DMKDEF	DMKHVD	DMKLOG	DMKSPL	DMKSYM				
DMKUDRRV	000008	DMKDEF	DMKHVD	DMKLNK	DMKLOG	DMKSPL	DMKSYM			
DMKUNTFR	000008	DMKCFP	DMKGIO	DMKHVC	DMKSYM	DMKVIO				
DMKUNTIS	000003	DMKISM	DMKSYM							
DMKUNTRN	000003	DMKGIO	DMKSYM	DMKVIO						
DMKUNTRS	000002	DMKCCW	DMKSYM							
DMKUSODS	000002	DMKCFC	DMKSYM							
DMKUSOFF	000003	DMKDSP	DMKLOG	DMKSYM						
DMKUSOFL	000002	DMKCFC	DMKSYM							
DMKUSOFM	000001	DMKSYM								
DMKUSOLG	000003	DMKCFC	DMKSYM							
DMKVATAB	000007	DMKCDB	DMKCDS	DMKDSP	DMKPRV	DMKSYM				
DMKVATBC	000006	DMKCDS	DMKCFP	DMKCPB	DMKDSP	DMKSYM	DMKUSO			
DMKVATEX	000004	DMKDSP	DMKPRV	DMKSYM	DMKTMR					
DMKVATLA	000002	DMKPRV	DMKSYM							
DMKVATMD	000005	DMKCDS	DMKCFG	DMKCPB	DMKDSP	DMKSYM				
DMKVATPF	000001	DMKPRG								
DMKVATPX	000002	DMKPRG	DMKSYM							
DMKVATRN	000006	DMKPRV	DMKSYM	DMKTMR	DMKTRC	DMKVER				
DMKVATSX	000002	DMKPRG	DMKSYM							
DMKVCARD	000002	DMKCFP	DMKSYM							
DMKVCARS	000004	DMKDEF	DMKDIA	DMKSYM	DMKVDR					

Label	Count	References								
DMKVCASH	000002	DMKSYM	DMKVIO							
DMKVCAST	000002	DMKSYM	DMKVIO							
DMKVCATS	000002	DMKSYM	DMKVIO							
DMKVCHDC	000002	DMKSYM	DMKVDB							
DMKVCNEX	000002	DMKSYM	DMKVIO							
DMKVDBAT	000002	DMKCFC	DMKSYM							
DMKVDBDE	000002	DMKCFC	DMKSYM							
DMKVDREL	000007	DMKCFP	DMKLNK	DMKNLD	DMKSYM	DMKUSO	DMKVCH	DMKVDB		
DMKVDSAT	000005	DMKLOG	DMKSYM	DMKVCH	DMKVDB					
DMKVDSDF	000004	DMKDEF	DMKLOG	DMKSYM						
DMKVDSLK	000002	DMKLNK	DMKSYM							
DMKVERD	000002	DMKPSA	DMKSYM							
DMKVERO	000002	DMKPSA	DMKSYM							
DMKVIOCI	000001	DMKMON								
DMKVIOCT	000002	DMKMON	DMKSYM							
DMKVIOCW	000002	DMKMON	DMKSYM							
DMKVIOEX	000003	DMKHVC	DMKPRV	DMKSYM						
DMKVIOHD	000001	DMKMON								
DMKVIOHI	000001	DMKMON								
DMKVIOIN	000008	DMKCSP	DMKCSU	DMKDIA	DMKIOS	DMKSPL	DMKSYM	DMKVCA		
DMKVIOMK	000007	DMKCFM	DMKCFP	DMKCPB	DMKDSP	DMKVCN	DMKVSP			
DMKVIOSF	000001	DMKMON								
DMKVIOSI	000001	DMKMON								
DMKVIOTC	000001	DMKMON								
DMKVIOTI	000001	DMKMON								
DMKVMA	000001	DMKSYM								
DMKVMACF	000004	DMKCDS	DMKCFD	DMKTRC						
DMKVMAPS	000020	DMKCCW	DMKCDS	DMKDGD	DMKDSP	DMKPTR	DMKSYM	DMKUSO	DMKVCN	DMKVSP
DMKVMASH	000003	DMKDSP	DMKSYM	DMKUSO						
DMKVMAS1	000003	DMKCFG								
DMKVMAS2	000001	DMKCFG								
DMKVMI	000002	DMKCFG	DMKCPI							
DMKVSPCO	000007	DMKCFP	DMKCPV	DMKCSP	DMKSYM	DMKVDR				
DMKVSPCR	000005	DMKCFP	DMKCSP	DMKDRD	DMKSYM	DMKVDR				
DMKVSPEX	000002	DMKSYM	DMKVIO							
DMKVSPRT	000007	DMKCDB	DMKRNH	DMKSYM	DMKTRC					
DMKVSPTO	000001	DMKVIO								
DMKVSPVP	000003	DMKQCN	DMKSYM							
DMKVSPWA	000003	DMKSYM	DMKUSO							
DMKWRM	000001	DMKLD00E								
DMKWRMST	000001	DMKCPI								
DMPCRS	000001	DMKEDM								

Label	Count	References											
DMPFLAG	000001	DMKDMP											
DMPFPRS	000002	DMKDMP	DMKEDM										
DMPGPRS	000002	DMKDMP	DMKEDM										
DMPINREC	000003	DMKDMP	DMKEDM										
DMPKEY	000002	DMKDMP											
DMPKYREC	000001	DMKDMP											
DMPLCORE	000002	DMKDMP	DMKEDM										
DMPPGMAP	000004	DMKDMP	DMKEDM										
DMPSYSRV	000002	DMKDMP	DMKEDM										
DMPTODCK	000002	DMKDMP	DMKEDM										
ECBLOK	000060	DMKBLD	DMKCDB	DMKCDS	DMKCFG	DMKCFP	DMKCFS	DMKDSP	DMKEDM	DMKPRG	DMKPRV	DMKSCH	DMKTMR
		DMKTRC	DMKUSO	DMKVAT									
ECSWLOG	000006	DMKCCH	DMKIOG										
EDIT	000024	DMKCFG	DMKCFM	DMKCFS	DMKCNS	DMKCPI	DMKEDM	DMKGRF	DMKLNK	DMKMSW	DMKNLD	DMKQCN	DMKRGA
		DMKRNH	DMKVCN										
EQCHK	000007	DMKRSE											
ERRBLOK	000009	DMKIOE	DMKIOF										
ERRCCNT	000003	DMKIOE	DMKIOF										
ERRCCW	000006	DMKIOE	DMKIOF										
ERRCONT	000001	DMKIOF											
ERRCORR	000003	DMKIOE	DMKIOF										
ERRHEADR	000002	DMKIOE											
ERRIOB	000012	DMKIOE	DMKIOF										
ERRIOER	000003	DMKIOE	DMKIOF										
ERRKEY	000004	DMKIOE	DMKIOF										
ERRMIOB	000003	DMKIOE	DMKIOF										
ERRMIOER	000002	DMKIOE	DMKIOF										
ERRMSG	000009	DMKCFG	DMKCQP	DMKDDR	DMKDIR	DMKDSP	DMKERM	DMKLNK	DMKNLD	DMKRNH			
ERROR	000070	DMKMCC	DMKMON										
ERRPARM	000003	DMKIOE	DMKIOF										
ERRSDR	000007	DMKIOE	DMKIOF										
ERRSIZE	000001	DMKIOE											
ERRVOLID	000003	DMKIOE	DMKIOF										
EXNPSW	000006	DMKCPI	DMKDSP	DMKSAV									
EXOPSW	000008	DMKDSP	DMKPSA										
EXTARCH	000007	DMKVAT											
EXTCCTRQ	000007	DMKBLD	DMKCDS	DMKCFP	DMKCFS	DMKTMR	DMKUSO						
EXTCOPY	000005	DMKVAT											
EXTCPTMR	000024	DMKCDS	DMKCFP	DMKSCH	DMKTMR								
EXTCPTRQ	000014	DMKBLD	DMKCFS	DMKSCH	DMKTMR	DMKUSO							
EXTCR0	000021	DMKBLD	DMKCDB	DMKCDS	DMKCFG	DMKCFP	DMKDSP	DMKPRV	DMKTRC	DMKVAT			
EXTCR1	000003	DMKVAT											

Label	Count	References											
EXTCR14	000002	DMKBLD	DMKCFP										
EXTCR15	000002	DMKBLD	DMKCFP										
EXTCR2	000006	DMKBLD	DMKCFP	DMKDSP									
EXTCR4	000003	DMKCFP	DMKDSP										
EXTCR7	000001	DMKDSP											
EXTCR9	000002	DMKPRV	DMKTMR										
EXTMASK	000002	DMKCFG	DMKTRC										
EXTMODE	000025	DMKCDS	DMKCFG	DMKCPB	DMKCPI	DMKDMP	DMKDSP	DMKPRV	DMKPSA	DMKTRC	DMKVER	DMKVMI	
EXTPERAD	000005	DMKDSP	DMKPRG	DMKPRV	DMKTMR								
EXTPERCD	000003	DMKDSP	DMKPRG										
EXTSEGLN	000005	DMKVAT											
EXTSHCR0	000011	DMKDSP	DMKPRV	DMKTMR	DMKTRC	DMKVAT							
EXTSHCR1	000011	DMKDSP	DMKVAT										
EXTSHLEN	000008	DMKVAT											
EXTSHSEG	000005	DMKVAT											
EXTSIZE	000010	DMKBLD	DMKCFS	DMKEDM	DMKUSO								
EXTSTOLD	000003	DMKVAT											
EXTVSEGS	000005	DMKVAT											
FAILADD	000002	DMKCCH											
FAILCCW	000007	DMKCCH											
FAILCSW	000011	DMKCCH											
FAILECSW	000004	DMKCCH											
FFS	000113	DMKBLD	DMKCCH	DMKCCW	DMKCDB	DMKCFC	DMKCFP	DMKCKS	DMKCNS	DMKCPI	DMKCQP	DMKCSP	DMKCST
		DMKCSU	DMKDEF	DMKDGD	DMKDIA	DMKDMP	DMKDRD	DMKDSP	DMKEIG	DMKHVD	DMKIOF	DMKLNK	DMKLOG
		DMKMCC	DMKMCH	DMKNES	DMKNLD	DMKOPR	DMKPGS	DMKPGT	DMKPRG	DMKPRV	DMKPSA	DMKPTR	DMKRPA
		DMKSCH	DMKSCN	DMKSEV	DMKSIX	DMKTRA	DMKTRC	DMKUNT	DMKUSO	DMKVCA	DMKVCH	DMKVDB	DMKVDR
		DMKVDS	DMKVSP	DMKWRM									
FREER0	000001	DMKFRE											
FREER1	000002	DMKFRE											
FREER14	000001	DMKFRE											
FREER15	000001	DMKFRE											
FREESAVE	000012	DMKFRE	DMKVCA										
FTREXTSN	000006	DMKCCW	DMKDAS	DMKIOE	DMKIOF								
FTRRPS	000006	DMKCPI	DMKDAS	DMKIOS	DMKVDB								
FTRRSRL	000001	DMKVDS											
FTRTYP1	000003	DMKNLD											
FTRUCS	000002	DMKCSO	DMKSSP										
FTR2311B	000005	DMKDIR	DMKLNK	DMKSCN	DMKVER								
FTR2311T	000005	DMKDIR	DMKLNK	DMKSCN	DMKVER								
FTR35MB	000010	DMKCKP	DMKCPI	DMKDAS	DMKHVD	DMKVDB	DMKVIO						
FTR70MB	000011	DMKCPI	DMKDAS	DMKPAG	DMKPGT	DMKSPL	DMKTDK	DMKVDB	DMKVER	DMKVIO			

Label	Count	References
F0	000024	DMKCFG DMKCPI DMKCQR DMKDSP DMKGRF DMKIOS DMKMON DMKNLD DMKPGS DMKPTR DMKRGA DMKSCH DMKSCN DMKSPL DMKUNT DMKVDB DMKVSP
F1	000194	DMKACO DMKBLD DMKBSC DMKCCW DMKCDB DMKCDS DMKCFC DMKCFD DMKCFG DMKCFS DMKCFT DMKCKS DMKCNS DMKCPI DMKCPV DMKCQG DMKCQP DMKCQR DMKCSO DMKCSP DMKCST DMKCSU DMKCVT DMKDAS DMKDGD DMKDIA DMKDRD DMKDSP DMKFRE DMKGRF DMKHVC DMKIOE DMKIOS DMKLNK DMKLOG DMKMCC DMKMON DMKMSG DMKNES DMKNLD DMKPAG DMKPGT DMKPRG DMKPSA DMKPTR DMKQCN DMKRGA DMKRGB DMKRNH DMKRPA DMKRSE DMKSAV DMKSCH DMKSIX DMKSNC DMKSPL DMKTAP DMKTHI DMKTRC DMKUDR DMKUNT DMKVCA DMKVCN DMKVDB DMKVDR DMKVER DMKVIO DMKVMA DMKVSP
F10	000022	DMKCCW DMKCKS DMKCNS DMKCPI DMKCVT DMKDAS DMKIOE DMKMSW DMKSCH DMKVDB
F15	000032	DMKBLD DMKCCW DMKCDB DMKCDS DMKCFG DMKCNS DMKDGD DMKIOF DMKLNK DMKPGS DMKPRV DMKPSA DMKSCH DMKTAP DMKTRC DMKUNT DMKUSO
F16	000021	DMKBLD DMKCCH DMKCCW DMKCDB DMKCDS DMKCNS DMKDGD DMKHVC DMKISM DMKPRV DMKRNH DMKTAP DMKTRC DMKUNT
F2	000082	DMKCCW DMKCDB DMKCDS DMKCFC DMKCFG DMKCFS DMKCFT DMKCNS DMKCPI DMKCPS DMKCPV DMKCQG DMKCQP DMKCQR DMKCSO DMKCSP DMKCST DMKCSU DMKDAS DMKERM DMKISM DMKLNK DMKMSG DMKPAG DMKPSA DMKQCN DMKRGA DMKSAV DMKSPL DMKTAP DMKTRC DMKVCA DMKVMA
F20	000001	DMKMSW
F24	000006	DMKCDB DMKCKS DMKCSU DMKRSP DMKVER
F240	000015	DMKCCW DMKCVT DMKDIA DMKLOG DMKPRV DMKPSA DMKTRC DMKUNT DMKVCA DMKVIO
F255	000018	DMKCFT DMKCKS DMKDRD DMKERM DMKGRF DMKIOE DMKIOF DMKMCH DMKNES DMKNET DMKRGA DMKTDK
F256	000039	DMKCFG DMKCNS DMKDAS DMKDRD DMKGRF DMKHVC DMKHVD DMKNLD DMKRGA DMKRGB DMKRNH DMKSNC DMKTDK DMKUDR DMKVCN DMKVER DMKWRM
F3	000095	DMKCCW DMKCDB DMKCFC DMKCFD DMKCFG DMKCFS DMKCKS DMKCPB DMKCPI DMKCPS DMKCPV DMKCQG DMKCQP DMKCSO DMKCSP DMKCST DMKCSU DMKDEF DMKDGD DMKDIA DMKGRF DMKHVD DMKMCC DMKMON DMKMSG DMKNES DMKNET DMKNLD DMKPAG DMKPGT DMKRGA DMKRSE DMKSAV DMKSCH DMKSPL DMKTAP DMKTHI DMKTRA DMKTRC DMKVCN DMKVDB
F4	000097	DMKACO DMKBLD DMKCCW DMKCDB DMKCDS DMKCFC DMKCFG DMKCFS DMKCKS DMKCNS DMKCPB DMKCPI DMKCSO DMKCSP DMKCSU DMKCVT DMKDEF DMKDGD DMKGRF DMKHVC DMKHVD DMKIOE DMKIOF DMKISM DMKMCC DMKMON DMKMSW DMKNES DMKNET DMKNLD DMKPAG DMKPGS DMKPGT DMKPRV DMKPTR DMKRGA DMKRGB DMKRNH DMKRPA DMKRSE DMKRSP DMKSAV DMKSCH DMKSPL DMKTAP DMKTHI DMKTMR DMKTRC DMKUNT DMKVAT DMKVCN DMKVDB DMKVER DMKVSP
F4095	000051	DMKBLD DMKCCW DMKCFG DMKCFT DMKCQR DMKDAS DMKDGD DMKDIA DMKDMP DMKHVC DMKHVD DMKLNK DMKLOG DMKMCC DMKMON DMKNES DMKNET DMKPSA DMKPTR DMKQCN DMKRGA DMKRGB DMKRNH DMKRSP DMKTMR DMKVER DMKVIO DMKVMA DMKVSP
F4096	000063	DMKCCW DMKCDB DMKCFG DMKCPI DMKCPV DMKDAS DMKDGD DMKDMP DMKDRD DMKFRE DMKHVD DMKNLD DMKPGS DMKPTR DMKRSP DMKSNC DMKSSP DMKUDR DMKVMA DMKVSP
F5	000020	DMKCDS DMKCFS DMKCPI DMKCSU DMKDEF DMKDGD DMKGRF DMKNLD DMKPAG DMKPRV DMKRGA DMKSCH DMKSCN DMKTAP DMKTMR
F6	000020	DMKCDB DMKCDS DMKCFC DMKCFD DMKCPB DMKCQP DMKCSU DMKDGD DMKMCH DMKMSW DMKPRV DMKTAP DMKVDB
F60	000031	DMKACO DMKCFC DMKCQR DMKCVT DMKDMP DMKHVC DMKHVD DMKMCC DMKNET DMKPRV DMKPSA DMKRNH DMKTHI DMKTMR DMKTRC

Label	Count	References											
F7	000045	DMKBLD	DMKBSC	DMKCCH	DMKCCW	DMKCFG	DMKCFS	DMKCPI	DMKCSP	DMKCSU	DMKIOE	DMKIOF	DMKIOG
		DMKLNK	DMKLOG	DMKPRV	DMKRSE	DMKSCN	DMKSEV	DMKSIX	DMKTMR	DMKTRM	DMKUNT	DMKVCN	DMKVDB
		DMKVER											
F8	000157	DMKACO	DMKBLD	DMKBSC	DMKCCH	DMKCCW	DMKCDS	DMKCFC	DMKCFG	DMKCFS	DMKCNS	DMKCPI	DMKCPV
		DMKCQG	DMKCQR	DMKCSO	DMKCSP	DMKCSU	DMKDAS	DMKDEF	DMKDGD	DMKDMP	DMKDRD	DMKGRF	DMKHVD
		DMKIOE	DMKISM	DMKLNK	DMKLOG	DMKMCC	DMKMCH	DMKMSW	DMKNET	DMKNLD	DMKPAG	DMKPGS	DMKPSA
		DMKPTR	DMKQCN	DMKRGA	DMKRNH	DMKRSE	DMKRSP	DMKSEV	DMKSIX	DMKTAP	DMKTHI	DMKTMR	DMKTRA
		DMKTRC	DMKUDR	DMKUSO	DMKVCN	DMKVDB	DMKVDS	DMKVER	DMKVIO	DMKVMA	DMKVSP	DMKWRM	
F9	000007	DMKCCW	DMKCPI	DMKCPV	DMKMSW	DMKUNT	DMKVDB						
HALFPAGE	000002	DMKDMP											
HARDSTCP	000002	DMKCPI	DMKDMP										
HIOCCH	000006	DMKCCH	DMKSEV	DMKSIX									
IDA	000057	DMKCCW	DMKDAS	DMKDGD	DMKDIA	DMKISM	DMKTAP	DMKTRC	DMKUNT	DMKVCA	DMKVCN	DMKVSP	
IDLEWAIT	000006	DMKCPI	DMKDSP	DMKMON	DMKSCH								
IFCC	000048	DMKBSC	DMKCCH	DMKCNS	DMKCPI	DMKDAS	DMKEIG	DMKGRF	DMKHVC	DMKIOE	DMKIOS	DMKMSW	DMKRSE
		DMKRSP	DMKSEV	DMKSIX	DMKTAP	DMKUNT							
IGBLAME	000027	DMKEIG	DMKSEV	DMKSIX									
IGPRGFLG	000009	DMKCCH	DMKSEV	DMKSIX									
IGTERMSQ	000048	DMKCCH	DMKEIG	DMKSEV	DMKSIX								
IGVALIDB	000029	DMKCCH	DMKEIG	DMKSEV	DMKSIX								
IL	000034	DMKCNS	DMKDIA	DMKGIO	DMKHVC	DMKIOS	DMKNLD	DMKPAG	DMKRNH	DMKRSP	DMKTAP	DMKVCA	DMKVCN
		DMKVIO	DMKVMI	DMKVSP									
INHIBIT	000032	DMKCNS	DMKGRF	DMKLNK	DMKLOG	DMKQCN	DMKRGA	DMKRGB	DMKRNH				
INTERCCH	000005	DMKCCH	DMKEIG	DMKSEV									
INTEX	000005	DMKDSP	DMKPSA										
INTEXF	000002	DMKDSP	DMKPSA										
INTKFLIN	000001	DMKEDM											
INTMASK	000001	DMKCPI											
INTMC	000001	DMKMCH											
INTPR	000017	DMKCKP	DMKPRG	DMKPRV									
INTPRL	000007	DMKDSP	DMKPRG	DMKPRV									
INTREQ	000037	DMKCNS	DMKCPI	DMKDDR	DMKDIA	DMKDMP	DMKGRF	DMKIOS	DMKMSW	DMKNLD	DMKRNH	DMKRSE	DMKSAV
		DMKVCA	DMKVCN	DMKVIO	DMKVSP								
INTSVC	000004	DMKPSA											
INTSVCL	000013	DMKPRG	DMKPSA	DMKTRC									
INTTIO	000014	DMKCCH	DMKCKP	DMKDMP	DMKDSP	DMKIOS	DMKSAV	DMKVMI					
IOBBPNT	000012	DMKDSP	DMKIOS	DMKNLD	DMKPAG	DMKSTK							
IOBCAW	000162	DMKACO	DMKBSC	DMKCCW	DMKCFP	DMKCNS	DMKCPB	DMKCPI	DMKCSO	DMKDAS	DMKDGD	DMKDIA	DMKGIO
		DMKGRF	DMKHVC	DMKIOS	DMKISM	DMKMCC	DMKMON	DMKNLD	DMKPAG	DMKRGA	DMKRGB	DMKRNH	DMKRSE
		DMKRSP	DMKSEP	DMKSPL	DMKTAP	DMKTRC	DMKUDR	DMKUNT	DMKVCA	DMKVDB	DMKVDR	DMKVIO	
IOBCCH	000004	DMKCCH											
IOBCC1	000015	DMKCNS	DMKDGD	DMKDIA	DMKIOS	DMKNLD	DMKRNH	DMKRSE	DMKRSP	DMKVCA			

Label	Count	References											
IOBCC2	000002	DMKIOS	DMKVIO										
IOBCC3	000029	DMKCNS	DMKCPS	DMKDAS	DMKDGD	DMKGIO	DMKIOS	DMKNLD	DMKRGA	DMKRNH	DMKRSE	DMKUNT	DMKVCA
		DMKVDB	DMKVIO										
IOBCOPY	000006	DMKGRF											
IOBCP	000039	DMKACO	DMKCCH	DMKCPS	DMKCSO	DMKDAS	DMKDIA	DMKIOE	DMKIOS	DMKNLD	DMKPAG	DMKRGA	DMKRGB
		DMKRNH	DMKRSP	DMKSPL	DMKTAP	DMKTDK	DMKUDR	DMKVDB					
IOBCSW	000249	DMKACO	DMKCCH	DMKCNS	DMKCPS	DMKCSO	DMKCSP	DMKCSU	DMKDAS	DMKDGD	DMKDIA	DMKDSP	DMKGIO
		DMKGRF	DMKHVC	DMKIOS	DMKMON	DMKNLD	DMKPAG	DMKRGA	DMKRNH	DMKRSE	DMKRSP	DMKSEP	DMKSPL
		DMKTAP	DMKTRC	DMKUNT	DMKVCA	DMKVIO							
IOBCYL	000024	DMKCCW	DMKDGD	DMKIOS	DMKMON	DMKPAG	DMKPGT	DMKSPL	DMKTDK				
IOBERP	000026	DMKBSC	DMKCNS	DMKDAS	DMKGRF	DMKIOS	DMKRSE	DMKRSP	DMKTAP				
IOBFATAL	000058	DMKACO	DMKBSC	DMKCNS	DMKCSO	DMKDAS	DMKDGD	DMKGIO	DMKGRF	DMKIOE	DMKIOF	DMKIOS	DMKMON
		DMKNLD	DMKPAG	DMKRGA	DMKRSE	DMKRSP	DMKSEP	DMKSPL	DMKTAP	DMKUDR	DMKVDB	DMKVIO	
IOBFLAG	000142	DMKACO	DMKBSC	DMKCCH	DMKCCW	DMKCFP	DMKCNS	DMKCPS	DMKCSO	DMKDAS	DMKDGD	DMKDIA	DMKDSP
		DMKGIO	DMKGRF	DMKIOE	DMKIOS	DMKMON	DMKNLD	DMKPAG	DMKRGA	DMKRGB	DMKRNH	DMKRSE	DMKRSP
		DMKSEP	DMKSPL	DMKTAP	DMKTDK	DMKUDR	DMKUNT	DMKVCA	DMKVDB	DMKVDR	DMKVIO		
IOBFPNT	000023	DMKDSP	DMKEDM	DMKIOS	DMKNLD	DMKPAG	DMKPGT	DMKSTK					
IOBHIO	000018	DMKCCH	DMKCFP	DMKCPS	DMKIOS	DMKVIO							
IOBHVC	000011	DMKCFP	DMKCPS	DMKDGD	DMKGIC	DMKIOE	DMKIOS						
IOBIOER	000115	DMKBSC	DMKCCH	DMKCFP	DMKCNS	DMKCPS	DMKDAS	DMKDGD	DMKDIA	DMKGIO	DMKGRF	DMKIOE	DMKIOS
		DMKMON	DMKNLD	DMKRGA	DMKRGB	DMKRNH	DMKRSE	DMKRSP	DMKTAP	DMKVCA	DMKVIO		
IOBIRA	000054	DMKACO	DMKCFP	DMKCNS	DMKCPB	DMKCPI	DMKCPS	DMKCSO	DMKCSP	DMKCSU	DMKDAS	DMKDGD	DMKDIA
		DMKDSP	DMKGIO	DMKGRF	DMKIOS	DMKISM	DMKMON	DMKNLD	DMKPAG	DMKRGA	DMKRGB	DMKRNH	DMKRSP
		DMKSEP	DMKSPL	DMKUDR	DMKUNT	DMKVCA	DMKVDB	DMKVDR	DMKVIO				
IOBLINK	000032	DMKACO	DMKCFP	DMKCNS	DMKCPS	DMKCSO	DMKCSP	DMKCSU	DMKDAS	DMKDGD	DMKDIA	DMKGIO	DMKGRF
		DMKIOS	DMKRGA	DMKRNH	DMKSEP	DMKSPL	DMKVCA	DMKVIO					
IOBLOK	000245	DMKACO	DMKBSC	DMKCCH	DMKCCW	DMKCFP	DMKCNS	DMKCPB	DMKCPI	DMKCPS	DMKCSO	DMKCSP	DMKCSU
		DMKDAS	DMKDGD	DMKDIA	DMKDSP	DMKEDM	DMKGIO	DMKGRF	DMKHVC	DMKIOE	DMKIOS	DMKISM	DMKLOG
		DMKMCC	DMKMON	DMKMSW	DMKNLD	DMKPAG	DMKPGT	DMKRGA	DMKRGB	DMKRNH	DMKRSE	DMKRSP	DMKSEP
		DMKSPL	DMKSTK	DMKTAP	DMKTDK	DMKTRC	DMKUDR	DMKUNT	DMKVCA	DMKVDB	DMKVDR	DMKVIO	
IOBMISC	000097	DMKACO	DMKCCW	DMKCFP	DMKCPB	DMKCPS	DMKCSO	DMKDGD	DMKDIA	DMKGIO	DMKGRF	DMKISM	DMKMCC
		DMKMON	DMKNLD	DMKPAG	DMKRGA	DMKRGB	DMKRNH	DMKRSP	DMKSEP	DMKTDK	DMKUDR	DMKUNT	DMKVDB
		DMKVIO											
IOBMISC2	000061	DMKACO	DMKCCW	DMKCFP	DMKCPS	DMKCSO	DMKDGD	DMKGIO	DMKMON	DMKNLD	DMKRGA	DMKRGB	DMKRNH
		DMKRSE	DMKSEP	DMKSPL	DMKUDR	DMKVDB	DMKVIO						
IOBPAG	000005	DMKDSP	DMKIOS	DMKPAG									
IOBRADD	000054	DMKACO	DMKCCH	DMKCNS	DMKCPS	DMKCSO	DMKDAS	DMKDIA	DMKGRF	DMKHVC	DMKIOE	DMKIOS	DMKMSW
		DMKNLD	DMKPAG	DMKRGA	DMKRNH	DMKRSP	DMKSPL	DMKTRC	DMKVIO				
IOBRCAW	000058	DMKBSC	DMKDAS	DMKDIA	DMKIOS	DMKNLD	DMKRNH	DMKRSE	DMKRSP	DMKTAP	DMKVIO		
IOBRCNT	000087	DMKBSC	DMKDAS	DMKGRF	DMKNLD	DMKRGA	DMKRGB	DMKRNH	DMKRSE	DMKRSP	DMKTAP		
IOBRELCU	000013	DMKCCW	DMKIOS	DMKVDR	DMKVIO								
IOBRES	000007	DMKCFP	DMKCNS	DMKIOS	DMKNLD	DMKUNT	DMKVCA						

Label	Count	References											
IOBRSTRT	000059	DMKBSC	DMKCSO	DMKDAS	DMKDIA	DMKIOS	DMKNLD	DMKRGA	DMKRGB	DMKRNH	DMKRSE	DMKRSP	DMKSEP
		DMKTAP	DMKVCA										
IOBSENS	000004	DMKGRF											
IOBSIOF	000003	DMKIOS	DMKVIO										
IOBSIZE	000148	DMKACO	DMKCCW	DMKCFP	DMKCNS	DMKCPB	DMKCPI	DMKCPS	DMKCSO	DMKCSP	DMKCSU	DMKDAS	DMKDGD
		DMKDIA	DMKDMP	DMKEDM	DMKGIO	DMKGRF	DMKHVC	DMKIOS	DMKMCC	DMKMON	DMKNLD	DMKPAG	DMKRGA
		DMKRGB	DMKRNH	DMKRSP	DMKSEP	DMKSPL	DMKUDR	DMKVCA	DMKVDB	DMKVDR	DMKVIO		
IOBSNSIO	000008	DMKIOS											
IOBSPEC	000068	DMKACO	DMKCCH	DMKCFP	DMKCNS	DMKCPS	DMKDAS	DMKGRF	DMKIOS	DMKNLD	DMKRGA	DMKRGB	DMKRNH
		DMKVDB	DMKVIO										
IOBSPLT	000010	DMKIOS											
IOBSTAT	000158	DMKACO	DMKBSC	DMKCCW	DMKCNS	DMKCPS	DMKCSO	DMKDAS	DMKDGD	DMKDIA	DMKGIO	DMKGRF	DMKIOE
		DMKIOS	DMKMON	DMKNLD	DMKPAG	DMKRGA	DMKRGB	DMKRNH	DMKRSE	DMKRSP	DMKSEP	DMKSPL	DMKTAP
		DMKTRC	DMKUDR	DMKUNT	DMKVCA	DMKVDB	DMKVIO						
IOBTIO	000032	DMKCCH	DMKCFP	DMKCPS	DMKDAS	DMKIOS	DMKNLD	DMKVDB	DMKVIO				
IOBUC	000008	DMKIOS											
IOBUNSL	000013	DMKCNS	DMKCPS	DMKGRF	DMKIOS	DMKRGA	DMKRNH	DMKVIO					
IOBUSER	000056	DMKACO	DMKCCH	DMKCFP	DMKCNS	DMKCPB	DMKCPI	DMKCPS	DMKCSO	DMKCSP	DMKCSU	DMKDAS	DMKDIA
		DMKDSP	DMKGRF	DMKIOE	DMKIOS	DMKLOG	DMKMCC	DMKNLD	DMKPAG	DMKRGA	DMKRGB	DMKRNH	DMKSPL
		DMKUDR	DMKVCA	DMKVDB	DMKVDR	DMKVIO							
IOBVADD	000017	DMKCPS	DMKCSP	DMKCSU	DMKDIA	DMKIOS	DMKSPL	DMKTRC	DMKVCA	DMKVIO			
IOBWRAP	000003	DMKCCW	DMKVIO										
IOELPNTR	000005	DMKCCH	DMKEIG	DMKIOG									
IOERACT	000005	DMKDAS	DMKMSW	DMKRSE	DMKTAP								
IOERADR	000020	DMKDAS	DMKIOE	DMKIOF	DMKMSW	DMKTAP							
IOERBLOK	000147	DMKBSC	DMKCCH	DMKCCW	DMKCFP	DMKCNS	DMKCPS	DMKDAS	DMKDGD	DMKDIA	DMKEDM	DMKEIG	DMKGIO
		DMKGRF	DMKIOE	DMKIOF	DMKIOS	DMKMSW	DMKNLD	DMKRGA	DMKRGB	DMKRNH	DMKRSE	DMKRSP	DMKSEV
		DMKSIX	DMKTAP	DMKUNT	DMKVCA	DMKVIO							
IOERBSR	000014	DMKTAP											
IOERCAL	000003	DMKDAS											
IOERCAN	000021	DMKBSC	DMKDAS	DMKTAP									
IOERCCW	000008	DMKDIA	DMKIOS	DMKVCA									
IOERCEMD	000013	DMKDAS	DMKIOE	DMKRSE									
IOERCLN	000005	DMKTAP											
IOERCNCL	000002	DMKMSW											
IOERCSW	000072	DMKBSC	DMKCCH	DMKDAS	DMKDIA	DMKGIO	DMKGRF	DMKIOE	DMKIOF	DMKIOS	DMKMSW	DMKRSE	DMKRSP
		DMKTAP	DMKVCA	DMKVIO									
IOERCYLR	000002	DMKUNT											
IOERDASD	000005	DMKDAS	DMKMSW										
IOERDATA	000203	DMKBSC	DMKCCH	DMKCCW	DMKCNS	DMKCPS	DMKDAS	DMKDIA	DMKGIO	DMKGRF	DMKIOE	DMKIOF	DMKIOS
		DMKMSW	DMKNLD	DMKRNH	DMKRSE	DMKRSP	DMKTAP	DMKUNT	DMKVCA	DMKVIO			
IOERDEC	000007	DMKDAS	DMKMSW										

Label	Count	References
IOERDEPD	000003	DMKRSE DMKRSP
IOERDERD	000004	DMKRSE DMKRSP
IOERDW	000015	DMKBSC DMKDAS DMKTAP
IOERECF	000003	DMKDAS
IOERECSW	000004	DMKCCH DMKRSE
IOERERG	000004	DMKTAP
IOERERP	000004	DMKRSE DMKRSP
IOERETN	000006	DMKNLD DMKPTR DMKRPA DMKSNC
IOERETRY	000008	DMKDAS DMKMSW DMKRSE
IOEREXT	000040	DMKBSC DMKCCH DMKCCW DMKCFP DMKCNS DMKCPS DMKDAS DMKDGD DMKDIA DMKGIO DMKGRF DMKIOE
		DMKIOF DMKIOS DMKNLD DMKRGA DMKRGB DMKRNH DMKRSE DMKRSP DMKTAP DMKVIO
IOERFLG1	000062	DMKDAS DMKMSW DMKRSE DMKRSP DMKTAP
IOERFLG2	000055	DMKBSC DMKDAS DMKIOE DMKRSE DMKTAP DMKUNT
IOERFLG3	000017	DMKBSC DMKCNS DMKDAS DMKGRF DMKIOF DMKRSE DMKTAP
IOERFSR	000010	DMKTAP
IOERHA	000004	DMKDAS
IOERIGN	000003	DMKMSW DMKRSE
IOERIGNR	000004	DMKDAS DMKMSW DMKTAP
IOERIND3	000042	DMKBSC DMKDAS DMKMSW DMKRSE DMKTAP
IOERIND4	000010	DMKDAS DMKMSW DMKTAP
IOERINFO	000020	DMKBSC DMKDAS DMKMSW DMKRSE DMKTAP
IOERLEN	000017	DMKCCW DMKDAS DMKIOE DMKIOF DMKIOS DMKMSW DMKUNT DMKVCA
IOERLOC	000026	DMKBSC DMKDAS DMKTAP
IOERMSG	000003	DMKDAS DMKTAP
IOERMSW	000009	DMKBSC DMKDAS DMKTAP
IOERNUM	000071	DMKBSC DMKCNS DMKDAS DMKGRF DMKMSW DMKRSE DMKTAP
IOERORA	000012	DMKTAP
IOEROVFL	000008	DMKBSC DMKCNS DMKDAS DMKGRF DMKIOF DMKRSE DMKTAP
IOERPEND	000013	DMKDAS DMKMSW DMKRSE DMKTAP
IOERPNT	000018	DMKDAS DMKIOE DMKIOF DMKRSE
IOERRBK	000012	DMKTAP
IOERREAD	000015	DMKBSC DMKCNS DMKDAS DMKGRF DMKIOF DMKRSE DMKTAP
IOERREW	000002	DMKTAP
IOERSIZE	000056	DMKBSC DMKCCH DMKCCW DMKCFP DMKCNS DMKCPS DMKDAS DMKDGD DMKDIA DMKEDM DMKGIO DMKGRF
		DMKIOE DMKIOS DMKMON DMKNLD DMKRGA DMKRGB DMKRNH DMKRSE DMKRSP DMKTAP DMKVCA DMKVIO
IOERSTAT	000008	DMKDAS
IOERSTRT	000003	DMKDAS DMKMSW DMKTAP
IOERSUPP	000006	DMKTAP
IOERVLD	000002	DMKTAP
IOERVOL1	000003	DMKDAS
IOERVSER	000009	DMKDAS DMKIOE DMKIOF
IOERWRK	000011	DMKTAP

Label	Count	References											
IOERXERP	000003	DMKRSE											
IOMASK	000003	DMKCFM	DMKDMP	DMKTRC									
IONPSW	000017	DMKCKP	DMKDMP	DMKDSP	DMKFMT	DMKSAV	DMKSSP						
IONTWAIT	000006	DMKCPI	DMKDSP	DMKMON	DMKSCH								
IOOPSW	000024	DMKCCH	DMKCKP	DMKDSP	DMKFMT	DMKIOS	DMKSAV	DMKSSP					
IPLCCW1	000009	DMKCPI	DMKDMP	DMKVMI									
IPLPSW	000012	DMKCKP	DMKCPI	DMKDMP	DMKMON	DMKVMI							
IPLREQ	000003	DMKNLD	DMKRNH										
IPUADDR	000002	DMKCPI	DMKHVD										
IRMAND	000003	DMKCFS	DMKIOE										
IRMBIT1	000002	DMKCFS	DMKIOE										
IRMBIT2	000002	DMKCFS	DMKIOE										
IRMBLOK	000004	DMKCFS	DMKIOE										
IRMBYT1	000002	DMKCFS	DMKIOE										
IRMBYT2	000002	DMKCFS	DMKIOE										
IRMFLG	000006	DMKCFS	DMKIOE										
IRMLMT	000003	DMKCFS	DMKIOE										
IRMLMTCT	000003	DMKIOE											
IRMMAXCT	000005	DMKIOE											
IRMOR	000003	DMKCFS	DMKIOE										
IRMRLADD	000003	DMKCFS	DMKIOE										
IRMSIZE	000005	DMKCFS	DMKIOE										
KEEPSEGS	000011	DMKBLD	DMKCFG	DMKCFP	DMKPGS								
KEYMASK	000001	DMKCPI											
LASTUSER	000007	DMKDSP	DMKUSO	DMKVMA									
LOCK	000038	DMKCCW	DMKCFC	DMKCFG	DMKCKS	DMKCPV	DMKCSO	DMKDGD	DMKIOF	DMKIOG	DMKMCC	DMKNLD	DMKPSA
		DMKPTR	DMKRPA	DMKRSP	DMKSEP	DMKSNC	DMKSPL	DMKVSP	DMKWRM				
LOCKBLOK	000004	DMKLOC											
LOCKNAME	000002	DMKLOC											
LOCKNEXT	000004	DMKLOC											
LOCKQUE	000004	DMKLOC											
LOCKSIZE	000002	DMKLOC											
LOGDROP	000012	DMKCNS	DMKGRF	DMKRGA	DMKRGB	DMKRNH	DMKUSO						
LOGHOLD	000011	DMKCNS	DMKDIA	DMKGRF	DMKRGA	DMKRGB	DMKRNH	DMKUSO					
MCCPUID	000001	DMKMCH											
MCDAMLEN	000001	DMKIOG											
MCFXDLCG	000015	DMKMCH											
MCHAREA	000003	DMKCCH	DMKCFS	DMKIOG									
MCHEK	000004	DMKCPI	DMKDMP	DMKMCH									
MCHFIX	000003	DMKIOG											
MCHMODEL	000014	DMKCCH	DMKIOG										
MCNPSW	000016	DMKCPI	DMKIOG	DMKMCH	DMKSAV	DMKSSP							

Label	Count	References			
MCOLDPW	000002	DMKMCH			
MCOPSW	000008	DMKMCH			
MCPROGID	000002	DMKMCH			
MCREC	000002	DMKMCH			
MCRECORD	000001	DMKMCH			
MCRECTYP	000001	DMKMCH			
MDRCUA1	000002	DMKVER			
MDRKEYN	000001	DMKVER			
MDRREC	000006	DMKVER			
MICBLOK	000007	DMKBLD	DMKCFS	DMKDSP	DMKLCG
MICCREG	000005	DMKCFS	DMKLOG		
MICPEND	000002	DMKDSP			
MICRSEG	000003	DMKBLD	DMKCFS	DMKLOG	
MICSIZE	000004	DMKCFS	DMKLOG	DMKUSO	
MICVIP	000002	DMKDSP			
MICVPSW	000002	DMKCFS	DMKLOG		
MICWORK	000002	DMKCFS	DMKLOG		
MIHCUA1	000002	DMKVER			
MIHKEYN	000001	DMKVER			
MIHREC	000002	DMKVER			
MIHVOL	000001	DMKVER			
MNBHDLEN	000002	DMKMCC	DMKMON		
MNCLDAST	000002	DMKMON			
MNCLINST	000004	DMKPRV			
MNCLPERF	000005	DMKMON			
MNCLRESP	000003	DMKQCN			
MNCLSCH	000003	DMKSCH			
MNCLSEEK	000001	DMKIOS			
MNCLSYS	000001	DMKMON			
MNCLUSER	000002	DMKMON			
MNCOAEL	000001	DMKSCH			
MNCOAQ	000001	DMKSCH			
MNCOBRD	000001	DMKQCN			
MNCOCYL	000001	DMKIOS			
MNCODA	000001	DMKMON			
MNCODAS	000001	DMKMON			
MNCODASH	000001	DMKMON			
MNCODQ	000001	DMKSCH			
MNCOERD	000001	DMKQCN			
MNCOSIM	000004	DMKPRV			
MNCOSUS	000001	DMKMON			
MNCOSYS	000002	DMKMON			

Label	Count	References
MNCOTH	000001	DMKMON
MNCOTT	000001	DMKMON
MNCOUSER	000002	DMKMON
MNCOWRIT	000001	DMKQCN
MNHCLASS	000001	DMKMON
MNHCODE	000001	DMKMON
MNHDR	000001	DMKMON
MNHDRLEN	000002	DMKMON
MNHRECSZ	000001	DMKMON
MNHTOD	000001	DMKMON
MN000	000002	DMKMON
MN000INT	000001	DMKMON
MN000LEN	000001	DMKMON
MN000PPA	000002	DMKMON
MN000PPC	000001	DMKMON
MN000PRB	000001	DMKMON
MN000PSI	000001	DMKMON
MN000Q1E	000002	DMKMON
MN000Q2E	000001	DMKMON
MN000WID	000001	DMKMON
MN000WIO	000001	DMKMON
MN000WPG	000001	DMKMON
MN097	000001	DMKMON
MN097CPU	000001	DMKMON
MN097CR8	000001	DMKMON
MN097DAT	000001	DMKMON
MN097LEN	000001	DMKMON
MN097LEV	000001	DMKMON
MN097TIM	000001	DMKMON
MN097UID	000001	DMKMON
MN098	000001	DMKMON
MN098LEN	000001	DMKMON
MN098UID	000001	DMKMON
MN099	000001	DMKMON
MN099CNT	000001	DMKMON
MN099LEN	000001	DMKMON
MN099TOD	000001	DMKMON
MN10X	000001	DMKMON
MN10XADD	000002	DMKMON
MN10XLEN	000001	DMKMON
MN10XUID	000001	DMKMON
MN10YCNT	000001	DMKMON

Label	Count	References
MN10YIO	000001	DMKMON
MN10YLEN	000001	DMKMON
MN2RSV1	000002	DMKMON
MN20X	000001	DMKMON
MN20XNPP	000001	DMKMON
MN20XQNM	000008	DMKMON
MN20XQ1E	000001	DMKMON
MN20XQ1N	000001	DMKMON
MN20XQ2E	000001	DMKMON
MN20XQ2N	000001	DMKMON
MN20XSWS	000001	DMKMON
MN20XUID	000001	DMKMON
MN20XWSS	000001	DMKMON
MN20YTTI	000001	DMKMON
MN20YVTI	000001	DMKMON
MN202APR	000001	DMKMON
MN202CRD	000001	DMKMON
MN202IOC	000001	DMKMON
MN202LEN	000001	DMKMON
MN202LIN	000001	DMKMON
MN202PGR	000001	DMKMON
MN202PNC	000001	DMKMON
MN202PRI	000001	DMKMON
MN202PST	000001	DMKMON
MN202REF	000001	DMKMON
MN202RES	000001	DMKMON
MN203LEN	000001	DMKMON
MN204LEN	000001	DMKMON
MN204PRI	000001	DMKMON
MN4RSV1	000001	DMKMON
MN400	000001	DMKMON
MN400CRD	000001	DMKMON
MN400INT	000001	DMKMON
MN400IOC	000001	DMKMON
MN400LEN	000001	DMKMON
MN400LIN	000001	DMKMON
MN400PDK	000001	DMKMON
MN400PDR	000001	DMKMON
MN400PGR	000001	DMKMON
MN400PGW	000001	DMKMON
MN400PNC	000001	DMKMON
MN400PST	000001	DMKMON

Label	Count	References
MN400QLV	000001	DMKMON
MN400RES	000001	DMKMON
MN400RST	000001	DMKMON
MN400TTI	000001	DMKMON
MN400UID	000001	DMKMON
MN400UPR	000001	DMKMON
MN400VTI	000001	DMKMON
MN400WSS	000001	DMKMON
MN500	000001	DMKMON
MN500INS	000001	DMKMON
MN500LEN	000001	DMKMON
MN500OVH	000001	DMKMON
MN500UID	000001	DMKMON
MN500VAD	000002	DMKMON
MN600ADD	000006	DMKMON
MN600CNT	000002	DMKMON
MN600DEV	000002	DMKMON
MN600DLN	000004	DMKMON
MN600HDR	000001	DMKMON
MN600HLN	000004	DMKMON
MN600MAX	000001	DMKMON
MN600NUM	000002	DMKMON
MN600SER	000002	DMKMON
MN600TY	000002	DMKMON
MN700	000001	DMKMON
MN700ADD	000001	DMKMON
MN700CCY	000001	DMKMON
MN700CYL	000001	DMKMON
MN700DIR	000002	DMKMON
MN700LEN	000001	DMKMON
MN700QCH	000001	DMKMON
MN700QCU	000001	DMKMON
MN700QDV	000001	DMKMON
MN700UID	000001	DMKMON
MN802CLN	000001	DMKMON
MN802CNT	000001	DMKMON
MN802CTR	000001	DMKMON
MN802DEV	000001	DMKMON
MN802DLN	000002	DMKMON
MN802NAU	000001	DMKMON
MN802NPP	000001	DMKMON
MN802NUM	000001	DMKMON

Label	Count	References						
MN802PGR	000001	DMKMON						
MN802PGW	000001	DMKMON						
MN802PRB	000001	DMKMON						
MN802WID	000001	DMKMON						
MN802WIO	000001	DMKMON						
MN802WPG	000001	DMKMON						
MODEFLAG	000008	DMKIOG						
MODEL135	000003	DMKIOG						
MODEL145	000004	DMKCCH	DMKIOG					
MODEL155	000003	DMKIOG						
MODEL158	000001	DMKIOG						
MODEL165	000005	DMKCCH	DMKIOG					
MODEL168	000001	DMKIOG						
MODEQUIT	000008	DMKIOG						
MODFLAG1	000011	DMKCFS						
MOD1RETY	000003	DMKCFS						
MONAIOB	000010	DMKCPS	DMKDMP	DMKMCC	DMKMON			
MONARDB	000006	DMKCPS	DMKDMP	DMKMCC	DMKMON			
MONATRB	000006	DMKMCC	DMKMON					
MONCLASS	000014	DMKMON	DMKPRG					
MONCLOCK	000004	DMKMON						
MONCODE	000016	DMKMON	DMKPRG					
MONCOM	000012	DMKCPS	DMKDMP	DMKMCC	DMKMON			
MONCTEB1	000002	DMKMCC	DMKMON					
MONDVLST	000009	DMKMCC	DMKMON					
MONDVNUM	000009	DMKMCC	DMKMON					
MONFLAG1	000033	DMKCPS	DMKMCC	DMKMON				
MONFLAG2	000005	DMKDMP	DMKMON					
MONNEXT	000008	DMKMCC	DMKMON					
MONSAVE	000003	DMKMON						
MONSIZE	000003	DMKMCC	DMKMON					
MONSUSCK	000002	DMKMON						
MONSUSCT	000012	DMKMON						
MONTIINT	000004	DMKMON						
MONUSER	000006	DMKCPS	DMKMCC	DMKMON				
NCPNAME	000002	DMKNLD	DMKSNC					
NCPPAGCT	000002	DMKNLD	DMKSNC					
NCPPNT	000002	DMKNLD	DMKSNC					
NCPSTART	000002	DMKNLD	DMKSNC					
NCPTBL	000003	DMKNLD	DMKSNC					
NCPVOL	000004	DMKNLD	DMKSNC					
NEWPAGES	000011	DMKBLD	DMKCFG	DMKCFP	DMKCPI	DMKDEF	DMKLOG	DMKPGS

Label	Count	References
NEWSEGS	000006	DMKBLD DMKCFP DMKCPI DMKDEF DMKLOG DMKPGS
NICALRM	000005	DMKRGA DMKRGB
NICAPL	000008	DMKCFT DMKCQR DMKRGA DMKRGB
NICATOF	000005	DMKCFT DMKCQR DMKRNH
NICATRB	000006	DMKRGA DMKRGB
NICATTN	000007	DMKRNH
NICBLOK	000037	DMKBLD DMKCFT DMKCKP DMKCPI DMKCQR DMKDIA DMKHVC DMKHVD DMKLOG DMKNES DMKNET DMKNLD DMKPSA DMKQCN DMKRGA DMKRGB DMKRNH DMKWRM
NICCARD	000005	DMKRGA
NICCIBM	000006	DMKBLD DMKDIA DMKNES DMKNET DMKNLD DMKRNH
NICCORD	000007	DMKRGA DMKRGB
NICCPNA	000006	DMKRGA
NICD[illegible]D	000001	DMKRNH
NICDIAG	000008	DMKRGA DMKRGB
NICDISA	000035	DMKCKP DMKCPI DMKDIA DMKNES DMKNET DMKRGA DMKRGB DMKRNH DMKWRM
NICDISB	000014	DMKCKP DMKNET DMKRGA DMKRGB DMKRNH
NICENAB	000024	DMKCKP DMKDIA DMKNES DMKNET DMKRGA DMKRNH DMKWRM
NICEPAD	000009	DMKDIA DMKNES DMKNET DMKNLD
NICEPMD	000016	DMKDIA DMKNES DMKNET DMKNLD DMKRNH
NICERLK	000004	DMKRNH
NICFLAG	000092	DMKCFT DMKCKP DMKCQR DMKDIA DMKLOG DMKNES DMKNET DMKNLD DMKRGA DMKRGB DMKRNH DMKWRM
NICFMT	000005	DMKRGA DMKRGB
NICGRAF	000003	DMKHVC DMKHVD DMKNET
NICHOLD	000006	DMKRGA DMKRGB
NICLBSC	000002	DMKNES DMKNET
NICLGRP	000006	DMKCKP DMKNET DMKWRM
NICLINE	000013	DMKCKP DMKDIA DMKNES DMKNET DMKRNH
NICLLEN	000006	DMKBLD DMKCFT DMKCQR DMKHVD DMKQCN
NICLTRC	000013	DMKDIA DMKNES DMKRNH
NICMORE	000006	DMKRGA DMKRGB
NICMTA	000002	DMKRNH
NICNAME	000018	DMKBLD DMKCPI DMKDIA DMKNET DMKNLD DMKPSA DMKRGA DMKRNH
NICNTRL	000027	DMKRGA DMKRGB DMKRNH
NICPOLL	000006	DMKRGA DMKRGB
NICPROCN	000006	DMKRGA DMKRGB
NICPSUP	000011	DMKCFT DMKCQR DMKLOG DMKNES DMKNLD DMKRNH
NICQPNT	000062	DMKDIA DMKNES DMKRGA DMKRGB DMKRNH
NICRCNT	000012	DMKRNH
NICREAD	000008	DMKRGA DMKRGB
NICRSPL	000007	DMKNET DMKRGA
NICRUNN	000011	DMKRGA DMKRGB
NICSELT	000005	DMKRGA DMKRGB

Label	Count	References											
NICSESN	000011	DMKDIA	DMKNES	DMKNET	DMKRNH								
NICSIO	000006	DMKRGA	DMKRGB										
NICSIZE	000062	DMKCFT	DMKCKP	DMKCPI	DMKCPS	DMKCQR	DMKDIA	DMKHVC	DMKHVD	DMKLOG	DMKNES	DMKNET	DMKNLD
		DMKPSA	DMKQCN	DMKRGA	DMKRGB	DMKRNH	DMKVDS	DMKWRM					
NICSTAT	000116	DMKCKP	DMKCPI	DMKDIA	DMKNES	DMKNET	DMKNLD	DMKRGA	DMKRGB	DMKRNH	DMKWRM		
NICSWEP	000007	DMKDIA	DMKNES	DMKNLD									
NICTABF	000003	DMKRGA											
NICTELE	000010	DMKDIA	DMKNET	DMKRNH									
NICTERM	000015	DMKBLD	DMKCKP	DMKNET	DMKNLD	DMKRNH	DMKWRM						
NICTMCD	000015	DMKCFT	DMKCQR	DMKRGA	DMKRGB								
NICTRQ	000006	DMKRGA	DMKRGB										
NICTYPE	000059	DMKBLD	DMKCKP	DMKDIA	DMKHVC	DMKHVD	DMKNES	DMKNET	DMKNLD	DMKRGA	DMKRNH	DMKWRM	
NICUSER	000051	DMKBLD	DMKDIA	DMKLOG	DMKNES	DMKNET	DMKNLD	DMKPSA	DMKRGA	DMKRGB	DMKRNH		
NIC3275	000002	DMKRGA											
NOADD	000001	DMKUDR											
NOAUTO	000021	DMKCFM	DMKCFS	DMKCNS	DMKCPI	DMKNLD	DMKOPR	DMKQCN	DMKRNH	DMKVCN			
NOMODEL	000001	DMKIOG											
NORET	000171	DMKACO	DMKBLD	DMKCCH	DMKCDB	DMKCDS	DMKCFC	DMKCFD	DMKCFG	DMKCFM	DMKCFS	DMKCPB	DMKCPI
		DMKCPS	DMKCPV	DMKCQG	DMKCQP	DMKCQR	DMKCSO	DMKCSU	DMKDAS	DMKDEF	DMKDIA	DMKDSP	DMKERM
		DMKGRF	DMKIOE	DMKLNK	DMKLOG	DMKMCC	DMKMCH	DMKMID	DMKMSG	DMKMSW	DMKNES	DMKNET	DMKNLD
		DMKPGT	DMKPRG	DMKPSA	DMKPTR	DMKQCN	DMKRGA	DMKRNH	DMKRSP	DMKSPL	DMKTHI	DMKTRA	DMKTRC
		DMKUDR	DMKUSO	DMKVCA	DMKVCH	DMKVCN	DMKVDB	DMKVDR	DMKVER				
NOTIME	000030	DMKCFM	DMKCPI	DMKGRF	DMKMSG	DMKMSW	DMKQCN	DMKRGA	DMKVCN				
OBRCORL	000002	DMKIOE	DMKIOF										
OBRCPIDN	000003	DMKIOE	DMKIOF	DMKVER									
OBRCSWN	000002	DMKIOE	DMKIOF										
OBRCUA	000002	DMKVER											
OBRCUAIN	000004	DMKIOE	DMKIOF	DMKVER									
OBRCUAPR	000006	DMKIOE	DMKIOF	DMKVER									
OBRDDCNT	000009	DMKIOE	DMKIOF										
OBRDEVSH	000011	DMKIOC											
OBRDEVTN	000022	DMKIOC	DMKIOF										
OBRFCCWN	000002	DMKIOE	DMKIOF										
OBRHAN	000004	DMKIOE	DMKIOF	DMKVER									
OBRIORTY	000001	DMKIOF											
OBRKEYN	000011	DMKIOE	DMKIOF	DMKVER									
OBRLSKN	000004	DMKIOE	DMKIOF	DMKVER									
OBRPGMN	000003	DMKIOE	DMKIOF	DMKVER									
OBRRECN	000009	DMKIOC	DMKIOE	DMKIOF	DMKVER								
OBRSDRCT	000010	DMKIOE	DMKIOF										
OBRSDRSH	000001	DMKIOF											
OBRSENSN	000009	DMKIOE	DMKVER										

Label	Count	References
OBRSHOBR	000008	DMKIOC DMKIOF
OBRSNSCT	000002	DMKIOE DMKIOF
OBRSSDR1	000001	DMKIOF
OBRSWSN	000016	DMKIOC DMKIOE DMKIOF DMKVER
OBRTAPSN	000001	DMKIOE
OBRTEMP	000002	DMKIOF
OBRURSNS	000001	DMKIOE
OBRVOLN	000005	DMKIOE DMKIOF DMKVER
OBR3211S	000001	DMKIOE
OBR33SNS	000014	DMKIOE DMKIOF DMKVER
OBR3420S	000001	DMKIOE
OLDVMSEG	000010	DMKBLD DMKCFG DMKCFP DMKPGS
OPERATCR	000048	DMKCCH DMKCSO DMKDAS DMKDIA DMKERM DMKLOG DMKMCH DMKMSW DMKNLD DMKPGT DMKQCN DMKRNH
		DMKRSP DMKUDR DMKUSO DMKVCH DMKVDE DMKVDR DMKVER DMKWRM
OPNSFB	000006	DMKCKS DMKVSP
OWNDLIST	000020	DMKCKP DMKCKS DMKCPI DMKDRD DMKPAG DMKPGT DMKPTR DMKSPL DMKUDR DMKVDB DMKWRM
OWNDPREF	000002	DMKCPI DMKVDB
OWNDRDEV	000015	DMKCKP DMKCKS DMKCPI DMKDRD DMKPAG DMKPGT DMKPTR DMKSPL DMKUDR DMKVDB DMKWRM
OWNDVSER	000008	DMKCKS DMKCPI DMKUDR DMKVDB
PAGCORE	000059	DMKBLD DMKCDS DMKCFG DMKCPI DMKMCH DMKPGS DMKPTR DMKRPA DMKSCH DMKVMA
PAGECUR	000006	DMKMCC DMKMON
PAGELOAD	000004	DMKPAG DMKSCH
PAGEND	000004	DMKMCC DMKMON
PAGENXT	000005	DMKMCC DMKMON
PAGERATE	000001	DMKPAG
PAGEWAIT	000008	DMKCPI DMKCQR DMKDSP DMKMON DMKPAG DMKSCH
PAGE4K	000001	DMKCPI
PAGINVAL	000021	DMKBLD DMKCDS DMKMCH DMKPGS DMKPTR DMKRPA DMKSCH DMKUDR DMKVMA
PAGREF	000010	DMKPGS DMKPTR DMKRPA DMKSCH
PAGSHR	000002	DMKCFG DMKVMA
PAGSWP	000008	DMKBLD DMKCFG DMKVMA
PAGTABLE	000020	DMKBLD DMKCFG DMKVMA
PCHCHN	000004	DMKCKS DMKSPL
PCI	000025	DMKDSP DMKHVC DMKIOS DMKRNH DMKRSE DMKVCA DMKVCN DMKVIO DMKVSP
PCIF	000006	DMKCCW DMKDGD DMKVCA DMKVCN DMKVSP
PERADD	000004	DMKDSP DMKPRG
PERCODE	000004	DMKDSP DMKPRG
PERFCL	000004	DMKMCC DMKMON
PERGPRS	000005	DMKPRV
PERMODE	000003	DMKDSP DMKTRC
PERSALT	000008	DMKPRV DMKTMR
PGADDR	000002	DMKDSP DMKVAT

Label	Count	References											
PGBLOK	000003	DMKCFP	DMKDSP	DMKVAT									
PGBSIZE	000003	DMKCFP	DMKDSP	DMKVAT									
PGPNT	000003	DMKCFP	DMKDSP	DMKVAT									
PGREAD	000004	DMKMON	DMKPTR										
PGSRATIO	000001	DMKPAG											
PGWAITPG	000002	DMKPAG											
PGWRITE	000004	DMKMON	DMKPTR										
PRGC	000022	DMKBSC	DMKCNS	DMKDAS	DMKDIA	DMKGRF	DMKHVC	DMKIOS	DMKRNH	DMKRSE	DMKTAP	DMKUNT	DMKVCA
		DMKVCN	DMKVSP										
PRIORITY	000027	DMKACO	DMKCNS	DMKCPS	DMKDIA	DMKGRF	DMKMSG	DMKQCN	DMKRGB	DMKRNH	DMKUSO	DMKVCN	
PRNPSW	000023	DMKCKP	DMKCPI	DMKDMP	DMKDSP	DMKPRG	DMKSAV	DMKSSP					
PROBMODE	000011	DMKDSP	DMKMCH	DMKPRG	DMKPRV	DMKPSA							
PROBTIME	000006	DMKCPI	DMKMON	DMKSCH									
PROPSW	000041	DMKCKP	DMKCPI	DMKDSP	DMKFMT	DMKMON	DMKPRG	DMKPRV					
PRTC	000019	DMKBSC	DMKCNS	DMKDAS	DMKDIA	DMKGRF	DMKHVC	DMKIOS	DMKRNH	DMKRSE	DMKTAP	DMKUNT	DMKVCA
		DMKVCN	DMKVSP										
PRTCHN	000005	DMKCKS	DMKSPL	DMKVSP									
PSA	000159	DMKACO	DMKBLD	DMKBSC	DMKCCH	DMKCCW	DMKCDB	DMKCDS	DMKCFC	DMKCFD	DMKCFG	DMKCFM	DMKCFP
		DMKCFS	DMKCFT	DMKCKP	DMKCKS	DMKCNS	DMKCPB	DMKCPI	DMKCPS	DMKCPV	DMKCQG	DMKCQP	DMKCQR
		DMKCSO	DMKCSP	DMKCST	DMKCSU	DMKCVT	DMKDAS	DMKDEF	DMKDGD	DMKDIA	DMKDMP	DMKDRD	DMKDSP
		DMKEDM	DMKEIG	DMKERM	DMKFMT	DMKFRE	DMKGIO	DMKGRF	DMKHVC	DMKHVD	DMKIOC	DMKIOE	DMKIOF
		DMKIOG	DMKIOS	DMKISM	DMKLNK	DMKLOC	DMKLOG	DMKMCC	DMKMCH	DMKMID	DMKMON	DMKMSG	DMKMSW
		DMKNES	DMKNET	DMKNLD	DMKOPR	DMKPAG	DMKPGS	DMKPGT	DMKPRG	DMKPRV	DMKPTR	DMKQCN	DMKRGA
		DMKRGB	DMKRNH	DMKRPA	DMKRSE	DMKRSP	DMKSAV	DMKSCH	DMKSCN	DMKSEP	DMKSEV	DMKSIX	DMKSNC
		DMKSPL	DMKSSP	DMKSTK	DMKTAP	DMKTDK	DMKTHI	DMKTMR	DMKTRA	DMKTRC	DMKTRM	DMKUDR	DMKUNT
		DMKUSO	DMKVAT	DMKVCA	DMKVCH	DMKVCN	DMKVDB	DMKVDR	DMKVDS	DMKVER	DMKVIO	DMKVMA	DMKVMI
		DMKVSP	DMKWRM										
PSASVCCT	000003	DMKMON	DMKPSA										
PSENDCLR	000001	DMKCPI											
PSTARTSV	000004	DMKSAV											
QUANTUM	000004	DMKDSP											
QUANTUMR	000011	DMKDSP	DMKIOS	DMKMCH	DMKPRG	DMKPSA							
RCHADD	000011	DMKCCH	DMKCKP	DMKCPI	DMKEDM	DMKIOS	DMKMON	DMKSCN					
RCHBLOK	000027	DMKCCH	DMKCKP	DMKCPB	DMKCPI	DMKCPS	DMKCPV	DMKCQP	DMKDIA	DMKEDM	DMKIOG	DMKIOS	DMKMON
		DMKNES	DMKSCN	DMKSSP	DMKVCH								
RCHBMX	000002	DMKIOS											
RCHBUSY	000014	DMKIOS											
RCHCUTBL	000018	DMKCCH	DMKCKP	DMKCPI	DMKCPS	DMKCPV	DMKCQP	DMKDIA	DMKEDM	DMKMON	DMKNES	DMKSCN	DMKSSP
		DMKVCH											
RCHDISA	000006	DMKCPI	DMKCPS	DMKIOS	DMKVCH								
RCHFIOB	000007	DMKEDM	DMKIOS										
RCHMPX	000003	DMKIOS											
RCHQCNT	000007	DMKIOS	DMKMON										

Label	Count	References
RCHSEL	000001	DMKIOS
RCHSIZE	000003	DMKEDM DMKSSP
RCHSTAT	000021	DMKCPI DMKCPS DMKIOS DMKVCH
RCHTYPE	000009	DMKIOG DMKIOS
RCH370	000003	DMKIOG DMKIOS
RCUADD	000014	DMKCCH DMKCKP DMKCPI DMKEDM DMKIOS DMKMON DMKSCN DMKSSP
RCUBLOK	000034	DMKCCH DMKCKP DMKCPB DMKCPI DMKCPS DMKCPV DMKCQP DMKDIA DMKEDM DMKGRF DMKIOC DMKIOF
		DMKIOS DMKMON DMKNES DMKNLD DMKSCN DMKSSP DMKVCH
RCUBUSY	000012	DMKIOS
RCUCHA	000008	DMKCKP DMKCPB DMKIOS DMKMON DMKSCN DMKSSP
RCUCHB	000001	DMKSSP
RCUDISA	000009	DMKCPI DMKCPS DMKIOS DMKNES DMKNLD DMKVCH
RCUDVTBL	000020	DMKCCH DMKCKP DMKCPI DMKCPS DMKCPV DMKCQP DMKDIA DMKEDM DMKGRF DMKMON DMKNES DMKNLD
		DMKSCN DMKSSP DMKVCH
RCUFIOB	000006	DMKEDM DMKIOS
RCUPRIME	000008	DMKCKP DMKCPB DMKCPI DMKIOS DMKMON DMKSCN
RCUQCNT	000007	DMKIOS DMKMON
RCUSCED	000005	DMKIOS
RCUSHRD	000006	DMKIOS
RCUSIZE	000004	DMKEDM DMKSSP
RCUSTAT	000027	DMKCPI DMKCPS DMKIOS DMKNES DMKNLD DMKVCH
RCUSUB	000008	DMKCKP DMKCPB DMKCPI DMKIOS DMKMON DMKSCN
RCUTYPE	000021	DMKCKP DMKCPB DMKCPI DMKIOC DMKIOF DMKIOS DMKMON DMKSCN DMKSSP
RCU2701	000003	DMKIOC DMKIOF
RCU2702	000003	DMKIOC DMKIOF
RCWADDR	000049	DMKCCW DMKDGD DMKDIA DMKHVC DMKUNT DMKVCA
RCWCCNT	000009	DMKCCW DMKISM DMKUNT
RCWCCW	000026	DMKCCW DMKDGD DMKDIA DMKHVC DMKISM DMKTRC DMKUNT DMKVCA
RCWCNT	000013	DMKCCW DMKDGD DMKDIA DMKUNT DMKVCA
RCWCOMND	000053	DMKCCW DMKDGD DMKDIA DMKUNT DMKVCA
RCWCTL	000041	DMKCCW DMKDGD DMKDIA DMKHVC DMKUNT DMKVCA
RCWFLAG	000081	DMKCCW DMKDGD DMKDIA DMKUNT DMKVCA
RCWGEN	000005	DMKCCW DMKTRC DMKUNT
RCWHEAD	000005	DMKCCW
RCWHMR	000006	DMKCCW DMKUNT
RCWINVL	000008	DMKCCW DMKDIA DMKTRC DMKVCA
RCWIO	000013	DMKCCW DMKDGD DMKISM DMKUNT
RCWISAM	000001	DMKCCW
RCWPNT	000014	DMKCCW DMKHVC DMKISM DMKTRC DMKUNT
RCWRCNT	000007	DMKCCW DMKISM DMKTRC DMKUNT
RCWREL	000008	DMKCCW
RCWSHR	000007	DMKCCW DMKDGD DMKUNT

Label	Count	References									
RCWTASK	000025	DMKCCW	DMKHVC	DMKISM	DMKTRC	DMKUNT					
RCWVCAW	000010	DMKCCW	DMKISM	DMKTRC	DMKUNT						
RCWVCNT	000004	DMKCCW	DMKTRC								
RCW2311	000005	DMKCCW	DMKUNT								
RDBUFLN	000004	DMKRNH									
RDBUFNC	000003	DMKRNH									
RDEVACNT	000015	DMKACO	DMKCKP	DMKCQP	DMKCSO	DMKRSE	DMKRSP	DMKSPL			
RDEVACTV	000022	DMKCNS	DMKDIA	DMKGRF							
RDEVADD	000026	DMKCCH	DMKCKP	DMKCPI	DMKCPS	DMKCQP	DMKDIA	DMKEDM	DMKIOS	DMKLOG	DMKMON	DMKNES	DMKNLD
		DMKSCN	DMKSSP	DMKVDB	DMKVDR						
RDEVAIOB	000027	DMKCCH	DMKCFP	DMKCKP	DMKCPI	DMKCPS	DMKCSO	DMKDIA	DMKEDM	DMKGRF	DMKIOS	DMKLOG	DMKNLD
		DMKVIO									
RDEVAIRA	000008	DMKDIA	DMKGRF								
RDEVALLN	000018	DMKCKS	DMKCPI	DMKMON	DMKPGT	DMKTDK	DMKVDB	DMKWRM			
RDEVAPLP	000009	DMKCFT	DMKCQR	DMKGRF	DMKQCN						
RDEVATNC	000004	DMKCNS									
RDEVATOF	000008	DMKCFT	DMKCNS	DMKCPI	DMKCQR	DMKTRM					
RDEVATT	000012	DMKCFP	DMKCQP	DMKDEF	DMKDIA	DMKIOS	DMKNLD	DMKVCH	DMKVDB	DMKVDR	DMKVDS
RDEVAUTO	000007	DMKCKP	DMKCPI	DMKCQP	DMKNLD	DMKRNH	DMKWRM				
RDEVBACK	000008	DMKCSO	DMKRSE	DMKRSP							
RDEVBASE	000005	DMKDIA	DMKNES	DMKNLD	DMKPSA						
RDEVBLCK	000207	DMKACO	DMKBLD	DMKBSC	DMKCCH	DMKCCW	DMKCFC	DMKCFG	DMKCFM	DMKCFP	DMKCFS	DMKCFT	DMKCKP
		DMKCKS	DMKCNS	DMKCPB	DMKCPI	DMKCPS	DMKCPV	DMKCQG	DMKCQP	DMKCQR	DMKCSO	DMKDAS	DMKDEF
		DMKDIA	DMKDMP	DMKDRD	DMKEDM	DMKGRF	DMKHVC	DMKHVD	DMKIOC	DMKIOE	DMKIOF	DMKIOG	DMKIOS
		DMKLNK	DMKLOG	DMKMCC	DMKMON	DMKMSW	DMKNES	DMKNET	DMKNLD	DMKOPR	DMKPAG	DMKPGT	DMKPSA
		DMKPTR	DMKQCN	DMKRGA	DMKRGB	DMKRNH	DMKRSE	DMKRSP	DMKSCN	DMKSEP	DMKSNC	DMKSPL	DMKSSP
		DMKTAP	DMKTDK	DMKTRM	DMKUNT	DMKUSO	DMKVCH	DMKVCN	DMKVDB	DMKVDR	DMKVDS	DMKVER	DMKVIO
		DMKWRM									
RDEVBSC	000010	DMKBSC	DMKRGA	DMKRGB							
RDEVBUCH	000009	DMKIOS									
RDEVBUSY	000015	DMKACO	DMKCCH	DMKCPB	DMKCPS	DMKCSO	DMKIOS	DMKRNH	DMKRSP	DMKVCH	
RDEVCKPT	000007	DMKRNH	DMKWRM								
RDEVCLAS	000016	DMKACO	DMKCKP	DMKCKS	DMKCQP	DMKCSO	DMKMSW	DMKRSP	DMKSPL	DMKSSP	DMKWRM
RDEVCODE	000016	DMKCFG	DMKCKS	DMKCPI	DMKHVD	DMKIOG	DMKNLD	DMKPGT	DMKSNC	DMKVDB	DMKWRM
RDEVCON	000055	DMKCNS	DMKDIA	DMKEDM	DMKGRF	DMKNES	DMKQCN	DMKRGA	DMKRGB	DMKRNH	
RDEVCONC	000003	DMKIOS									
RDEVCORD	000015	DMKDIA	DMKGRF	DMKOPR							
RDEVCORR	000002	DMKCNS	DMKTRM								
RDEVCPNA	000005	DMKGRF									
RDEVCTL	000029	DMKCNS	DMKDIA	DMKGRF							
RDEVCTRS	000016	DMKCPS	DMKIOE	DMKIOF	DMKNES	DMKNET					
RDEVCUA	000016	DMKCKP	DMKCPB	DMKDIA	DMKGRF	DMKIOC	DMKIOF	DMKIOS	DMKMON	DMKNES	DMKNLD	DMKSCN	DMKSSP
RDEVCYL	000006	DMKDIA	DMKIOS	DMKMON	DMKPGT						

Label	Count	References											
RDEVDED	000057	DMKACO	DMKCFS	DMKCKP	DMKCPS	DMKCPV	DMKCQP	DMKCSO	DMKDAS	DMKDIA	DMKGRF	DMKIOS	DMKLOG
		DMKMCC	DMKMSW	DMKNES	DMKNET	DMKNLD	DMKRGB	DMKRNH	DMKRSP	DMKSCN	DMKSPL	DMKVCH	DMKVDB
		DMKVDR	DMKVDS	DMKVER									
RDEVDISA	000062	DMKACO	DMKCFS	DMKCKP	DMKCKS	DMKCNS	DMKCPI	DMKCPS	DMKCPV	DMKCQP	DMKCSO	DMKDAS	DMKGRF
		DMKIOS	DMKLOG	DMKMCC	DMKMON	DMKNES	DMKNET	DMKNLD	DMKRGA	DMKRGB	DMKRNH	DMKRSP	DMKSCN
		DMKSPL	DMKVCH	DMKVDB	DMKVDS	DMKWRM							
RDEVDISB	000025	DMKCKP	DMKCNS	DMKCPV	DMKGRF	DMKNES	DMKNET	DMKRGA	DMKRGB				
RDEVDRAN	000028	DMKACO	DMKCKP	DMKCKS	DMKCPS	DMKCQP	DMKCSO	DMKRSP	DMKSPL	DMKVCH	DMKVDS	DMKWRM	
RDEVENAB	000036	DMKCKP	DMKCNS	DMKCPI	DMKCPS	DMKCPV	DMKCQP	DMKGRF	DMKNES	DMKNET	DMKNLD	DMKRGA	DMKVCH
		DMKVDS	DMKWRM										
RDEVEPDV	000018	DMKDIA	DMKNES	DMKNLD									
RDEVEPLN	000005	DMKCPS	DMKDIA	DMKNES	DMKNLD	DMKVDS							
RDEVEPMD	000007	DMKCNS	DMKDIA	DMKNES	DMKNLD								
RDEVFICB	000005	DMKEDM	DMKIOS	DMKNLD	DMKPGT								
RDEVFLAG	000308	DMKACO	DMKBLD	DMKCFG	DMKCFS	DMKCFT	DMKCKP	DMKCKS	DMKCNS	DMKCPI	DMKCPS	DMKCPV	DMKCQP
		DMKCQR	DMKCSO	DMKDAS	DMKDIA	DMKEDM	DMKGRF	DMKIOS	DMKLOG	DMKMCC	DMKMON	DMKNES	DMKNET
		DMKNLD	DMKPGT	DMKPSA	DMKRGA	DMKRGB	DMKRNH	DMKRSE	DMKRSP	DMKSCN	DMKSEP	DMKSNC	DMKSPL
		DMKTRM	DMKVCH	DMKVDB	DMKVDR	DMKVDS	DMKWRM						
RDEVFTR	000041	DMKCCW	DMKCKP	DMKCPI	DMKCSO	DMKDAS	DMKHVD	DMKIOE	DMKIOF	DMKIOS	DMKNLD	DMKPAG	DMKPGT
		DMKSPL	DMKSSP	DMKTDK	DMKVDB	DMKVDS	DMKVER	DMKVIO					
RDEVHIO	000015	DMKCNS	DMKDIA	DMKGRF	DMKPSA								
RDEVHOLD	000005	DMKGRF											
RDEVIDNT	000008	DMKCNS	DMKCPI	DMKTRM									
RDEVIOCT	000007	DMKIOS	DMKMON										
RDEVIOER	000036	DMKBSC	DMKCFP	DMKCNS	DMKCPS	DMKCSO	DMKDAS	DMKEDM	DMKGRF	DMKIOE	DMKIOS	DMKMSW	DMKRSE
		DMKRSP	DMKTAP										
RDEVIRM	000007	DMKCFS	DMKIOE	DMKNES	DMKNLD								
RDEVLCEP	000010	DMKCKP	DMKCPS	DMKCQP	DMKDIA	DMKNLD	DMKRNH						
RDEVLIOB	000002	DMKIOS											
RDEVLLEN	000005	DMKBLD	DMKCFT	DMKCQR	DMKHVD	DMKQCN							
RDEVLNCP	000021	DMKCKP	DMKCPS	DMKCQP	DMKDIA	DMKNES	DMKNET	DMKNLD	DMKRNH	DMKVDS			
RDEVLNKS	000008	DMKCPS	DMKCQP	DMKSCN	DMKVDB	DMKVDR	DMKVDS						
RDEVLOAD	000005	DMKCSO	DMKRSP	DMKSEP									
RDEVLOG	000010	DMKCNS	DMKCPV	DMKGRF									
RDEVMAX	000028	DMKCKP	DMKCPI	DMKCPS	DMKNES	DMKNET	DMKNLD	DMKRGA	DMKRGB	DMKRNH	DMKVDS	DMKWRM	
RDEVMDL	000029	DMKCKP	DMKCPI	DMKHVD	DMKIOC	DMKIOE	DMKIOF	DMKNES	DMKNLD	DMKPAG	DMKUNT	DMKVDB	DMKVIO
RDEVMORE	000005	DMKGRF											
RDEVMOUT	000010	DMKCPS	DMKCQP	DMKDAS	DMKSCN	DMKVDB	DMKVDR	DMKVDS					
RDEVNCP	000010	DMKCKP	DMKCQP	DMKNLD	DMKRNH	DMKWRM							
RDEVNICL	000056	DMKCFT	DMKCKP	DMKCPI	DMKCPS	DMKCQR	DMKDIA	DMKHVC	DMKHVD	DMKLOG	DMKNES	DMKNET	DMKNLD
		DMKPSA	DMKQCN	DMKRGA	DMKRGB	DMKRNH	DMKVDS	DMKWRM					
RDEVNRDY	000044	DMKCNS	DMKCPI	DMKCPS	DMKCQP	DMKCSO	DMKDAS	DMKDIA	DMKNES	DMKNET	DMKNLD	DMKRGA	DMKRGB
		DMKRNH	DMKRSE	DMKRSP	DMKTAP								

Label	Count	References											
RDEVOWN	000019	DMKCFG	DMKCPI	DMKCPS	DMKCQP	DMKDAS	DMKEDM	DMKLOG	DMKNLD	DMKSNC	DMKVCH	DMKVDB	DMKVDS
RDEVPAGE	000009	DMKEDM	DMKPGT	DMKUSO									
RDEVPDLY	000003	DMKNES	DMKRGA										
RDEVPNT	000014	DMKCPI	DMKPGT	DMKTDK	DMKUSO	DMKVDB							
RDEVPREF	000010	DMKCPI	DMKMON	DMKPGT	DMKVDB								
RDEVPREP	000014	DMKCNS	DMKDIA										
RDEVPSUP	000015	DMKBLD	DMKCFT	DMKCNS	DMKCQR	DMKLOG							
RDEVPTTC	000006	DMKCNS	DMKCPI	DMKNES	DMKNLD	DMKTRM							
RDEVQCNT	000005	DMKIOS	DMKMON										
RDEVRACT	000011	DMKIOS											
RDEVRCNT	000011	DMKCNS											
RDEVRCVY	000014	DMKCPS	DMKDIA	DMKNES	DMKNLD	DMKRNH	DMKVCH	DMKVDS					
RDEVREAD	000007	DMKGRF											
RDEVRECS	000011	DMKCKP	DMKCKS	DMKDMP	DMKEDM	DMKPGT	DMKUSO	DMKWRM					
RDEVREST	000004	DMKCNS											
RDEVRSTR	000009	DMKCSO	DMKRSE	DMKRSP									
RDEVRSVD	000017	DMKCPS	DMKNES	DMKNET	DMKNLD	DMKRGA	DMKRGB	DMKRNH	DMKVCH	DMKVDS			
RDEVRUN	000011	DMKCPI	DMKDIA	DMKGRF									
RDEVSADN	000006	DMKCCW	DMKCNS	DMKIOC	DMKIOF	DMKNES							
RDEVSCED	000007	DMKCPS	DMKIOS	DMKRNH	DMKVCH								
RDEVSEP	000011	DMKCKP	DMKCQP	DMKCSO	DMKRSP	DMKSEP	DMKWRM						
RDEVSER	000038	DMKCFG	DMKCKS	DMKCPI	DMKCQG	DMKCQP	DMKDAS	DMKIOE	DMKLOG	DMKMON	DMKSCN	DMKVDB	DMKVER
		DMKWRM											
RDEVSIZE	000005	DMKEDM	DMKLOG	DMKSCN	DMKSSP								
RDEVSKUP	000004	DMKIOS	DMKMON										
RDEVSLCW	000009	DMKCQP	DMKNES	DMKRNH									
RDEVSPAC	000004	DMKCSO	DMKRSP										
RDEVSPL	000037	DMKACO	DMKCKP	DMKCKS	DMKCPS	DMKCQP	DMKCSO	DMKEDM	DMKRSE	DMKRSP	DMKSPL	DMKVCH	DMKVDS
		DMKWRM											
RDEVSTAT	000184	DMKACO	DMKCCH	DMKCFS	DMKCKP	DMKCKS	DMKCNS	DMKCPB	DMKCPI	DMKCPS	DMKCPV	DMKCQP	DMKCSO
		DMKDAS	DMKDIA	DMKGRF	DMKIOE	DMKIOS	DMKLOG	DMKMCC	DMKMON	DMKMSW	DMKNES	DMKNET	DMKNLD
		DMKRGA	DMKRGB	DMKRNH	DMKRSE	DMKRSP	DMKSCN	DMKSPL	DMKTAP	DMKVCH	DMKVDB	DMKVDR	DMKVDS
		DMKVER	DMKWRM										
RDEVSTA2	000022	DMKIOS	DMKNLD										
RDEVSYS	000022	DMKCFS	DMKCPI	DMKCPS	DMKCQP	DMKDAS	DMKLOG	DMKMCC	DMKMON	DMKVCH	DMKVDB	DMKVDR	DMKVDS
RDEVTBTU	000003	DMKNES	DMKRNH										
RDEVTCTL	000002	DMKNES											
RDEVTERM	000013	DMKCSO	DMKRSE	DMKRSP									
RDEVTFLG	000078	DMKCFT	DMKCNS	DMKCPI	DMKCPV	DMKCQR	DMKDIA	DMKGRF	DMKNLD	DMKTRM			
RDEVTMAT	000004	DMKACO	DMKDIA	DMKVDR	DMKVDS								
RDEVTMCD	000024	DMKCFT	DMKCNS	DMKCPI	DMKCQR	DMKGRF	DMKIOC	DMKIOF	DMKNES	DMKNLD	DMKQCN	DMKTRM	
RDEVTRQ	000004	DMKGRF											

Label	Count	References											
RDEVTYPC	000262	DMKBLD	DMKCFM	DMKCFS	DMKCFT	DMKCKP	DMKCNS	DMKCPI	DMKCPS	DMKCPV	DMKCQG	DMKCQP	DMKCQR
		DMKCSO	DMKDIA	DMKDMP	DMKEDM	DMKGRF	DMKHVC	DMKHVD	DMKIOC	DMKIOE	DMKIOF	DMKIOS	DMKLNK
		DMKLOG	DMKMCC	DMKMON	DMKNES	DMKNET	DMKNLD	DMKOPR	DMKPSA	DMKQCN	DMKRGA	DMKRNH	DMKRSE
		DMKRSP	DMKSCN	DMKSSP	DMKUSO	DMKVCH	DMKVCN	DMKVDB	DMKVDR	DMKVDS	DMKVER	DMKWRM	
RDEVTYPE	000376	DMKACO	DMKBLD	DMKCFC	DMKCFG	DMKCFM	DMKCFS	DMKCFT	DMKCKP	DMKCKS	DMKCNS	DMKCPI	DMKCPS
		DMKCPV	DMKCQG	DMKCQP	DMKCQR	DMKCSO	DMKDAS	DMKDIA	DMKDMP	DMKDRD	DMKGRF	DMKHVC	DMKHVD
		DMKIOC	DMKIOE	DMKIOF	DMKIOG	DMKIOS	DMKLNK	DMKLOG	DMKNES	DMKNLD	DMKOPR	DMKPAG	DMKPGT
		DMKPTR	DMKQCN	DMKRGA	DMKRSE	DMKRSP	DMKSEP	DMKSNC	DMKSPL	DMKSSP	DMKTAP	DMKTDK	DMKUSO
		DMKVCH	DMKVCN	DMKVDB	DMKVDS	DMKVER	DMKWRM						
RDEVUSC8	000002	DMKCNS	DMKNES										
RDEVUSER	000034	DMKBLD	DMKCFS	DMKCNS	DMKCPI	DMKCPS	DMKCPV	DMKCQP	DMKCSO	DMKDIA	DMKGRF	DMKIOS	DMKLOG
		DMKMCC	DMKNES	DMKNET	DMKNLD	DMKPSA	DMKVDB	DMKVDR	DMKVDS				
RDEVWAII	000011	DMKRGA	DMKRGB										
RDEVWAIT	000008	DMKNES	DMKRNH										
RDRCHN	000006	DMKCKS	DMKDMP	DMKNLD	DMKSPL								
READNRM	000002	DMKRNH											
RECBLOK	000055	DMKCKP	DMKCKS	DMKCPI	DMKDMP	DMKEDM	DMKPGT	DMKRSP	DMKSPL	DMKVSP	DMKWRM		
RECCCPD	000004	DMKIOF	DMKIOG										
RECCYL	000030	DMKCKP	DMKCKS	DMKCPI	DMKDMP	DMKPGT	DMKRSP	DMKVSP	DMKWRM				
RECFLAG1	000009	DMKIOF	DMKIOG										
RECFLAG2	000003	DMKIOG											
RECMAP	000037	DMKCKS	DMKCPI	DMKDMP	DMKPGT	DMKRSP	DMKVSP						
RECMAX	000012	DMKCKP	DMKCKS	DMKCPI	DMKPGT								
RECNXT	000010	DMKIOF	DMKIOG										
RECOVRPT	000005	DMKMCH											
RECPAG	000003	DMKIOF	DMKIOG										
RECPAGFL	000004	DMKIOF	DMKIOG										
RECPAGFM	000003	DMKIOG											
RECPAGFR	000002	DMKIOG											
RECPAGIU	000003	DMKIOF	DMKIOG										
RECPNT	000035	DMKCKP	DMKCKS	DMKCPI	DMKDMP	DMKEDM	DMKPGT	DMKRSP	DMKSPL	DMKVSP	DMKWRM		
RECSIZE	000022	DMKCKP	DMKCKS	DMKCPI	DMKEDM	DMKPGT	DMKRSP	DMKSPL	DMKUSO	DMKVSP	DMKWRM		
RECUSED	000024	DMKCKP	DMKCKS	DMKCPI	DMKDMP	DMKPGT	DMKRSP	DMKVSP					
RSPDPAGE	000014	DMKRSP	DMKSPL										
RSPLCTL	000014	DMKCKP	DMKCQP	DMKCSO	DMKEDM	DMKRSP	DMKSPL						
RSPMISC	000004	DMKCSO	DMKRSP										
RSPRPAGE	000015	DMKRSP	DMKSPL										
RSPRSTRT	000007	DMKRSP											
RSPSFBLK	000019	DMKCKP	DMKCQP	DMKCSO	DMKEDM	DMKRSP	DMKSPL						
RSPSIZE	000007	DMKEDM	DMKRSP	DMKSPL									
RSPVPAGE	000010	DMKRSP	DMKSPL										
RTCODE0	000002	DMKEIG											

Label	Count	References											
RTCODE1	000004	DMKEIG	DMKSEV	DMKSIX									
RTCODE2	000004	DMKEIG	DMKSEV	DMKSIX									
RTCODE3	000007	DMKEIG	DMKSEV	DMKSIX									
RTCODE4	000010	DMKEIG	DMKSEV	DMKSIX									
RTCODE5	000004	DMKEIG	DMKSEV	DMKSIX									
RTCODE7	000004	DMKSEV											
RUNCR0	000006	DMKCPI	DMKDSP	DMKPRV									
RUNCR1	000002	DMKCPI	DMKDSP										
RUNPSW	000015	DMKDSP	DMKPSA										
RUNUSER	000029	DMKCDS	DMKCPI	DMKDIA	DMKDSP	DMKIOS	DMKLOG	DMKMCH	DMKPRG	DMKPSA	DMKTHI	DMKUSO	DMKVCA
		DMKVMA											
R0	004816	DMKACO	DMKBLD	DMKBSC	DMKCCH	DMKCCW	DMKCDB	DMKCDS	DMKCFC	DMKCFD	DMKCFG	DMKCFM	DMKCFP
		DMKCFS	DMKCFT	DMKCKP	DMKCKS	DMKCNS	DMKCPB	DMKCPI	DMKCPS	DMKCPV	DMKCQG	DMKCQP	DMKCQR
		DMKCSO	DMKCSP	DMKCST	DMKCSU	DMKCVT	DMKDAS	DMKDDR	DMKDEF	DMKDGD	DMKDIA	DMKDIR	DMKDMP
		DMKDRD	DMKDSP	DMKEDM	DMKEIG	DMKERM	DMKFMT	DMKFRE	DMKGIO	DMKGRF	DMKHVC	DMKHVD	DMKIOC
		DMKIOE	DMKIOF	DMKIOG	DMKIOS	DMKISM	DMKLD00E	DMKLNK	DMKLOC	DMKLOG	DMKMCC	DMKMCH	DMKMID
		DMKMON	DMKMSG	DMKMSW	DMKNEM	DMKNES	DMKNET	DMKNLD	DMKOPR	DMKPAG	DMKPGS	DMKPGT	DMKPRG
		DMKPRV	DMKPSA	DMKPTR	DMKQCN	DMKRGA	DMKRGB	DMKRNH	DMKRPA	DMKRSE	DMKRSP	DMKSAV	DMKSCH
		DMKSCN	DMKSEP	DMKSEV	DMKSIX	DMKSNC	DMKSPL	DMKSSP	DMKSTK	DMKTAP	DMKTDK	DMKTHI	DMKTMR
		DMKTRA	DMKTRC	DMKTRM	DMKUDR	DMKUNT	DMKUSO	DMKVAT	DMKVCA	DMKVCH	DMKVCN	DMKVDB	DMKVDR
		DMKVDS	DMKVER	DMKVIO	DMKVMA	DMKVMI	DMKVSP	DMKWRM					
R1	009506	DMKACO	DMKBLD	DMKBSC	DMKCCH	DMKCCW	DMKCDB	DMKCDS	DMKCFC	DMKCFD	DMKCFG	DMKCFM	DMKCFP
		DMKCFS	DMKCFT	DMKCKP	DMKCKS	DMKCNS	DMKCPB	DMKCPI	DMKCPS	DMKCPV	DMKCQG	DMKCQP	DMKCQR
		DMKCSO	DMKCSP	DMKCST	DMKCSU	DMKCVT	DMKDAS	DMKDDR	DMKDEF	DMKDGD	DMKDIA	DMKDIR	DMKDMP
		DMKDRD	DMKDSP	DMKEDM	DMKEIG	DMKERM	DMKFMT	DMKFRE	DMKGIO	DMKGRF	DMKHVC	DMKHVD	DMKIOE
		DMKIOF	DMKIOG	DMKIOS	DMKISM	DMKLD00E	DMKLNK	DMKLOC	DMKLOG	DMKMCC	DMKMCH	DMKMID	DMKMON
		DMKMSG	DMKMSW	DMKNEM	DMKNES	DMKNET	DMKNLD	DMKOPR	DMKPAG	DMKPGS	DMKPGT	DMKPRG	DMKPRV
		DMKPSA	DMKPTR	DMKQCN	DMKRGA	DMKRGB	DMKRNH	DMKRPA	DMKRSE	DMKRSP	DMKSAV	DMKSCH	DMKSCN
		DMKSEP	DMKSEV	DMKSIX	DMKSNC	DMKSPL	DMKSSP	DMKSTK	DMKTAP	DMKTDK	DMKTHI	DMKTMR	DMKTRA
		DMKTRC	DMKTRM	DMKUDR	DMKUNT	DMKUSO	DMKVAT	DMKVCA	DMKVCH	DMKVCN	DMKVDB	DMKVDR	DMKVDS
		DMKVER	DMKVIO	DMKVMA	DMKVMI	DMKVSP	DMKWRM						
R10	001964	DMKACO	DMKBLD	DMKBSC	DMKCCH	DMKCCW	DMKCDB	DMKCFD	DMKCFG	DMKCFP	DMKCFS	DMKCFT	DMKCKP
		DMKCKS	DMKCNS	DMKCPB	DMKCPI	DMKCPS	DMKCPV	DMKCQG	DMKCQP	DMKCQR	DMKCSO	DMKCSP	DMKCST
		DMKCSU	DMKCVT	DMKDAS	DMKDDR	DMKDEF	DMKDGD	DMKDIA	DMKDIR	DMKDMP	DMKDRD	DMKDSP	DMKEDM
		DMKERM	DMKFMT	DMKFRE	DMKGIO	DMKGRF	DMKHVC	DMKHVD	DMKIOE	DMKIOF	DMKIOG	DMKIOS	DMKISM
		DMKLD00E	DMKLNK	DMKLOC	DMKLOG	DMKMCC	DMKMCH	DMKMID	DMKMON	DMKMSG	DMKMSW	DMKNES	DMKNET
		DMKNLD	DMKOPR	DMKPAG	DMKPGS	DMKPGT	DMKPRG	DMKPRV	DMKPSA	DMKPTR	DMKQCN	DMKRGA	DMKRGB
		DMKRNH	DMKRSE	DMKRSP	DMKSAV	DMKSCH	DMKSCN	DMKSEP	DMKSNC	DMKSPL	DMKSSP	DMKSTK	DMKTAP
		DMKTDK	DMKTHI	DMKTMR	DMKTRC	DMKUDR	DMKUNT	DMKUSO	DMKVAT	DMKVCA	DMKVCN	DMKVDB	DMKVDR
		DMKVDS	DMKVER	DMKVIO	DMKVMA	DMKVMI	DMKVSP	DMKWRM					

Label	Count	References											
R11	000656	DMKACO	DMKBLD	DMKCCH	DMKCCW	DMKCDB	DMKCDS	DMKCFC	DMKCFD	DMKCFG	DMKCFM	DMKCFP	DMKCFS
		DMKCFT	DMKCKP	DMKCKS	DMKCNS	DMKCPB	DMKCPI	DMKCPS	DMKCPV	DMKCQG	DMKCQP	DMKCQR	DMKCSO
		DMKCSP	DMKCST	DMKCSU	DMKDAS	DMKDDR	DMKDEF	DMKDGD	DMKDIA	DMKDIR	DMKDMP	DMKDRD	DMKDSP
		DMKEDM	DMKERM	DMKFMT	DMKFRE	DMKGIO	DMKGRF	DMKHVC	DMKHVD	DMKIOE	DMKIOF	DMKIOG	DMKIOS
		DMKISM	DMKLNK	DMKLOG	DMKMCC	DMKMCH	DMKMID	DMKMON	DMKMSG	DMKMSW	DMKNES	DMKNET	DMKNLD
		DMKPAG	DMKPGS	DMKPGT	DMKPRG	DMKPRV	DMKPSA	DMKPTR	DMKQCN	DMKRGA	DMKRGB	DMKRNH	DMKRPA
		DMKRSE	DMKRSP	DMKSAV	DMKSCH	DMKSCN	DMKSEP	DMKSNC	DMKSPL	DMKSSP	DMKTDK	DMKTHI	DMKTMR
		DMKTRA	DMKTRC	DMKUDR	DMKUNT	DMKUSO	DMKVAT	DMKVCA	DMKVCH	DMKVCN	DMKVDB	DMKVDR	DMKVDS
		DMKVER	DMKVIO	DMKVMA	DMKVMI	DMKVSP							
R12	000605	DMKACO	DMKBLD	DMKCCW	DMKCFM	DMKCFP	DMKCKP	DMKCNS	DMKCPB	DMKCPI	DMKCPS	DMKCQG	DMKCQP
		DMKCQR	DMKCSO	DMKCSU	DMKDAS	DMKDDR	DMKDEF	DMKDGD	DMKDIA	DMKDIR	DMKDMP	DMKDRD	DMKDSP
		DMKEDM	DMKEIG	DMKERM	DMKFMT	DMKFRE	DMKGIO	DMKGRF	DMKHVC	DMKIOC	DMKIOE	DMKIOF	DMKIOG
		DMKIOS	DMKISM	DMKLD00E	DMKLNK	DMKLOC	DMKLOG	DMKMCH	DMKMID	DMKMON	DMKMSW	DMKNEM	DMKNLD
		DMKPAG	DMKPGT	DMKPRG	DMKPRV	DMKPSA	DMKPTR	DMKQCN	DMKRGA	DMKRGB	DMKRNH	DMKRSP	DMKSAV
		DMKSCH	DMKSEP	DMKSEV	DMKSIX	DMKSPL	DMKSSP	DMKTDK	DMKTMR	DMKTRA	DMKTRC	DMKUDR	DMKUNT
		DMKVAT	DMKVCA	DMKVCN	DMKVDB	DMKVDS	DMKVIO	DMKVMA	DMKVMI	DMKVSP	DMKWRM		
R13	000501	DMKACO	DMKBLD	DMKBSC	DMKCCH	DMKCCW	DMKCDB	DMKCDS	DMKCFC	DMKCFD	DMKCFG	DMKCFM	DMKCFP
		DMKCFS	DMKCFT	DMKCKP	DMKCKS	DMKCNS	DMKCPB	DMKCPI	DMKCPS	DMKCPV	DMKCQG	DMKCQP	DMKCQR
		DMKCSO	DMKCSP	DMKCST	DMKCSU	DMKDAS	DMKDDR	DMKDEF	DMKDGD	DMKDIA	DMKDIR	DMKDMP	DMKDRD
		DMKEDM	DMKEIG	DMKERM	DMKFMT	DMKFRE	DMKGIO	DMKGRF	DMKHVC	DMKHVD	DMKIOC	DMKIOE	DMKIOF
		DMKIOG	DMKIOS	DMKISM	DMKLD00E	DMKLNK	DMKLOG	DMKMCC	DMKMCH	DMKMID	DMKMON	DMKMSG	DMKMSW
		DMKNEM	DMKNES	DMKNET	DMKNLD	DMKPAG	DMKPGS	DMKPGT	DMKPRG	DMKPRV	DMKPSA	DMKPTR	DMKQCN
		DMKRGA	DMKRGB	DMKRNH	DMKRPA	DMKRSE	DMKRSP	DMKSAV	DMKSCH	DMKSEP	DMKSEV	DMKSIX	DMKSNC
		DMKSPL	DMKSSP	DMKTAP	DMKTDK	DMKTHI	DMKTRA	DMKTRC	DMKTRM	DMKUDR	DMKUNT	DMKUSO	DMKVAT
		DMKVCA	DMKVCH	DMKVCN	DMKVDB	DMKVDR	DMKVDS	DMKVER	DMKVIO	DMKVMA	DMKVMI	DMKVSP	DMKWRM
R14	002696	DMKACO	DMKBLD	DMKCCH	DMKCCW	DMKCDB	DMKCDS	DMKCFG	DMKCFM	DMKCFP	DMKCFS	DMKCFT	DMKCKP
		DMKCKS	DMKCNS	DMKCPI	DMKCPS	DMKCPV	DMKCQP	DMKCQR	DMKCSO	DMKCSP	DMKCST	DMKCSU	DMKCVT
		DMKDDR	DMKDGD	DMKDIA	DMKDIR	DMKDMP	DMKDRD	DMKDSP	DMKEDM	DMKEIG	DMKERM	DMKFMT	DMKFRE
		DMKGRF	DMKHVC	DMKHVD	DMKIOE	DMKIOF	DMKIOG	DMKIOS	DMKISM	DMKLD00E	DMKLNK	DMKLOC	DMKLOG
		DMKMCH	DMKMON	DMKNET	DMKNLD	DMKOPR	DMKPAG	DMKPGS	DMKPGT	DMKPRG	DMKPRV	DMKPSA	DMKPTR
		DMKQCN	DMKRGA	DMKRGB	DMKRNH	DMKRPA	DMKRSE	DMKRSP	DMKSAV	DMKSCH	DMKSCN	DMKSEP	DMKSEV
		DMKSIX	DMKSPL	DMKSSP	DMKSTK	DMKTDK	DMKTMR	DMKTRA	DMKTRC	DMKUDR	DMKUNT	DMKUSO	DMKVAT
		DMKVCA	DMKVCH	DMKVCN	DMKVDB	DMKVDS	DMKVER	DMKVIO	DMKVMA	DMKVMI	DMKVSP	DMKWRM	
R15	002637	DMKACO	DMKBLD	DMKCCH	DMKCCW	DMKCDB	DMKCDS	DMKCFC	DMKCFD	DMKCFG	DMKCFM	DMKCFP	DMKCFS
		DMKCFT	DMKCKP	DMKCKS	DMKCNS	DMKCPB	DMKCPI	DMKCPS	DMKCPV	DMKCQG	DMKCQR	DMKCSO	DMKCSU
		DMKCVT	DMKDAS	DMKDDR	DMKDEF	DMKDGD	DMKDIA	DMKDIR	DMKDMP	DMKDRD	DMKDSP	DMKEDM	DMKEIG
		DMKERM	DMKFMT	DMKFRE	DMKGIO	DMKGRF	DMKHVC	DMKHVD	DMKIOE	DMKIOF	DMKIOG	DMKIOS	DMKISM
		DMKLD00E	DMKLNK	DMKLOC	DMKLOG	DMKMCH	DMKMON	DMKMSG	DMKNEM	DMKNES	DMKNET	DMKNLD	DMKOPR
		DMKPAG	DMKPGS	DMKPGT	DMKPRG	DMKPRV	DMKPSA	DMKPTR	DMKQCN	DMKRGA	DMKRGB	DMKRNH	DMKRPA
		DMKRSE	DMKRSP	DMKSAV	DMKSCH	DMKSCN	DMKSEP	DMKSEV	DMKSIX	DMKSPL	DMKSTK	DMKTDK	DMKTMR
		DMKTRA	DMKTRC	DMKUDR	DMKUNT	DMKUSO	DMKVAT	DMKVCA	DMKVCH	DMKVCN	DMKVDB	DMKVDR	DMKVDS
		DMKVER	DMKVIO	DMKVMA	DMKVMI	DMKVSP	DMKWRM						

Label	Count	References											
R2	006218	DMKACO	DMKBLD	DMKBSC	DMKCCH	DMKCCW	DMKCDB	DMKCDS	DMKCFC	DMKCFD	DMKCFG	DMKCFM	DMKCFP
		DMKCFS	DMKCFT	DMKCKP	DMKCKS	DMKCNS	DMKCPB	DMKCPI	DMKCPS	DMKCPV	DMKCQG	DMKCQP	DMKCQR
		DMKCSO	DMKCSP	DMKCST	DMKCSU	DMKCVT	DMKDAS	DMKDDR	DMKDEF	DMKDGD	DMKDIA	DMKDIR	DMKDMP
		DMKDRD	DMKDSP	DMKEDM	DMKEIG	DMKERM	DMKFMT	DMKFRE	DMKGIO	DMKGRF	DMKHVC	DMKHVD	DMKIOE
		DMKIOF	DMKIOG	DMKIOS	DMKISM	DMKLD00E	DMKLNK	DMKLOC	DMKLOG	DMKMCC	DMKMCH	DMKMID	DMKMON
		DMKMSG	DMKMSW	DMKNEM	DMKNES	DMKNET	DMKNLD	DMKOPR	DMKPAG	DMKPGS	DMKPGT	DMKPRG	DMKPRV
		DMKPSA	DMKPTR	DMKQCN	DMKRGA	DMKRGB	DMKRNH	DMKRPA	DMKRSE	DMKRSP	DMKSAV	DMKSCH	DMKSCN
		DMKSEP	DMKSEV	DMKSIX	DMKSNC	DMKSPL	DMKSSP	DMKTAP	DMKTDK	DMKTHI	DMKTMR	DMKTRA	DMKTRC
		DMKTRM	DMKUDR	DMKUNT	DMKUSO	DMKVAT	DMKVCA	DMKVCH	DMKVCN	DMKVDB	DMKVDR	DMKVDS	DMKVER
		DMKVIO	DMKVMA	DMKVMI	DMKVSP	DMKWRM							
R3	004791	DMKACO	DMKBLD	DMKBSC	DMKCCH	DMKCCW	DMKCDB	DMKCDS	DMKCFC	DMKCFD	DMKCFG	DMKCFM	DMKCFP
		DMKCFS	DMKCFT	DMKCKP	DMKCKS	DMKCNS	DMKCPB	DMKCPI	DMKCPS	DMKCPV	DMKCQG	DMKCQP	DMKCQR
		DMKCSO	DMKCSP	DMKCST	DMKCSU	DMKCVT	DMKDAS	DMKDDR	DMKDEF	DMKDGD	DMKDIA	DMKDIR	DMKDMP
		DMKDRD	DMKDSP	DMKEDM	DMKEIG	DMKERM	DMKFMT	DMKFRE	DMKGRF	DMKHVC	DMKHVD	DMKIOE	DMKIOF
		DMKIOG	DMKIOS	DMKISM	DMKLD00E	DMKLNK	DMKLOC	DMKLOG	DMKMCC	DMKMCH	DMKMID	DMKMON	DMKMSG
		DMKMSW	DMKNEM	DMKNES	DMKNET	DMKNLD	DMKOPR	DMKPAG	DMKPGS	DMKPGT	DMKPRG	DMKPRV	DMKPSA
		DMKPTR	DMKQCN	DMKRGA	DMKRGB	DMKRNH	DMKRPA	DMKRSE	DMKRSP	DMKSAV	DMKSCH	DMKSCN	DMKSEP
		DMKSEV	DMKSIX	DMKSNC	DMKSPL	DMKSSP	DMKTAP	DMKTDK	DMKTHI	DMKTMR	DMKTRA	DMKTRC	DMKTRM
		DMKUDR	DMKUNT	DMKUSO	DMKVAT	DMKVCA	DMKVCH	DMKVCN	DMKVDB	DMKVDR	DMKVDS	DMKVER	DMKVIO
		DMKVMA	DMKVMI	DMKVSP	DMKWRM								
R4	003787	DMKACO	DMKBLD	DMKBSC	DMKCCH	DMKCCW	DMKCDB	DMKCDS	DMKCFC	DMKCFD	DMKCFG	DMKCFM	DMKCFP
		DMKCFS	DMKCFT	DMKCKP	DMKCKS	DMKCNS	DMKCPB	DMKCPI	DMKCPS	DMKCPV	DMKCQG	DMKCQP	DMKCQR
		DMKCSO	DMKCSP	DMKCST	DMKCSU	DMKDAS	DMKDDR	DMKDEF	DMKDGD	DMKDIA	DMKDIR	DMKDMP	DMKDRD
		DMKDSP	DMKEDM	DMKEIG	DMKERM	DMKFMT	DMKFRE	DMKGIO	DMKGRF	DMKHVC	DMKHVD	DMKIOC	DMKIOE
		DMKIOF	DMKIOG	DMKIOS	DMKISM	DMKLD00E	DMKLNK	DMKLOC	DMKLOG	DMKMCC	DMKMCH	DMKMID	DMKMON
		DMKMSG	DMKMSW	DMKNEM	DMKNES	DMKNET	DMKNLD	DMKOPR	DMKPAG	DMKPGS	DMKPGT	DMKPRG	DMKPRV
		DMKPSA	DMKPTR	DMKQCN	DMKRGA	DMKRGB	DMKRNH	DMKRSE	DMKRSP	DMKSAV	DMKSCH	DMKSCN	DMKSEP
		DMKSEV	DMKSIX	DMKSNC	DMKSPL	DMKSSP	DMKTAP	DMKTDK	DMKTHI	DMKTMR	DMKTRA	DMKTRC	DMKTRM
		DMKUDR	DMKUNT	DMKUSO	DMKVAT	DMKVCA	DMKVCH	DMKVCN	DMKVDB	DMKVDR	DMKVDS	DMKVER	DMKVIO
		DMKVMA	DMKVMI	DMKVSP	DMKWRM								
R5	003502	DMKACO	DMKBLD	DMKBSC	DMKCCH	DMKCCW	DMKCDB	DMKCDS	DMKCFC	DMKCFD	DMKCFG	DMKCFM	DMKCFP
		DMKCFS	DMKCFT	DMKCKP	DMKCKS	DMKCNS	DMKCPB	DMKCPI	DMKCPS	DMKCPV	DMKCQG	DMKCQP	DMKCQR
		DMKCSO	DMKCSP	DMKCST	DMKCSU	DMKCVT	DMKDAS	DMKDDR	DMKDEF	DMKDGD	DMKDIA	DMKDIR	DMKDMP
		DMKDRD	DMKDSP	DMKEDM	DMKERM	DMKFMT	DMKFRE	DMKGIO	DMKGRF	DMKHVC	DMKHVD	DMKIOC	DMKIOE
		DMKIOF	DMKIOG	DMKIOS	DMKISM	DMKLD00E	DMKLNK	DMKLOC	DMKLOG	DMKMCC	DMKMCH	DMKMID	DMKMON
		DMKMSG	DMKMSW	DMKNEM	DMKNES	DMKNET	DMKNLD	DMKOPR	DMKPAG	DMKPGS	DMKPGT	DMKPRG	DMKPRV
		DMKPTR	DMKQCN	DMKRGA	DMKRGB	DMKRNH	DMKRPA	DMKRSE	DMKRSP	DMKSAV	DMKSCH	DMKSCN	DMKSEP
		DMKSNC	DMKSPL	DMKSSP	DMKTAP	DMKTDK	DMKTHI	DMKTMR	DMKTRA	DMKTRC	DMKTRM	DMKUDR	DMKUNT
		DMKUSO	DMKVAT	DMKVCA	DMKVCH	DMKVCN	DMKVDB	DMKVDS	DMKVER	DMKVIO	DMKVMA	DMKVMI	DMKVSP
		DMKWRM											

Label	Count	References											
R6	002567	DMKBLD	DMKBSC	DMKCCH	DMKCCW	DMKCDB	DMKCDS	DMKCFC	DMKCFD	DMKCFG	DMKCFM	DMKCFP	DMKCFS
		DMKCFT	DMKCKP	DMKCKS	DMKCNS	DMKCPB	DMKCPI	DMKCPS	DMKCPV	DMKCQG	DMKCQP	DMKCQR	DMKCSO
		DMKCSP	DMKCST	DMKCSU	DMKCVT	DMKDAS	DMKDDR	DMKDEF	DMKDGD	DMKDIA	DMKDMP	DMKDRD	DMKDSP
		DMKEDM	DMKERM	DMKFMT	DMKFRE	DMKGIO	DMKGRF	DMKHVC	DMKHVD	DMKIOC	DMKIOE	DMKIOF	DMKIOG
		DMKIOS	DMKISM	DMKLD00E	DMKLNK	DMKLOC	DMKLOG	DMKMCC	DMKMCH	DMKMID	DMKMON	DMKMSW	DMKNES
		DMKNET	DMKNLD	DMKPAG	DMKPGS	DMKPGT	DMKPRG	DMKPRV	DMKPTR	DMKQCN	DMKRGA	DMKRGB	DMKRNH
		DMKRSE	DMKRSP	DMKSAV	DMKSCH	DMKSCN	DMKSEP	DMKSNC	DMKSPL	DMKSSP	DMKTAP	DMKTDK	DMKTHI
		DMKTMR	DMKTRC	DMKUDR	DMKUNT	DMKUSO	DMKVAT	DMKVCA	DMKVCH	DMKVCN	DMKVDB	DMKVDR	DMKVDS
		DMKVER	DMKVIO	DMKVMA	DMKVMI	DMKVSP	DMKWRM						
R7	002858	DMKACO	DMKBLD	DMKBSC	DMKCCH	DMKCCW	DMKCDB	DMKCDS	DMKCFC	DMKCFD	DMKCFG	DMKCFM	DMKCFP
		DMKCFS	DMKCFT	DMKCKP	DMKCKS	DMKCNS	DMKCPB	DMKCPI	DMKCPS	DMKCPV	DMKCQG	DMKCQP	DMKCQR
		DMKCSO	DMKCSP	DMKCST	DMKCSU	DMKCVT	DMKDAS	DMKDDR	DMKDEF	DMKDGD	DMKDIA	DMKDMP	DMKDRD
		DMKDSP	DMKEDM	DMKERM	DMKFMT	DMKFRE	DMKGIO	DMKGRF	DMKHVC	DMKHVD	DMKIOE	DMKIOF	DMKIOS
		DMKISM	DMKLD00E	DMKLNK	DMKLOC	DMKLOG	DMKMCC	DMKMCH	DMKMID	DMKMON	DMKMSG	DMKMSW	DMKNES
		DMKNET	DMKNLD	DMKPAG	DMKPGS	DMKPGT	DMKPRG	DMKPRV	DMKPTR	DMKQCN	DMKRGA	DMKRGB	DMKRNH
		DMKRPA	DMKRSE	DMKRSP	DMKSAV	DMKSCH	DMKSCN	DMKSEP	DMKSNC	DMKSPL	DMKSSP	DMKTAP	DMKTDK
		DMKTHI	DMKTMR	DMKTRC	DMKUDR	DMKUNT	DMKUSO	DMKVAT	DMKVCA	DMKVCH	DMKVCN	DMKVDB	DMKVDS
		DMKVER	DMKVIO	DMKVMA	DMKVSP	DMKWRM							
R8	002093	DMKACO	DMKBLD	DMKBSC	DMKCCH	DMKCCW	DMKCDB	DMKCDS	DMKCFD	DMKCFG	DMKCFM	DMKCFP	DMKCFS
		DMKCFT	DMKCKP	DMKCKS	DMKCNS	DMKCPB	DMKCPI	DMKCPV	DMKCPV	DMKCQG	DMKCQP	DMKCQR	DMKCSO
		DMKCSP	DMKCST	DMKCSU	DMKCVT	DMKDAS	DMKDDR	DMKDEF	DMKDGD	DMKDIA	DMKDMP	DMKDRD	DMKDSP
		DMKEDM	DMKERM	DMKFMT	DMKFRE	DMKGIO	DMKGRF	DMKHVC	DMKHVD	DMKIOC	DMKIOE	DMKIOF	DMKIOG
		DMKIOS	DMKISM	DMKLD00E	DMKLNK	DMKLOC	DMKLOG	DMKMCC	DMKMCH	DMKMID	DMKMON	DMKMSG	DMKMSW
		DMKNES	DMKNET	DMKNLD	DMKOPR	DMKPAG	DMKPGS	DMKPGT	DMKPRG	DMKPRV	DMKPSA	DMKPTR	DMKQCN
		DMKRGA	DMKRGB	DMKRNH	DMKRSE	DMKRSP	DMKSAV	DMKSCH	DMKSCN	DMKSEP	DMKSNC	DMKSPL	DMKSSP
		DMKTAP	DMKTDK	DMKTHI	DMKTMR	DMKTRC	DMKTRM	DMKUDR	DMKUNT	DMKUSO	DMKVAT	DMKVCA	DMKVCH
		DMKVCN	DMKVDB	DMKVDR	DMKVDS	DMKVER	DMKVIO	DMKVMA	DMKVSP	DMKWRM			
R9	002001	DMKACO	DMKBLD	DMKBSC	DMKCCH	DMKCCW	DMKCDB	DMKCFG	DMKCFM	DMKCFP	DMKCFS	DMKCKP	DMKCKS
		DMKCNS	DMKCPI	DMKCPS	DMKCPV	DMKCQG	DMKCQP	DMKCQR	DMKCSO	DMKCSP	DMKCST	DMKCSU	DMKCVT
		DMKDAS	DMKDDR	DMKDGD	DMKDIA	DMKDIR	DMKDMP	DMKDRD	DMKDSP	DMKEDM	DMKEIG	DMKERM	DMKFMT
		DMKFRE	DMKGIO	DMKGRF	DMKHVC	DMKHVD	DMKIOC	DMKIOE	DMKIOF	DMKIOG	DMKIOS	DMKISM	DMKLD00E
		DMKLNK	DMKLOC	DMKLOG	DMKMCC	DMKMCH	DMKMID	DMKMON	DMKMSG	DMKMSW	DMKNES	DMKNET	DMKNLD
		DMKPAG	DMKPGS	DMKPGT	DMKPRG	DMKPRV	DMKPTR	DMKQCN	DMKRGA	DMKRGB	DMKRNH	DMKRPA	DMKRSE
		DMKRSP	DMKSAV	DMKSCH	DMKSCN	DMKSEP	DMKSEV	DMKSIX	DMKSNC	DMKSPL	DMKSSP	DMKTAP	DMKTDK
		DMKTHI	DMKTMR	DMKTRA	DMKTRC	DMKUDR	DMKUNT	DMKUSO	DMKVAT	DMKVCA	DMKVCH	DMKVCN	DMKVDB
		DMKVDS	DMKVER	DMKVIO	DMKVMA	DMKVMI	DMKVSP	DMKWRM					
SAVCREGS	000007	DMKBLD	DMKCFG										
SAVEAREA	000143	DMKACO	DMKBLD	DMKBSC	DMKCCH	DMKCCW	DMKCDB	DMKCDS	DMKCFC	DMKCFD	DMKCFG	DMKCFM	DMKCFP
		DMKCFS	DMKCFT	DMKCKS	DMKCNS	DMKCPB	DMKCPS	DMKCPV	DMKCQG	DMKCQP	DMKCQR	DMKCSO	DMKCSP
		DMKCST	DMKCSU	DMKDAS	DMKDEF	DMKDGD	DMKDIA	DMKDRD	DMKEIG	DMKERM	DMKGIO	DMKGRF	DMKHVD
		DMKIOC	DMKIOE	DMKIOF	DMKIOG	DMKIOS	DMKISM	DMKLNK	DMKLOG	DMKMCC	DMKMCH	DMKMID	DMKMON
		DMKMSG	DMKMSW	DMKNEM	DMKNES	DMKNET	DMKNLD	DMKPGS	DMKPSA	DMKPTR	DMKQCN	DMKRGA	DMKRGB
		DMKRNH	DMKRPA	DMKRSE	DMKRSP	DMKSEP	DMKSEV	DMKSIX	DMKSNC	DMKSPL	DMKSSP	DMKTAP	DMKTDK
		DMKTHI	DMKTRA	DMKTRC	DMKTRM	DMKUDR	DMKUNT	DMKUSO	DMKVAT	DMKVCA	DMKVCH	DMKVDB	DMKVDR
		DMKVDS	DMKVER	DMKVMA	DMKVSP	DMKWRM							

Label	Count	References											
SAVENEXT	000004	DMKPSA											
SAVEREGS	000032	DMKCCW	DMKCPV	DMKCQP	DMKDGD	DMKIOE	DMKIOG	DMKPTR	DMKRSP	DMKSEP	DMKVAT		
SAVERETN	000031	DMKCFC	DMKCFG	DMKCPB	DMKDIA	DMKLNK	DMKLOG	DMKPSA	DMKPTR	DMKUSO	DMKVAT	DMKVCA	DMKVER
SAVER0	000049	DMKCNS	DMKCQG	DMKCQP	DMKCQR	DMKDRD	DMKERM	DMKGRF	DMKHVD	DMKMSW	DMKNEM	DMKPTR	DMKQCN
		DMKRGA	DMKRNH	DMKRSP	DMKSNC	DMKTDK	DMKTRC	DMKUDR	DMKVAT	DMKVCA	DMKVSP		
SAVER1	000042	DMKBLD	DMKCCW	DMKCFC	DMKCKS	DMKCNS	DMKCQP	DMKERM	DMKIOE	DMKPGS	DMKPTR	DMKQCN	DMKRNH
		DMKRPA	DMKTDK	DMKTRC	DMKVAT	DMKVCA	DMKVDS	DMKVSP					
SAVER10	000013	DMKCCW	DMKSEP	DMKVCA	DMKVCH								
SAVER11	000059	DMKACO	DMKBLD	DMKCFM	DMKCPS	DMKCPV	DMKCQG	DMKCQP	DMKCSU	DMKDAS	DMKDIA	DMKIOS	DMKLNK
		DMKLOG	DMKMID	DMKMSG	DMKMSW	DMKNES	DMKNLD	DMKPTR	DMKQCN	DMKSPL	DMKTHI	DMKUSO	DMKVCA
		DMKVCH	DMKVDB	DMKVMA									
SAVER12	000014	DMKCCW	DMKDGD	DMKPSA	DMKPTR	DMKVAT	DMKVCA	DMKVER					
SAVER13	000003	DMKPSA	DMKPTR	DMKVAT									
SAVER2	000148	DMKBLD	DMKCCW	DMKCDS	DMKCFC	DMKCFM	DMKCKS	DMKCNS	DMKCQP	DMKDEF	DMKDGD	DMKDIA	DMKDRD
		DMKERM	DMKGIO	DMKGRF	DMKLNK	DMKLOG	DMKMSG	DMKNES	DMKNET	DMKNLD	DMKPGS	DMKPSA	DMKPTR
		DMKQCN	DMKRGA	DMKRGB	DMKRNH	DMKRPA	DMKSNC	DMKTRA	DMKTRC	DMKUDR	DMKVAT	DMKVCH	DMKVDB
		DMKVDS	DMKVMA	DMKVSP	DMKWRM								
SAVER3	000006	DMKERM	DMKPTR	DMKQCN	DMKVAT								
SAVER4	000001	DMKTRC											
SAVER5	000007	DMKCFG	DMKTRC	DMKVCA									
SAVER6	000008	DMKCFG	DMKDRD	DMKSNC									
SAVER7	000002	DMKPTR	DMKSPL										
SAVER8	000022	DMKBLD	DMKCCW	DMKCKS	DMKDIA	DMKSEP	DMKSPL	DMKTDK	DMKVDR	DMKVDS	DMKVSP		
SAVER9	000007	DMKCCW	DMKLOG	DMKNES	DMKNET	DMKSPL							
SAVESIZE	000008	DMKCPI	DMKFRE	DMKPSA									
SAVEWRK1	001001	DMKBLD	DMKCCH	DMKCCW	DMKCDB	DMKCDS	DMKCFD	DMKCFG	DMKCFS	DMKCFT	DMKCKS	DMKCPS	DMKCPV
		DMKCQG	DMKCQP	DMKCQR	DMKCSO	DMKCSP	DMKCST	DMKCSU	DMKDEF	DMKDIA	DMKDRD	DMKEIG	DMKLNK
		DMKLOG	DMKMCC	DMKMSG	DMKNES	DMKNET	DMKNLD	DMKPGS	DMKPTR	DMKQCN	DMKRPA	DMKRSE	DMKSEP
		DMKSEV	DMKSIX	DMKSNC	DMKSPL	DMKTHI	DMKTRA	DMKTRC	DMKUNT	DMKUSO	DMKVCA	DMKVCH	DMKVDB
		DMKVDS	DMKVER	DMKVMA	DMKWRM								
SAVEWRK2	000552	DMKACO	DMKBLD	DMKCCH	DMKCCW	DMKCDB	DMKCDS	DMKCFC	DMKCFD	DMKCFG	DMKCFS	DMKCKS	DMKCPB
		DMKCPS	DMKCPV	DMKCQG	DMKCQP	DMKCQR	DMKCSO	DMKCSP	DMKCST	DMKCSU	DMKDEF	DMKDIA	DMKDRD
		DMKIOG	DMKLNK	DMKLOG	DMKMSG	DMKNES	DMKNET	DMKNLD	DMKPGS	DMKPSA	DMKPTR	DMKQCN	DMKSEP
		DMKSNC	DMKSPL	DMKTAP	DMKTDK	DMKTHI	DMKTRA	DMKTRC	DMKUDR	DMKUNT	DMKUSO	DMKVCA	DMKVCH
		DMKVDB	DMKVDR	DMKVDS	DMKVER	DMKVMA	DMKVSP	DMKWRM					
SAVEWRK3	000138	DMKACO	DMKCCH	DMKCDS	DMKCFD	DMKCFG	DMKCFS	DMKCKS	DMKCPS	DMKCPV	DMKCQG	DMKCQP	DMKDEF
		DMKDIA	DMKDRD	DMKIOG	DMKMCC	DMKNES	DMKNET	DMKNLD	DMKPGS	DMKPTR	DMKQCN	DMKSNC	DMKTAP
		DMKTHI	DMKTRC	DMKUNT	DMKUSO	DMKVCA	DMKVDB	DMKVDR	DMKVDS	DMKVER	DMKVMA	DMKWRM	
SAVEWRK4	000170	DMKCCH	DMKCDB	DMKCDS	DMKCFC	DMKCFD	DMKCFG	DMKCFS	DMKCKS	DMKCPB	DMKCPS	DMKCPV	DMKCQG
		DMKCQR	DMKCSO	DMKCSP	DMKCST	DMKCSU	DMKDEF	DMKDIA	DMKLNK	DMKMSG	DMKNES	DMKNET	DMKNLD
		DMKPGS	DMKQCN	DMKSNC	DMKTRC	DMKVCA	DMKVDB	DMKVDR	DMKVDS	DMKVER	DMKWRM		

Label	Count	References
SAVEWRK5	000148	DMKCCH DMKCDB DMKCDS DMKCFD DMKCFG DMKCFS DMKCKS DMKCPB DMKCQG DMKCQP DMKCSO DMKCSP
		DMKCST DMKCSU DMKDEF DMKDIA DMKLNK DMKNES DMKNET DMKNLD DMKPTR DMKSEP DMKSNC DMKTRC
		DMKUNT DMKVDB DMKVER DMKVMA
SAVEWRK6	000148	DMKACO DMKCCH DMKCDB DMKCFG DMKCFM DMKCFP DMKCKS DMKCQP DMKCSO DMKCSP DMKCST DMKCSU
		DMKDEF DMKDIA DMKDRD DMKLNK DMKMSG DMKNLD DMKPTR DMKSNC DMKTHI DMKTRC DMKUNT DMKVCA
		DMKVDB DMKVDR DMKVDS DMKVMA DMKVSP DMKWRM
SAVEWRK7	000066	DMKACO DMKCCH DMKCFD DMKCFG DMKCFS DMKCKS DMKCSO DMKCST DMKDEF DMKDIA DMKIOG DMKLNK
		DMKNES DMKNET DMKNLD DMKPGS DMKSEP DMKTRA DMKTRC DMKVDS DMKVER DMKVMA DMKWRM
SAVEWRK8	000229	DMKACO DMKCCH DMKCDB DMKCDS DMKCFP DMKCFS DMKCKS DMKCPV DMKCQG DMKCQP DMKCQR DMKCSO
		DMKCSP DMKCSU DMKDEF DMKDIA DMKLNK DMKLOG DMKMSG DMKNES DMKNET DMKNLD DMKSEP DMKSNC
		DMKSPL DMKTRC DMKVMA
SAVEWRK9	000116	DMKACO DMKCCH DMKCCW DMKCFG DMKCKS DMKCPS DMKCSP DMKCSU DMKDEF DMKDGD DMKDIA DMKEIG
		DMKLNK DMKNES DMKNET DMKNLD DMKPGS DMKPTR DMKSEP DMKSEV DMKSIX DMKSNC DMKTRC DMKUNT
		DMKVCA DMKVDB DMKVDR DMKVDS DMKVER DMKVMA
SAVFPRES	000002	DMKCFG
SAVGREGS	000003	DMKCFG
SAVKEYS	000002	DMKCFG
SAVPSW	000002	DMKCFG
SAVTABLE	000002	DMKCFG
SCHEDCI	000001	DMKMCC
SDRBLOK	000006	DMKIOE DMKIOF
SDRBSIZE	000001	DMKIOE
SDRCTRS	000014	DMKIOE DMKIOF
SDRCUA	000002	DMKIOF
SDRFLAGS	000005	DMKIOE DMKIOF
SDRFLCT	000004	DMKIOF
SDRLNGTH	000008	DMKIOE DMKIOF
SDRMAX	000003	DMKIOF
SDROVFWK	000006	DMKIOF
SDRPRMCT	000004	DMKIOF
SDRRDEV	000002	DMKIOF
SDRSHRT	000007	DMKIOE DMKIOF
SDRSIZE	000001	DMKIOF
SDRSIZE1	000001	DMKIOF
SEGPAGE	000033	DMKBLD DMKCPI DMKPGS DMKVMA
SEGPLEN	000011	DMKBLD DMKPGS DMKVMA
SEGTABLE	000015	DMKBLD DMKPGS DMKVMA
SFBCLAS	000025	DMKCKP DMKCQG DMKCQR DMKCSP DMKCSU DMKDRD DMKNLD DMKRSP DMKSEP DMKSPL DMKVSP
SFBCOPY	000018	DMKCKP DMKCKS DMKCQG DMKCSC DMKCSU DMKDRD DMKNLD DMKRSP DMKSPL
SFBDATE	000009	DMKCKP DMKCQG DMKDMP DMKNLD DMKSEP DMKSPL DMKWRM
SFBDIST	000018	DMKCKP DMKCQG DMKCSP DMKCSU DMKNLD DMKSEP DMKSPL
SFBDUMP	000008	DMKCKS DMKCQG DMKDMP DMKDRD DMKNLD DMKSPL DMKVSP

Label	Count	References											
SFBEOF	000014	DMKCKS	DMKDRD	DMKVSP	DMKWRM								
SFBFILID	000035	DMKCKS	DMKCQG	DMKCSP	DMKCST	DMKCSU	DMKDMP	DMKDRD	DMKNLD	DMKRSP	DMKSEP	DMKSPL	DMKVSP
		DMKWRM											
SFBFIRST	000002	DMKCKP	DMKSPL										
SFBFLAG	000090	DMKCKP	DMKCKS	DMKCQG	DMKCQR	DMKCSO	DMKCSP	DMKCSU	DMKDRD	DMKNLD	DMKRSP	DMKSPL	DMKUSO
		DMKVSP	DMKWRM										
SFBFLAG2	000032	DMKCKP	DMKCKS	DMKCSO	DMKCSP	DMKRSP	DMKSEP	DMKSPL	DMKVSP	DMKWRM			
SFBFNAME	000012	DMKCQG	DMKCSP	DMKCSU	DMKNLD	DMKRSP	DMKSEP	DMKSPL					
SFBFTYPE	000003	DMKCQG	DMKNLD	DMKRSP									
SFBHOLD	000011	DMKCSP	DMKSPL	DMKVSP									
SFBINUSE	000013	DMKCKS	DMKCQG	DMKCQR	DMKCSP	DMKCSU	DMKDRD	DMKUSO	DMKVSP	DMKWRM			
SFBLAST	000036	DMKCKP	DMKCKS	DMKDMP	DMKDRD	DMKNLD	DMKRSP	DMKSPL	DMKVSP				
SFBLOK	000067	DMKCKP	DMKCKS	DMKCPI	DMKCQG	DMKCQR	DMKCSO	DMKCSP	DMKCST	DMKCSU	DMKDMP	DMKDRD	DMKEDM
		DMKNLD	DMKRSP	DMKSEP	DMKSPL	DMKUSO	DMKVSP	DMKWRM					
SFBMISC1	000003	DMKVSP											
SFBNOHID	000010	DMKCSP	DMKSPL	DMKVSP									
SFBOPEN	000007	DMKCKS	DMKDRD	DMKVSP	DMKWRM								
SFBORIG	000012	DMKCKP	DMKCPI	DMKCQG	DMKCSU	DMKNLD	DMKRSP	DMKSEP	DMKSPL				
SFBPNT	000055	DMKCKP	DMKCKS	DMKCQG	DMKCQR	DMKCST	DMKCSU	DMKDMP	DMKDRD	DMKEDM	DMKNLD	DMKRSP	DMKSPL
		DMKVSP	DMKWRM										
SFBPURGE	000005	DMKCKP	DMKSPL										
SFBRECER	000031	DMKCKP	DMKCKS	DMKCSO	DMKDRD	DMKRSP	DMKSPL	DMKVSP	DMKWRM				
SFBRECNO	000017	DMKCKS	DMKCQG	DMKNLD	DMKRSP	DMKSEP	DMKVSP						
SFBRECCK	000003	DMKCSO	DMKRSP										
SFBRECS	000018	DMKCKP	DMKCKS	DMKRSP	DMKSPL	DMKVSP	DMKWRM						
SFBRECSZ	000006	DMKNLD	DMKSPL	DMKVSP									
SFBREQUE	000007	DMKCSO	DMKRSP	DMKSPL									
SFBRSTRT	000008	DMKCKS	DMKRSP	DMKSEP	DMKSPL	DMKWRM							
SFBSHOLD	000016	DMKCQG	DMKCQR	DMKCSO	DMKCSP	DMKCSU	DMKRSP	DMKSPL					
SFBSIZE	000037	DMKCKP	DMKCKS	DMKDMP	DMKDRD	DMKEDM	DMKNLD	DMKRSP	DMKSPL	DMKUSO	DMKWRM		
SFBSTART	000056	DMKCKP	DMKCKS	DMKCPI	DMKCST	DMKDMP	DMKDRD	DMKNLD	DMKRSP	DMKSPL	DMKVSP		
SFBTICER	000001	DMKRSP											
SFBTIME	000009	DMKCKS	DMKCQG	DMKDMP	DMKNLD	DMKSEP	DMKSPL	DMKVSP					
SFBTYPE	000014	DMKCQG	DMKDMP	DMKDRD	DMKNLD	DMKRSP	DMKSPL	DMKVSP					
SFBUHOLD	000018	DMKCKS	DMKCQG	DMKCQR	DMKCSP	DMKCSU	DMKDRD	DMKRSP	DMKSPL	DMKVSP			
SFBUSER	000039	DMKCKP	DMKCPI	DMKCQG	DMKCQR	DMKCSP	DMKCST	DMKCSU	DMKDRD	DMKEDM	DMKNLD	DMKRSP	DMKSEP
		DMKSPL	DMKVSP										
SHQBLCK	000011	DMKCKS	DMKCQR	DMKCSP	DMKSPL	DMKWRM							
SHQBSIZE	000011	DMKCKP	DMKCKS	DMKCSP	DMKWRM								
SHQFLAGS	000001	DMKCSP											
SHQPNT	000001	DMKCSP											
SHQSHOID	000005	DMKCQR	DMKCSP	DMKSPL									

Label	Count	References
SHQUSER	000007	DMKCKS DMKCQR DMKCSP DMKSPL
SHRBPNT	000006	DMKCFG DMKPGS DMKVMA
SHRFPNT	000017	DMKCFG DMKPGS DMKVMA
SHRNAME	000018	DMKCFG DMKPGS DMKVMA
SHRPAGE	000013	DMKCFG
SHRSEGCT	000009	DMKCFG DMKPGS DMKVMA
SHRSEGNM	000013	DMKCFG DMKPGS DMKVMA
SHRTABLE	000024	DMKCFG DMKPGS DMKVMA
SHRTSIZE	000003	DMKCFG DMKPGS DMKVMA
SHRUSECT	000009	DMKCFG DMKPGS DMKVMA
SIGMASK	000001	DMKDSP
SILI	000896	DMKACO DMKBSC DMKCCW DMKCKP DMKCNS DMKCPB DMKCPI DMKCSO DMKDAS DMKDDR DMKDGD DMKDIA DMKDIR DMKDMP DMKFMT DMKGRF DMKIOS DMKMCC DMKNLD DMKOPR DMKPAG DMKRGA DMKRGB DMKRNH DMKRSE DMKRSP DMKSAV DMKSEP DMKSPL DMKSSP DMKTAP DMKTDK DMKUCB DMKUCS DMKUDR DMKVCA DMKVCN DMKVDB DMKVDR DMKVMI DMKVSP DMKWRM
SIOCCH	000002	DMKCCH
SKIP	000138	DMKACO DMKCCW DMKCKP DMKCNS DMKCSO DMKCST DMKDAS DMKDDR DMKDIA DMKDMP DMKDRD DMKFMT DMKIOS DMKPAG DMKRSP DMKSAV DMKSEP DMKSPL DMKTAP DMKUNT DMKVCA DMKVCN DMKVDB DMKVMI DMKVSP
SM	000022	DMKCNS DMKCPI DMKFMT DMKIOS DMKNLD DMKPSA DMKRSE DMKVIO DMKVMI
SPLINK	000009	DMKCKS DMKDRD DMKRSP DMKSPL DMKVSP
SPNXTPAG	000023	DMKCKS DMKDRD DMKRSP DMKVSP
SPPREPAG	000013	DMKDRD DMKRSP DMKVSP
SPRECNUM	000015	DMKCKS DMKRSP DMKVSP
SPRMISC	000002	DMKRSP
SPROFCI	000003	DMKMCC DMKMON
SPSIZE	000012	DMKRSP DMKSPL DMKVSP
STARTIME	000011	DMKCKP DMKCPI DMKCQR DMKWRM
SUSPEND	000006	DMKMON
SVCNPSW	000008	DMKCPI DMKPRG DMKPSA DMKTRC
SVCOPSW	000034	DMKPRG DMKPSA DMKTRC
SWPALLCC	000006	DMKPTR DMKVMA
SWPCHG1	000006	DMKCFG DMKCPI DMKMCH DMKPTR
SWPCHG2	000006	DMKCFG DMKCPI DMKMCH DMKPTR
SWPCODE	000003	DMKPAG DMKPTR
SWPCYL	000009	DMKCFG DMKCPI DMKPAG DMKPGS DMKPGT DMKPTR DMKRPA
SWPDPAGE	000006	DMKPAG DMKPGT DMKPTR
SWPFLAG	000074	DMKBLD DMKCCW DMKCDS DMKCFG DMKCPI DMKDGD DMKMCH DMKPAG DMKPGS DMKPGT DMKPRV DMKPTR DMKRPA DMKVMA
SWPKEY1	000013	DMKCFG DMKMCH DMKPGS DMKPRV DMKPTR
SWPKEY2	000003	DMKMCH DMKPTR
SWPPAG	000002	DMKBLD DMKVMA

Label	Count	References										
SWPRECMP	000009	DMKBLD	DMKPGS	DMKPGT	DMKPTR	DMKRPA	DMKVMA					
SWPREF1	000003	DMKPTR										
SWPREF2	000003	DMKPTR										
SWPSHR	000008	DMKCFG	DMKPGS	DMKPRV	DMKPTR	DMKRPA	DMKVMA					
SWPTABLE	000017	DMKBLD	DMKEDM	DMKPGS	DMKVMA							
SWPTRANS	000011	DMKCDS	DMKPAG	DMKPTR	DMKRPA	DMKVMA						
SWPVM	000005	DMKBLD	DMKEDM	DMKPGS	DMKVMA							
SWPVPAGE	000004	DMKPGS	DMKPTR	DMKVMA								
SYNCLOG	000001	DMKCPI										
SYSCYL	000002	DMKCFG										
SYSHRSEG	000003	DMKCFG										
SYSIPLDV	000012	DMKCKS	DMKCPI	DMKDMP	DMKHVD	DMKIOG	DMKWRM					
SYSLOCS	000019	DMKACO	DMKBLD	DMKCFS	DMKCFT	DMKCKP	DMKLOC	DMKLOG	DMKUDR	DMKUSO		
SYSNAME	000047	DMKCFG										
SYSPAGCT	000003	DMKCFG										
SYSPAGIN	000009	DMKCFG										
SYSPAGNM	000016	DMKCFG										
SYSPNT	000003	DMKCFG										
SYSSEGLN	000004	DMKCFG										
SYSSIZE	000002	DMKCFG										
SYSSTART	000002	DMKCFG										
SYSTBL	000003	DMKCFG										
SYSTEM	000123	DMKCFC	DMKCFG	DMKCKS	DMKCNS	DMKCPB	DMKCPI	DMKCPV	DMKCSO	DMKCST	DMKDRD	DMKERM	DMKGRF
		DMKIOF	DMKIOG	DMKMCC	DMKNLD	DMKPSA	DMKPTR	DMKRGA	DMKRGB	DMKRNH	DMKRPA	DMKRSP	DMKSEP
		DMKSNC	DMKSPL	DMKSSP	DMKUDR	DMKVCH	DMKVSP	DMKWRM					
SYSVADDR	000005	DMKCFG										
SYSVOL	000005	DMKCFG										
TBUSY	000010	DMKCPS	DMKMCC	DMKMON								
TEMPR0	000015	DMKCNS	DMKCPI	DMKVCA	DMKVSP							
TEMPR1	000005	DMKVSP										
TEMPR10	000002	DMKCCW										
TEMPR14	000016	DMKCCW	DMKCDS	DMKCPI	DMKIOS	DMKPRG						
TEMPR15	000006	DMKCCW	DMKCDS	DMKCPI	DMKPRG							
TEMPR2	000016	DMKCCW	DMKCPI	DMKCSP	DMKCSU	DMKRGA	DMKSAV	DMKVCA				
TEMPR3	000013	DMKCCW	DMKCPI	DMKCSU	DMKRGA	DMKRGB	DMKVCA					
TEMPR4	000007	DMKCPI	DMKCSU	DMKSAV	DMKVCA							
TEMPR5	000004	DMKCPI	DMKVCA									
TEMPR6	000004	DMKHVC										
TEMPR7	000002	DMKRGA										
TEMPR8	000002	DMKHVC										
TEMPSAVE	000136	DMKCFS	DMKCNS	DMKCPI	DMKCQR	DMKCVT	DMKFRE	DMKGRF	DMKLOG	DMKMID	DMKNET	DMKNLD	DMKPRV
		DMKQCN	DMKRGA	DMKRGB	DMKRNH	DMKSAV	DMKSCH	DMKVIO					

Label	Count	References											
TERMSYS	000009	DMKCCH	DMKEIG	DMKSEV	DMKSIX								
TIMER	000021	DMKCPI	DMKDSP	DMKIOS	DMKMCH	DMKPRG	DMKPSA	DMKPTR	DMKSCH				
TIOCCH	000012	DMKCCH	DMKEIG	DMKSEV	DMKSIX								
TNSCPIDN	000002	DMKIOE	DMKIOF										
TNSDEVAD	000024	DMKIOE	DMKIOF										
TNSKEYN	000005	DMKIOE	DMKIOF										
TNSREC	000006	DMKIOE	DMKIOF										
TNSSNS1	000006	DMKIOE	DMKIOF										
TNSSWS3	000018	DMKIOE	DMKIOF										
TNSVCLID	000004	DMKIOE	DMKIOF										
TODATE	000013	DMKCPI	DMKCVT	DMKDMP	DMKEDM	DMKMID							
TRACBEF	000003	DMKCNS	DMKIOS	DMKVIO									
TRACCURR	000040	DMKCNS	DMKCPI	DMKDSP	DMKFRE	DMKIOS	DMKMCC	DMKMCH	DMKPRG	DMKPSA	DMKRNH	DMKSCH	DMKVIO
TRACEFLG	000006	DMKCPI	DMKMCC										
TRACEND	000020	DMKCNS	DMKCPI	DMKDSP	DMKFRE	DMKIOS	DMKMCH	DMKPRG	DMKPSA	DMKRNH	DMKSCH	DMKVIO	
TRACFLG1	000011	DMKFRE	DMKIOS	DMKMCH	DMKPRG	DMKPSA	DMKSCH						
TRACFLG2	000008	DMKCNS	DMKDSP	DMKIOS	DMKRNH	DMKVIO							
TRACSTRT	000021	DMKCNS	DMKCPI	DMKDSP	DMKFRE	DMKIOS	DMKMCC	DMKMCH	DMKPRG	DMKPSA	DMKRNH	DMKSCH	DMKVIO
TRAC0A	000001	DMKDSP											
TRAC0C	000001	DMKDSP											
TRAC0D	000001	DMKVIO											
TRAC01	000001	DMKPSA											
TRAC02	000002	DMKPSA											
TRAC03	000002	DMKPRG											
TRAC04	000001	DMKMCH											
TRAC05	000001	DMKIOS											
TRAC08	000001	DMKSCH											
TRAC09	000001	DMKSCH											
TRAC10	000001	DMKDSP											
TRAC11	000001	DMKRNH											
TRAC67	000002	DMKFRE											
TRANMODE	000020	DMKCDS	DMKCFG	DMKCPB	DMKDSP	DMKMCH	DMKPRG	DMKPRV	DMKTMR	DMKTRC	DMKVER		
TREXADD	000004	DMKPRG	DMKVAT										
TREXANSI	000005	DMKPGS	DMKTRA	DMKTRC									
TREXBRAN	000012	DMKTRA	DMKTRC										
TREXBUFF	000004	DMKTRC											
TREXCCW	000006	DMKTRA	DMKTRC										
TREXCR9	000003	DMKDSP	DMKPRV	DMKTMR									
TREXCSW	000006	DMKTRA	DMKTRC	DMKVIO									
TREXCTL	000004	DMKTRA											
TREXCTL1	000002	DMKTRC											
TREXCTL2	000007	DMKTRC	DMKVIO										

Label	Count	References
TREXFLAG	000004	DMKTRC
TREXINST	000010	DMKTRA DMKTRC
TREXINTC	000004	DMKPRG DMKPRV
TREXINTL	000001	DMKPRG
TREXIN1	000010	DMKDSP DMKPGS DMKPRV DMKPSA DMKTRA DMKTRC
TREXIN2	000007	DMKDSP DMKTRC
TREXLCNT	000003	DMKTRC
TREXLOCK	000001	DMKCFM
TREXNSI	000012	DMKPGS DMKPRV DMKTRC
TREXPERA	000003	DMKPRG DMKPRV DMKTMR
TREXPERC	000001	DMKPRG
TREXPRNT	000004	DMKTRA DMKTRC
TREXPSW	000002	DMKPRG
TREXRUNF	000004	DMKTRA DMKTRC
TREXSIZE	000007	DMKEDM DMKTRA DMKTRC DMKUSO
TREXSVC1	000002	DMKTRC
TREXSVC2	000002	DMKTRC
TREXT	000020	DMKCFM DMKDSP DMKEDM DMKPGS DMKPRG DMKPRV DMKPSA DMKTMR DMKTRA DMKTRC DMKVIO
TREXTERM	000005	DMKCFM DMKTRA DMKTRC
TREXVAT	000004	DMKTRC
TRQBBPNT	000004	DMKPSA DMKSCH
TRQBFPNT	000013	DMKCFP DMKPSA DMKSCH DMKTMR DMKUSO
TRQBIRA	000011	DMKBLD DMKCFC DMKCFS DMKCPI DMKGRF DMKLOG DMKMCC DMKQCN DMKRGA
TRQBLOK	000058	DMKBLD DMKCDS DMKCFC DMKCFP DMKCFS DMKCPI DMKGRF DMKLOG DMKMCC DMKMID DMKMON DMKPSA DMKQCN DMKRGA DMKRGB DMKSCH DMKTMR DMKUSO
TRQBQUE	000014	DMKSCH DMKTMR
TRQBSIZE	000033	DMKBLD DMKCFC DMKCFM DMKCFS DMKCPI DMKDIA DMKGRF DMKLOG DMKMCC DMKMON DMKQCN DMKRGA DMKUSO
TRQBTOD	000021	DMKCPI DMKMCC DMKMON DMKSCH DMKTMR
TRQBUSER	000011	DMKBLD DMKCFC DMKCFS DMKCPI DMKGRF DMKLOG DMKMCC DMKQCN DMKRGA
TRQBVAL	000030	DMKCDS DMKCFC DMKCFP DMKCPI DMKGRF DMKMCC DMKMID DMKMON DMKPSA DMKQCN DMKRGA DMKSCH DMKTMR
TRUN	000008	DMKDMP DMKMON
TYPBSC	000038	DMKBLD DMKCFM DMKCFT DMKCKP DMKCPI DMKCPS DMKCQG DMKCQP DMKCQR DMKDIA DMKHVC DMKHVD DMKIOS DMKLOG DMKNES DMKNLD DMKQCN DMKRGA DMKUSO DMKVCN DMKVDS DMKWRM
TYPCTCA	000022	DMKCFP DMKCPB DMKCQG DMKDEF DMKDIA DMKDIR DMKIOS DMKSCN DMKVCA DMKVDB DMKVDR DMKVDS DMKVIO DMKVMI
TYPIBM1	000006	DMKDEF DMKDIA DMKDIR DMKNLD DMKSCN
TYPPRT	000052	DMKCFS DMKCKP DMKCQG DMKCQR DMKCSO DMKCSP DMKCST DMKCSU DMKDEF DMKDMP DMKDRD DMKNLD DMKRSP DMKSCN DMKSPL DMKSSP DMKVSP
TYPPUN	000047	DMKCKP DMKCQR DMKCSO DMKCSP DMKCST DMKCSU DMKDRD DMKRSE DMKRSP DMKSCN DMKSEP DMKSPL DMKSSP DMKVSP

Label	Count	References
TYPRDR	000015	DMKCQG DMKCSP DMKCST DMKCSU DMKDRD DMKRSP DMKSCN DMKSPL DMKVMI
TYPTELE2	000005	DMKDEF DMKDIA DMKDIR DMKSCN
TYPTIMER	000004	DMKCQG DMKDIR DMKVSP
TYPTTY	000014	DMKCFT DMKCNS DMKCPV DMKIOF DMKNES
TYPUNDEF	000003	DMKCNS DMKNES DMKNLD
TYPUNSUP	000003	DMKCCW DMKVMI
TYP1050	000006	DMKCNS DMKIOF
TYP1052	000007	DMKDEF DMKDIR DMKLOG DMKSPL DMKVDR DMKVDS
TYP1403	000005	DMKDEF DMKDIR DMKDMP DMKIOF DMKRSE
TYP1442R	000001	DMKCCW
TYP1443	000005	DMKDIR DMKIOF DMKRSE
TYP2305	000059	DMKCCW DMKCFG DMKCKP DMKCPI DMKCPS DMKCQG DMKCQP DMKDAS DMKDDR DMKDEF DMKDGD DMKDIR DMKDRD DMKHVD DMKIOC DMKIOE DMKIOF DMKIOG DMKLOG DMKPAG DMKPGT DMKPTR DMKSAV DMKUNT DMKVDB DMKVDR DMKVDS DMKVER DMKWRM
TYP2311	000017	DMKCCW DMKCQG DMKDDR DMKDGD DMKDIR DMKIOC DMKIOF DMKLNK DMKSCN DMKVDS
TYP2314	000046	DMKCCW DMKCFG DMKCKP DMKCKS DMKCPI DMKCQG DMKDAS DMKDDR DMKDGD DMKDIR DMKDMP DMKLNK DMKNLD DMKPAG DMKPGT DMKSAV DMKSNC DMKSPL DMKSSP DMKTDK DMKVDB DMKVER DMKVIO
TYP2319	000003	DMKDDR DMKDRD DMKHVD
TYP2401	000004	DMKDDR DMKTAP DMKVMI
TYP2415	000003	DMKDDR DMKTAP DMKVMI
TYP2420	000003	DMKDDR DMKTAP DMKVMI
TYP2501	000004	DMKDIR DMKIOF DMKRSE DMKVMI
TYP2520P	000002	DMKRSE
TYP2520R	000001	DMKIOF
TYP2540P	000004	DMKACO DMKDIR DMKRSE DMKSSP
TYP2540R	000008	DMKDIR DMKIOF DMKRSE DMKRSP DMKSSP DMKVMI
TYP2700	000003	DMKIOF DMKNES DMKSCN
TYP2741	000012	DMKCFC DMKCNS DMKCPI DMKIOF
TYP2955	000001	DMKCCW
TYP3066	000012	DMKCPI DMKCPV DMKGRF DMKIOF DMKOPR DMKSSP
TYP3158	000001	DMKDIR
TYP3210	000040	DMKCCW DMKCFP DMKCKP DMKCNS DMKCPB DMKCPI DMKCQG DMKCSP DMKCST DMKDIR DMKHVD DMKIOF DMKSCN DMKSPL DMKSSP DMKVCN DMKVDS DMKVIO DMKVSP
TYP3211	000024	DMKCSO DMKDEF DMKDIR DMKIOE DMKIOF DMKRSE DMKSPL DMKVDR DMKVSP
TYP3215	000004	DMKCFP DMKDIR DMKEDM
TYP3277	000033	DMKCCW DMKCFT DMKCKP DMKCPI DMKCPV DMKDIA DMKDIR DMKGRF DMKHVC DMKHVD DMKSSP DMKVDS
TYP3284	000014	DMKCKP DMKCPV DMKGRF
TYP3330	000060	DMKCCW DMKCFG DMKCKP DMKCKS DMKCPI DMKCQG DMKDAS DMKDDR DMKDGD DMKDIR DMKDMP DMKDRD DMKHVD DMKIOC DMKIOE DMKIOF DMKIOG DMKNLD DMKPAG DMKPGT DMKSAV DMKSNC DMKSSP DMKTDK DMKUNT DMKVDB DMKVER DMKVIO DMKWRM
TYP3340	000057	DMKCCW DMKCFG DMKCKP DMKCPI DMKCQG DMKDAS DMKDDR DMKDGD DMKDIR DMKDMP DMKDRD DMKHVD DMKIOE DMKIOF DMKIOG DMKMSW DMKPAG DMKPGT DMKSAV DMKSPL DMKSSP DMKTDK DMKUNT DMKVDB DMKVER DMKVIO DMKWRM

Label	Count	References											
TYP3350	000069	DMKCCW	DMKCFG	DMKCKP	DMKCKS	DMKCPI	DMKCQG	DMKDAS	DMKDDR	DMKDGD	DMKDIR	DMKDMP	DMKDRD
		DMKHVD	DMKIOE	DMKIOF	DMKIOG	DMKNLD	DMKPAG	DMKPGT	DMKSAV	DMKSNC	DMKSPL	DMKSSP	DMKTDK
		DMKUNT	DMKVDB	DMKVER	DMKWRM								
TYP3410	000007	DMKCCW	DMKDDR	DMKIOE	DMKIOF	DMKTAP							
TYP3411	000001	DMKDDR											
TYP3420	000007	DMKCCW	DMKDDR	DMKIOE	DMKIOF	DMKTAP							
TYP3505	000008	DMKDIR	DMKIOE	DMKIOF	DMKRSE	DMKVSP							
TYP3525	000001	DMKDIR											
TYP3705	000014	DMKBLD	DMKCCW	DMKCKP	DMKCPS	DMKCQP	DMKNES	DMKNLD	DMKRNH	DMKSCN	DMKUSO	DMKVCH	DMKVDS
		DMKWRM											
UC	000086	DMKBSC	DMKCKP	DMKCNS	DMKCPI	DMKDAS	DMKDDR	DMKDIA	DMKDIR	DMKDMP	DMKDSP	DMKFMT	DMKGRF
		DMKHVC	DMKIOE	DMKIOS	DMKMCN	DMKNLD	DMKOPR	DMKRNH	DMKRSE	DMKRSP	DMKSSP	DMKTAP	DMKUNT
		DMKVCA	DMKVCN	DMKVIO	DMKVMI	DMKVSP							
UCASE	000019	DMKCFG	DMKCFM	DMKCFS	DMKCNS	DMKCPI	DMKGRF	DMKLNK	DMKMSW	DMKNLD	DMKQCN	DMKRGA	DMKRNH
UDBFBLCK	000015	DMKDEF	DMKHVD	DMKLNK	DMKLOG	DMKSPL	DMKUDR						
UDBFDASD	000004	DMKUDR											
UDBFSIZE	000011	DMKDEF	DMKHVD	DMKLNK	DMKLOG	DMKSPL							
UDBFVADD	000012	DMKDEF	DMKHVD	DMKLNK	DMKLOG	DMKSPL	DMKUDR						
UDBFWORK	000005	DMKUDR											
UDEVADD	000034	DMKDEF	DMKDIR	DMKLNK	DMKLOG	DMKUDR	DMKVDB	DMKVDS					
UDEVBLCK	000042	DMKDEF	DMKDIR	DMKLNK	DMKLOG	DMKSCN	DMKUDR	DMKVDB	DMKVDS				
UDEVCLAS	000006	DMKDEF	DMKDIR	DMKVDS									
UDEVDASD	000004	DMKDIR	DMKUDR										
UDEVDED	000003	DMKDIR	DMKLNK	DMKLOG									
UDEVDISP	000010	DMKDEF	DMKDIR	DMKLNK	DMKLOG	DMKUDR							
UDEVFTR	000007	DMKDEF	DMKDIR	DMKLNK	DMKSCN	DMKVDS							
UDEVLINK	000007	DMKDIR	DMKLNK	DMKLOG									
UDEVLKDV	000004	DMKDIR	DMKLNK	DMKLOG									
UDEVLKID	000009	DMKDIR	DMKLNK	DMKLOG									
UDEVLM	000023	DMKDIR	DMKLNK										
UDEVLONG	000004	DMKDIR	DMKLNK	DMKLOG									
UDEVLR	000008	DMKDIR	DMKLNK										
UDEVLW	000014	DMKDIR	DMKLNK										
UDEVMODE	000015	DMKDIR	DMKLNK	DMKLOG	DMKVDB	DMKVDS							
UDEVMR	000001	DMKDIR											
UDEVMW	000001	DMKDIR											
UDEVNCYL	000006	DMKDEF	DMKDIR	DMKVDS									
UDEVPASM	000002	DMKDIR											
UDEVPASR	000003	DMKDIR	DMKLNK										
UDEVPASW	000002	DMKDIR											
UDEVR	000002	DMKDIR	DMKLNK										
UDEVRELN	000004	DMKDIR	DMKLNK	DMKSCN	DMKVDS								

Label	Count	References											
UDEVRR	000001	DMKDIR											
UDEVSIZE	000014	DMKDIR	DMKLOG	DMKUDR									
UDEVSPCO	000001	DMKDIR											
UDEVSTAT	000019	DMKDEF	DMKDIR	DMKLNK	DMKICG	DMKVDS							
UDEVTDSK	000006	DMKDEF	DMKDIR	DMKLNK	DMKLOG	DMKVDS							
UDEVTYPC	000024	DMKDEF	DMKDIR	DMKLNK	DMKICG	DMKVDS							
UDEVTYPE	000017	DMKDEF	DMKDIR	DMKLNK	DMKLOG	DMKVDS							
UDEVVSER	000014	DMKDIR	DMKLNK	DMKLOG									
UDEVW	000004	DMKDIR	DMKLNK	DMKVDB									
UDEVWR	000001	DMKDIR											
UDEV3158	000003	DMKDEF	DMKDIR	DMKVDS									
UDIRBLOK	000015	DMKCSP	DMKDEF	DMKDIR	DMKHVD	DMKLNK	DMKLOG	DMKSPL	DMKUDR				
UDIRDASD	000003	DMKDIR	DMKUDR										
UDIRDISP	000008	DMKDEF	DMKDIR	DMKHVD	DMKLNK	DMKLOG	DMKSPL	DMKUDR					
UDIRPASS	000007	DMKCSP	DMKDIR	DMKLOG									
UDIRSIZE	000010	DMKCSP	DMKDIR	DMKUDR									
UDIRUSER	000008	DMKDIR	DMKHVD	DMKLOG	DMKUDR								
UE	000052	DMKCNS	DMKCSO	DMKDDR	DMKDIR	DMKDMP	DMKFMT	DMKGIO	DMKGRF	DMKHVC	DMKMON	DMKRGA	DMKRNH
		DMKRSP	DMKSEP	DMKSSP	DMKVCA	DMKVCN	DMKVIO	DMKVMI	DMKVSP				
UMACACC	000002	DMKDIR	DMKLOG										
UMACACCT	000006	DMKDIR	DMKHVD	DMKLOG									
UMACBLOK	000019	DMKDEF	DMKDIR	DMKHVD	DMKICG	DMKSPL	DMKUDR						
UMACBMX	000002	DMKDIR	DMKLOG										
UMACCDEL	000002	DMKDIR	DMKLOG										
UMACCLA	000001	DMKLOG											
UMACCLEV	000005	DMKDIR	DMKLOG										
UMACCORE	000002	DMKDIR	DMKLOG										
UMACDASD	000002	DMKDIR	DMKUDR										
UMACDISP	000003	DMKDIR	DMKUDR										
UMACDIST	000003	DMKDIR	DMKLOG	DMKSPL									
UMACDVCT	000003	DMKDIR	DMKLOG										
UMACECOP	000002	DMKDIR	DMKLOG										
UMACES	000002	DMKDIR	DMKLOG										
UMACIPL	000002	DMKDIR	DMKLOG										
UMACISAM	000002	DMKDIR	DMKLOG										
UMACLDEL	000002	DMKDIR	DMKLOG										
UMACLEND	000003	DMKDIR	DMKLOG										
UMACMCOR	000002	DMKDEF	DMKDIR										
UMACNSVC	000002	DMKDIR	DMKLOG										
UMACOPT	000014	DMKDIR	DMKLOG										
UMACPRIR	000003	DMKDIR	DMKLOG										
UMACRT	000002	DMKDIR	DMKLOG										

Label	Count	References											
UMACSIZE	000004	DMKDIR											
UMACVROP	000002	DMKDIR	DMKLOG										
USERCARD	000002	DMKACO	DMKCKP										
USERCL	000004	DMKMCC	DMKMON										
VCHADD	000036	DMKCFM	DMKCFP	DMKCPB	DMKCQG	DMKCSP	DMKCSU	DMKDEF	DMKDIA	DMKDSP	DMKEDM	DMKLOG	DMKSCN
		DMKSPL	DMKVCH	DMKVCN	DMKVDB	DMKVDS	DMKVIO	DMKVSP					
VCHBLOK	000050	DMKCFM	DMKCFP	DMKCKP	DMKCPB	DMKCPV	DMKCQG	DMKCSP	DMKCSU	DMKDEF	DMKDIA	DMKDSP	DMKEDM
		DMKLNK	DMKLOG	DMKPRV	DMKSCN	DMKSPL	DMKUSO	DMKVCH	DMKVCN	DMKVDB	DMKVDS	DMKVIO	DMKVSP
VCHBMX	000009	DMKDEF	DMKPRV	DMKVDS	DMKVIO	DMKVSP							
VCHBUSY	000009	DMKCFP	DMKDSP	DMKVIO									
VCHCEDEV	000004	DMKCFP	DMKDSP	DMKVIO	DMKVSP								
VCHCEPND	000010	DMKCFP	DMKDSP	DMKVIO	DMKVSP								
VCHCUINT	000011	DMKCFM	DMKCFP	DMKCPB	DMKDSP	DMKVCN	DMKVIO	DMKVSP					
VCHCUTBL	000027	DMKCFM	DMKCFP	DMKCKP	DMKCPV	DMKCQG	DMKCSP	DMKCSU	DMKDEF	DMKDIA	DMKDSP	DMKEDM	DMKSCN
		DMKSPL	DMKVCH	DMKVDB	DMKVDS	DMKVIO	DMKVSP						
VCHDED	000008	DMKCFP	DMKDEF	DMKLNK	DMKVCH	DMKVDB							
VCHSEL	000014	DMKDEF	DMKDSP	DMKPRV	DMKVDS	DMKVIO	DMKVSP						
VCHSIZE	000006	DMKEDM	DMKLOG	DMKUSO	DMKVDS								
VCHSTAT	000027	DMKCFP	DMKDEF	DMKDSP	DMKLNK	DMKVCH	DMKVDB	DMKVIO	DMKVSP				
VCHTYPE	000019	DMKDEF	DMKDSP	DMKPRV	DMKVDS	DMKVIO	DMKVSP						
VCONADDR	000005	DMKVCN											
VCONBFSZ	000004	DMKVCN	DMKVDR										
VCONBUF	000010	DMKVCN	DMKVDR										
VCONCAW	000006	DMKVCN											
VCONCCW	000014	DMKVCN											
VCONCNT	000006	DMKVCN											
VCONCOMD	000017	DMKVCN											
VCONCTL	000006	DMKCFP	DMKGRF	DMKLOG	DMKRGA	DMKVCN	DMKVDR						
VCONFLAG	000026	DMKVCN											
VCONIDAP	000003	DMKVCN											
VCONRBSZ	000006	DMKCFP	DMKGRF	DMKLOG	DMKRGA	DMKVCN	DMKVDR						
VCONRBUF	000014	DMKCFP	DMKGRF	DMKLOG	DMKRGA	DMKVCN	DMKVDR						
VCONRCNT	000005	DMKGRF	DMKLOG	DMKRGA	DMKVCN								
VCONRSV4	000007	DMKVCN											
VCONSIZE	000005	DMKEDM	DMKVDR	DMKVDS									
VCONWBSZ	000005	DMKCFP	DMKVCN	DMKVDR									
VCONWBUF	000009	DMKCFP	DMKVCN	DMKVDR									
VCONWCNT	000002	DMKVCN											
VCUACTV	000008	DMKCFP	DMKVIO										
VCUADD	000029	DMKCFM	DMKCFP	DMKCPB	DMKCQG	DMKCSP	DMKCSU	DMKDEF	DMKDIA	DMKDSP	DMKEDM	DMKLOG	DMKSCN
		DMKSPL	DMKVCH	DMKVCN	DMKVDB	DMKVDS	DMKVIO	DMKVSP					
VCUBLOK	000039	DMKCFM	DMKCFP	DMKCKP	DMKCPB	DMKCPV	DMKCQG	DMKCSP	DMKCSU	DMKDEF	DMKDIA	DMKDSP	DMKEDM
		DMKLOG	DMKNLD	DMKSCN	DMKSPL	DMKUSO	DMKVCH	DMKVCN	DMKVDB	DMKVDS	DMKVIO	DMKVSP	

Label	Count	References
VCUBUSY	000008	DMKCFP DMKVIO
VCUCEPND	000004	DMKCFP DMKDSP DMKVCN DMKVIO
VCUCHBSY	000004	DMKCFP DMKVIO
VCUCTCA	000013	DMKDEF DMKDSP DMKVDS DMKVIO
VCUCUEPN	000004	DMKCFP DMKVIO
VCUDVINT	000011	DMKCFM DMKCFP DMKCPB DMKDSP DMKVCN DMKVIO DMKVSP
VCUDVTBL	000036	DMKCFM DMKCFP DMKCKP DMKCPV DMKCQG DMKCSP DMKCSU DMKDEF DMKDIA DMKDSP DMKEDM DMKNLD
		DMKSCN DMKSPL DMKUSO DMKVCH DMKVDB DMKVDS DMKVIO DMKVSP
VCUINTS	000012	DMKCFP DMKDSP DMKVIO
VCUSHRD	000006	DMKDSP DMKVCN DMKVDS DMKVIO
VCUSIZE	000007	DMKEDM DMKLOG DMKUSO DMKVDS
VCUSTAT	000024	DMKCFP DMKDSP DMKVCN DMKVIO
VCUTYPE	000016	DMKDEF DMKDSP DMKVCN DMKVDS DMKVIO
VDEVADD	000041	DMKCFM DMKCFP DMKCPB DMKCQG DMKCQP DMKCSP DMKCST DMKCSU DMKDEF DMKDIA DMKDSP DMKEDM
		DMKLOG DMKNLD DMKSCN DMKSPL DMKUSO DMKVCH DMKVCN DMKVDB DMKVDS DMKVIO DMKVSP
VDEVATTN	000008	DMKVCN
VDEVAUCR	000003	DMKCFP DMKLOG DMKVCN
VDEVBLCK	000100	DMKACO DMKCCW DMKCFG DMKCFM DMKCFP DMKCKP DMKCPB DMKCPS DMKCPV DMKCQG DMKCQP DMKCSO
		DMKCSP DMKCST DMKCSU DMKDEF DMKDGD DMKDIA DMKDRD DMKDSP DMKEDM DMKGIO DMKGRF DMKHVC
		DMKHVD DMKIOS DMKLNK DMKLOG DMKNLD DMKQCN DMKRGA DMKSCN DMKSPL DMKTHI DMKTRC DMKUNT
		DMKUSO DMKVCA DMKVCH DMKVCN DMKVDB DMKVDR DMKVDS DMKVER DMKVIO DMKVSP
VDEVBND	000012	DMKCCW DMKCQG DMKCQP DMKDGD DMKVDR DMKVDS DMKVIO
VDEVBUSY	000035	DMKCFM DMKCFP DMKCPB DMKDGD DMKDRD DMKGIO DMKVCN DMKVIO DMKVSP
VDEVCATT	000004	DMKVDB DMKVDR
VDEVCCW1	000028	DMKCFP DMKVCA DMKVCN DMKVSP
VDEVCFCL	000005	DMKVSP
VDEVCFLG	000021	DMKCFP DMKLOG DMKVCN
VDEVCHAN	000017	DMKCFP DMKDGD DMKDSP DMKGIO DMKVCN DMKVIO DMKVSP
VDEVCHBS	000015	DMKCFM DMKCFP DMKVCN DMKVIO DMKVSP
VDEVCLAS	000019	DMKCKP DMKCQG DMKCSP DMKCSU DMKDRD DMKSPL DMKVDS DMKVSP
VDEVCON	000008	DMKCFP DMKEDM DMKGRF DMKLOG DMKRGA DMKVCN DMKVDR DMKVDS
VDEVCONT	000012	DMKCQG DMKCSP DMKDRD DMKVSP
VDEVCOPY	000006	DMKCKP DMKCQG DMKCSP DMKSPL DMKVDS
VDEVCSPL	000009	DMKCQG DMKCSP DMKQCN DMKVCN DMKVDS DMKVSP
VDEVCSW	000108	DMKCFP DMKCSP DMKCSU DMKDSP DMKGIO DMKSPL DMKTRC DMKUNT DMKVCN DMKVIO DMKVSP
VDEVCUE	000012	DMKCFP DMKDSP DMKVIO
VDEVDED	000054	DMKCCW DMKCFP DMKCKP DMKCPB DMKCPV DMKCQG DMKCSO DMKCSP DMKCST DMKDEF DMKDGD DMKDIA
		DMKGIO DMKHVD DMKSCN DMKTRC DMKVDB DMKVDR DMKVDS DMKVER DMKVIO DMKVSP
VDEVDIAG	000008	DMKDRD DMKVSP
VDEVDIAL	000015	DMKCCW DMKCFP DMKDIA DMKNLD DMKVIO
VDEVENAB	000012	DMKCCW DMKCFP DMKCQG DMKDIA DMKVIO
VDEVEOF	000009	DMKCQG DMKCSP DMKVDS DMKVSP

Label	Count	References											
VDEVEXTN	000015	DMKCKP	DMKCQG	DMKCSP	DMKCST	DMKSPL	DMKVDR						
VDEVFCBK	000013	DMKCSO	DMKVDR	DMKVSP									
VDEVFEED	000007	DMKCFP	DMKVSP										
VDEVFLAG	000110	DMKACO	DMKCCW	DMKCFP	DMKCKP	DMKCPV	DMKCQG	DMKCQP	DMKCSP	DMKDEF	DMKDGD	DMKDIA	DMKDSP
		DMKGIO	DMKLNK	DMKNLD	DMKQCN	DMKSCN	DMKUNT	DMKVCN	DMKVDR	DMKVDS	DMKVIO	DMKVSP	
VDEVFOR	000021	DMKCQG	DMKCSP	DMKCST	DMKSPL								
VDEVHOLD	000009	DMKCQG	DMKCSP	DMKSPL	DMKVSP								
VDEVINTS	000031	DMKCFM	DMKCFP	DMKCPB	DMKDSP	DMKVCA	DMKVCN	DMKVIO	DMKVSP				
VDEVIOB	000018	DMKCFP	DMKDGD	DMKDIA	DMKEDM	DMKGIO	DMKHVC	DMKVIO					
VDEVIOCT	000008	DMKIOS	DMKVCA	DMKVCN	DMKVSP								
VDEVIOER	000018	DMKCCW	DMKCFP	DMKDGD	DMKEDM	DMKGIO	DMKVIO						
VDEVKEY	000013	DMKVCN	DMKVSP										
VDEVLINK	000016	DMKDEF	DMKSCN	DMKVDR	DMKVDS								
VDEVNRDY	000027	DMKCFP	DMKCPB	DMKCQG	DMKDIA	DMKVCA	DMKVCN	DMKVDS	DMKVIO	DMKVSP			
VDEVPEND	000018	DMKCFM	DMKCFP	DMKCPB	DMKCSP	DMKCSU	DMKDGD	DMKDSP	DMKGIO	DMKSPL	DMKVCN	DMKVIO	DMKVSP
VDEVPOSN	000008	DMKCCW	DMKDEF	DMKDGD									
VDEVPOST	000007	DMKDGD	DMKDSP	DMKGIO	DMKVIO								
VDEVPURG	000009	DMKCSP	DMKVSP										
VDEVRDO	000011	DMKCCW	DMKCQG	DMKCQP	DMKDGD	DMKLNK	DMKSCN	DMKVDS	DMKVIO				
VDEVREAL	000047	DMKACO	DMKCCW	DMKCFG	DMKCFP	DMKCPB	DMKCQG	DMKCQP	DMKDIA	DMKHVD	DMKIOS	DMKLNK	DMKSCN
		DMKTHI	DMKTRC	DMKUNT	DMKVCA	DMKVDR	DMKVDS	DMKVER	DMKVIO				
VDEVRELN	000029	DMKCCW	DMKCFG	DMKCQP	DMKDEF	DMKDGD	DMKLNK	DMKSCN	DMKUNT	DMKVDR	DMKVDS	DMKVER	
VDEVRSRL	000002	DMKCCW	DMKVDS										
VDEVSAS	000004	DMKCCW	DMKVIO										
VDEVSFLG	000068	DMKCFP	DMKCKP	DMKCQG	DMKCSP	DMKCST	DMKDRD	DMKQCN	DMKSPL	DMKVCN	DMKVDS	DMKVSP	
VDEVSIZE	000016	DMKCQP	DMKCSP	DMKCST	DMKEDM	DMKLOG	DMKSCN	DMKUSO	DMKVDS				
VDEVSNSE	000029	DMKVCN	DMKVSP										
VDEVSPL	000035	DMKCFP	DMKCKP	DMKCPS	DMKCSP	DMKCSU	DMKDRD	DMKEDM	DMKSPL	DMKVDR	DMKVIO	DMKVSP	
VDEVSTAT	000163	DMKCCW	DMKCFM	DMKCFP	DMKCKP	DMKCPB	DMKCPV	DMKCQG	DMKCSO	DMKCSP	DMKCST	DMKCSU	DMKDEF
		DMKDGD	DMKDIA	DMKDRD	DMKDSP	DMKGIO	DMKHVD	DMKSCN	DMKSPL	DMKTRC	DMKVCA	DMKVCN	DMKVDB
		DMKVDR	DMKVDS	DMKVER	DMKVIO	DMKVSP							
VDEVSVC	000018	DMKVSP											
VDEVTDSK	000010	DMKACO	DMKCKP	DMKCPV	DMKCQG	DMKCQP	DMKDEF	DMKVDR	DMKVDS				
VDEVTERM	000011	DMKCQG	DMKCSP	DMKQCN	DMKVCN	DMKVDS							
VDEVTIC	000006	DMKVCN											
VDEVTMAT	000002	DMKACO	DMKVDS										
VDEVTRAN	000003	DMKVCN											
VDEVTYPC	000139	DMKACO	DMKCCW	DMKCFP	DMKCKP	DMKCPB	DMKCPV	DMKCQG	DMKCQP	DMKCSP	DMKCST	DMKCSU	DMKDEF
		DMKDGD	DMKDIA	DMKDRD	DMKEDM	DMKGIO	DMKHVD	DMKLNK	DMKSCN	DMKSPL	DMKTRC	DMKVCN	DMKVDB
		DMKVDR	DMKVDS	DMKVER	DMKVIO	DMKVSP							
VDEVTYPE	000119	DMKCCW	DMKCFP	DMKCKP	DMKCPB	DMKCQG	DMKCSO	DMKCSP	DMKCST	DMKCSU	DMKDGD	DMKDIA	DMKDRD
		DMKEDM	DMKHVD	DMKLNK	DMKSCN	DMKSPL	DMKUNT	DMKVCN	DMKVDR	DMKVDS	DMKVER	DMKVIO	DMKVSP

Label	Count	References											
VDEVUC	000010	DMKCCW	DMKCFP	DMKDGD	DMKGIO	DMKVIO							
VDEVUNIT	000004	DMKVSP											
VDEVUSER	000003	DMKLNK	DMKSCN	DMKVDS									
VDEVVCF	000003	DMKVCN											
VDEVXFER	000021	DMKCKP	DMKCQG	DMKCSP	DMKCST	DMKSPL							
VDEV231B	000008	DMKCCW	DMKCQG	DMKUNT									
VDEV231T	000003	DMKCCW	DMKCQG										
VFCBBLOK	000012	DMKCSO	DMKVSP										
VFCBCHI	000005	DMKVSP											
VFCBCNT	000009	DMKCSO	DMKVSP										
VFCBEOF	000003	DMKVSP											
VFCBFLAG	000009	DMKVSP											
VFCBLOAD	000009	DMKCSO	DMKVSP										
VFCBNDEX	000008	DMKCSO	DMKVSP										
VFCBSIZE	000006	DMKCSO	DMKVDR	DMKVSP									
VMABLOK	000003	DMKCFG	DMKPGS	DMKVMA									
VMACCOUN	000003	DMKHVD	DMKLOG	DMKUSO									
VMACNT	000004	DMKLOG	DMKUSO										
VMACOUNT	000007	DMKHVD	DMKLOG	DMKSPL	DMKUSO								
VMACTDEV	000004	DMKDGD	DMKGIO	DMKTHI	DMKVIO								
VMADSTOP	000008	DMKCFD	DMKCFS	DMKPGS	DMKPSA								
VMAEX	000011	DMKBLD	DMKCFS	DMKMON	DMKSCH								
VMAEXP	000007	DMKCFS	DMKDSP	DMKSCH	DMKUSO								
VMAFPNT	000007	DMKCFG	DMKPGS	DMKVMA									
VMANAME	000005	DMKCFG	DMKPGS	DMKVMA									
VMASHRBK	000002	DMKCFG	DMKVMA										
VMASIZE	000003	DMKCFG	DMKPGS	DMKVMA									
VMASSIST	000006	DMKCFG	DMKPGS	DMKVMA									
VMBADCR0	000004	DMKVAT											
VMBLOK	000303	DMKACO	DMKBLD	DMKCCH	DMKCCW	DMKCDB	DMKCDS	DMKCFC	DMKCFD	DMKCFG	DMKCFM	DMKCFP	DMKCFS
		DMKCFT	DMKCKP	DMKCKS	DMKCNS	DMKCPB	DMKCPI	DMKCPS	DMKCPV	DMKCQG	DMKCQP	DMKCQR	DMKCSO
		DMKCSP	DMKCST	DMKCSU	DMKDEF	DMKDGD	DMKDIA	DMKDRD	DMKDSP	DMKEDM	DMKERM	DMKGIO	DMKGRF
		DMKHVC	DMKHVD	DMKIOE	DMKIOF	DMKIOG	DMKIOS	DMKISM	DMKLNK	DMKLOG	DMKMCC	DMKMCH	DMKMID
		DMKMON	DMKMSG	DMKMSW	DMKNES	DMKNET	DMKNLD	DMKPAG	DMKPER	DMKPGS	DMKPGT	DMKPRG	DMKPRV
		DMKPSA	DMKPTR	DMKQCN	DMKRGA	DMKRGB	DMKRNH	DMKRPA	DMKRSE	DMKRSP	DMKSCH	DMKSCN	DMKSEP
		DMKSNC	DMKSPL	DMKTHI	DMKTMR	DMKTRA	DMKTRC	DMKUDR	DMKUNT	DMKUSO	DMKVAT	DMKVCA	DMKVCH
		DMKVCN	DMKVDB	DMKVDR	DMKVDS	DMKVER	DMKVIO	DMKVMA	DMKVSP				
VMBSIZE	000010	DMKBLD	DMKDIA	DMKEDM	DMKLOG	DMKUSO							
VMCF	000026	DMKCFM	DMKCNS	DMKDIA	DMKDSP	DMKGRF	DMKHVC	DMKLOG	DMKQCN	DMKRGA	DMKRSE	DMKVCN	
VMCFREAD	000006	DMKCFM	DMKDSP	DMKLOG	DMKQCN								
VMCFRUN	000009	DMKCFM	DMKCFS	DMKCQR	DMKDSP	DMKPRG	DMKQCN						
VMCFWAIT	000029	DMKBLD	DMKCFM	DMKCNS	DMKGRF	DMKHVC	DMKLOG	DMKPRG	DMKQCN	DMKRGA	DMKRNH	DMKTRA	DMKTRC

Label	Count	References											
VMCHCNT	000004	DMKLOG	DMKUSO	DMKVDS									
VMCHSTRT	000053	DMKCFM	DMKCFP	DMKCKP	DMKCPV	DMKCQG	DMKCSP	DMKCSU	DMKDEF	DMKDIA	DMKDSP	DMKEDM	DMKLOG
		DMKPRV	DMKSCN	DMKSPL	DMKUSC	DMKVCN	DMKVDS	DMKVIO	DMKVSP				
VMCHTBL	000026	DMKBLD	DMKCFM	DMKCFP	DMKCKP	DMKCPV	DMKCQG	DMKCSP	DMKCSU	DMKDEF	DMKDIA	DMKDSP	DMKEDM
		DMKPRV	DMKSCN	DMKSPL	DMKUSC	DMKVCH	DMKVDB	DMKVDS	DMKVSP				
VMCLASSA	000011	DMKCFC	DMKCFM	DMKCFS	DMKCQR	DMKHVD	DMKMSG	DMKNET	DMKTHI				
VMCLASSB	000011	DMKCFC	DMKCFM	DMKCFS	DMKCQR	DMKHVD	DMKMSG	DMKNET	DMKTHI	DMKVDB			
VMCLASSC	000012	DMKCFC	DMKCFM	DMKCFS	DMKHVD	DMKNET	DMKTHI						
VMCLASSD	000018	DMKCFC	DMKCFM	DMKCFS	DMKCQG	DMKCQR	DMKCSU	DMKNET	DMKTHI				
VMCLASSE	000013	DMKCFC	DMKCFM	DMKCFS	DMKCQR	DMKHVD	DMKNET	DMKTHI					
VMCLASSF	000011	DMKCCW	DMKCFC	DMKCFM	DMKCFS	DMKHVD	DMKIOE	DMKNET	DMKTHI				
VMCLASSG	000006	DMKCFC	DMKCFM	DMKCFS	DMKCQG	DMKNET	DMKTHI						
VMCLASSH	000002	DMKCFC	DMKCFM										
VMCLEVEL	000039	DMKCCW	DMKCFC	DMKCFS	DMKCQG	DMKCQR	DMKCSU	DMKHVD	DMKIOE	DMKLOG	DMKMSG	DMKNET	DMKTHI
		DMKVDB											
VMCOMND	000008	DMKCFC	DMKCFG	DMKCSU	DMKHVC	DMKLNK	DMKLOG	DMKUSO					
VMCOMP	000005	DMKDGD	DMKDSP	DMKGIO	DMKSCH								
VMCPUTMR	000013	DMKPSA	DMKSCH	DMKTMR									
VMCPWAIT	000003	DMKCFM	DMKDSP	DMKSCH									
VMCRDS	000005	DMKMON	DMKTHI	DMKVSP									
VMCUCNT	000004	DMKLOG	DMKUSO	DMKVDS									
VMCUSTRT	000052	DMKCFM	DMKCFP	DMKCKP	DMKCPV	DMKCQG	DMKCSP	DMKCSU	DMKDEF	DMKDIA	DMKDSP	DMKEDM	DMKLOG
		DMKSCN	DMKSPL	DMKVCN	DMKVDS	DMKVIO	DMKVSP						
VMDELAY	000011	DMKCFC	DMKCFM	DMKLOG	DMKQCN	DMKUSO							
VMDISC	000025	DMKCQG	DMKCQP	DMKCQR	DMKLOG	DMKMSG	DMKMSW	DMKPSA	DMKQCN	DMKUSO	DMKVCN		
VMDIST	000010	DMKCKP	DMKCQG	DMKCSP	DMKCST	DMKLOG	DMKSPL						
VMDROP1	000003	DMKSCH											
VMDSP	000011	DMKDSP	DMKPRV	DMKSCH	DMKTMR								
VMDSTAT	000051	DMKCFP	DMKDSP	DMKPRV	DMKPSA	DMKSCH	DMKTHI	DMKTMR	DMKVCN	DMKVIO			
VMDVCNT	000008	DMKCQP	DMKCSP	DMKCST	DMKLOG	DMKSCN	DMKUSO	DMKVDS					
VMDVSTRT	000210	DMKCCW	DMKCFM	DMKCFP	DMKCFT	DMKCKP	DMKCPS	DMKCPV	DMKCQG	DMKCQP	DMKCSP	DMKCST	DMKCSU
		DMKDEF	DMKDGD	DMKDIA	DMKDRD	DMKDSP	DMKEDM	DMKGIO	DMKGRF	DMKHVC	DMKHVD	DMKLOG	DMKQCN
		DMKRGA	DMKSCN	DMKSPL	DMKUSC	DMKVCA	DMKVCN	DMKVDR	DMKVDS	DMKVIO	DMKVSP		
VMECEXT	000045	DMKBLD	DMKCDB	DMKCDS	DMKCFG	DMKCFP	DMKCFS	DMKDSP	DMKEDM	DMKLOG	DMKPRG	DMKPRV	DMKSCH
		DMKTMR	DMKTRC	DMKUSO	DMKVAT	DMKVIO							
VMELIG	000005	DMKSCH	DMKTHI										
VMEPRIOR	000004	DMKMON	DMKSCH										
VMESTAT	000117	DMKCDB	DMKCDS	DMKCFD	DMKCFG	DMKCFP	DMKCFS	DMKCPB	DMKDGD	DMKDSP	DMKEDM	DMKGIO	DMKHVC
		DMKHVD	DMKIOS	DMKLOG	DMKMCH	DMKPGS	DMKPRG	DMKPRV	DMKPSA	DMKPTR	DMKRPA	DMKTMR	DMKTRC
		DMKUDR	DMKVAT	DMKVCN	DMKVIC	DMKVSP							
VMEXTCM	000060	DMKCDB	DMKCDS	DMKCFG	DMKCPB	DMKDGD	DMKDSP	DMKEDM	DMKGIO	DMKHVC	DMKHVD	DMKIOS	DMKMCH
		DMKPRG	DMKPRV	DMKPSA	DMKTMR	DMKTRC	DMKUDR	DMKVCN	DMKVIO	DMKVSP			

Label	Count	References											
VMEXTPND	000005	DMKDSP	DMKPRV										
VMEXWAIT	000035	DMKCFP	DMKDGD	DMKDSP	DMKGIO	DMKHVC	DMKMCH	DMKPRG	DMKPRV	DMKPSA	DMKTHI	DMKTMR	DMKTRA
		DMKTRC	DMKVAT	DMKVCN	DMKVER	DMKVIO	DMKVSP						
VMFBMX	000006	DMKCQG	DMKDEF	DMKLOG	DMKVDS								
VMFPRS	000042	DMKCDB	DMKCDS	DMKCFG	DMKDSP	DMKIOS	DMKMCH	DMKPRG	DMKPSA				
VMFSTAT	000007	DMKCQG	DMKDEF	DMKLOG	DMKUSC	DMKVDS							
VMGENIC	000019	DMKGRF	DMKQCN	DMKRGA	DMKRGB	DMKVCN							
VMGPRS	000065	DMKCDB	DMKCDS	DMKCFG	DMKDGD	DMKDSP	DMKGIO	DMKHVC	DMKHVD	DMKIOS	DMKMCH	DMKPRG	DMKPRV
		DMKPSA	DMKTMR	DMKTRC	DMKVER	DMKVIO							
VMHIPRI	000007	DMKCFS	DMKDSP	DMKSCH									
VMIDLE	000012	DMKCFP	DMKDSP	DMKIOS	DMKVCN	DMKVIO	DMKVMA						
VMINQ	000018	DMKDSP	DMKPRV	DMKSCH	DMKTHI	DMKTMR							
VMINST	000072	DMKHVC	DMKHVD	DMKMON	DMKPRV	DMKPSA	DMKTMR	DMKTRC	DMKVIO	DMKVSP			
VMINVPAG	000017	DMKCDB	DMKCDS	DMKCFP	DMKDSP	DMKMCH	DMKPGS	DMKPRV	DMKPTR	DMKRPA	DMKTMR	DMKTRC	DMKVAT
VMINVSEG	000012	DMKCDB	DMKCDS	DMKDSP	DMKPRV	DMKVAT							
VMIOACTV	000010	DMKCFP	DMKDSP	DMKVIO									
VMIOCNT	000011	DMKACO	DMKDGD	DMKGIO	DMKMON	DMKTHI	DMKVIO						
VMIOINT	000015	DMKCFM	DMKCFP	DMKCPB	DMKDSP	DMKPRV	DMKSCH	DMKVCN	DMKVIO	DMKVSP			
VMIOPND	000014	DMKCFM	DMKCFP	DMKCPB	DMKDSP	DMKPRG	DMKVAT	DMKVCN	DMKVIO	DMKVSP			
VMIOWAIT	000024	DMKCFG	DMKCFP	DMKDGD	DMKDIA	DMKGIO	DMKHVC	DMKIOS	DMKPRG	DMKTHI	DMKVCA	DMKVIO	
VMISAM	000005	DMKCCW	DMKCFS	DMKCQR	DMKLOG								
VMKILL	000019	DMKCFM	DMKCFP	DMKDIA	DMKDSP	DMKLNK	DMKLOG	DMKMCH	DMKMSG	DMKQCN	DMKUSO		
VMLINS	000005	DMKMON	DMKTHI	DMKVSP									
VMLOGOFF	000035	DMKACO	DMKCDB	DMKCFM	DMKCFP	DMKCNS	DMKCPV	DMKDGD	DMKDIA	DMKDSP	DMKGRF	DMKMSG	DMKPGS
		DMKQCN	DMKRGA	DMKRGB	DMKSCN	DMKTRC	DMKUSO	DMKVCA	DMKVCN				
VMLOGON	000032	DMKBLD	DMKCFC	DMKCFM	DMKCKP	DMKCNS	DMKCPI	DMKCPV	DMKGRF	DMKLNK	DMKLOG	DMKMCN	DMKQCN
		DMKRGA	DMKRNH	DMKSCN	DMKUSO								
VMLONGWT	000001	DMKSCH											
VMLOPRI	000004	DMKDGD	DMKDSP	DMKGIO	DMKSCH								
VMMACCON	000005	DMKCFS	DMKCPV	DMKCQR	DMKICG								
VMMADDR	000003	DMKCFS	DMKDSP										
VMMCODE	000013	DMKBLD	DMKCFS	DMKCQR	DMKHVC	DMKLOG	DMKQCN	DMKVMI					
VMMCPENV	000014	DMKCFT	DMKCNS	DMKCQR	DMKGRF	DMKLOG	DMKRGA	DMKRNH					
VMMCR6	000020	DMKCFD	DMKCFS	DMKCQR	DMKDSP	DMKLOG	DMKPSA						
VMMFE	000008	DMKCFS	DMKCPI	DMKCQR	DMKDSP	DMKLOG							
VMMICRO	000008	DMKBLD	DMKCFS	DMKDSP	DMKICG	DMKUSO							
VMMICSVC	000007	DMKCFD	DMKCFP	DMKCFS	DMKLOG	DMKPSA							
VMMIMSG	000012	DMKCFS	DMKCQR	DMKCSU	DMKDEF	DMKLOG	DMKVDB						
VMMLEVEL	000062	DMKBLD	DMKCFG	DMKCFM	DMKCFS	DMKCFT	DMKCNS	DMKCPV	DMKCQR	DMKCSU	DMKGRF	DMKHVC	DMKLOG
		DMKMID	DMKMSG	DMKQCN	DMKRGA	DMKRNH	DMKSPL	DMKUSO	DMKVCN				
VMMLINED	000009	DMKCFS	DMKCQR	DMKGRF	DMKLOG	DMKMSG	DMKRGA	DMKVCN					
VMMLVL2	000012	DMKCFS	DMKCQR	DMKCSU	DMKDEF	DMKLOG	DMKVDB						

Label	Count	References
VMMNOSK	000002	DMKDSP
VMMPROB	000002	DMKDSP
VMMSGON	000014	DMKBLD DMKCFS DMKCQR DMKCSU DMKLOG DMKMID DMKMSG DMKSPL DMKUSO
VMMSHADT	000002	DMKDSP
VMMSTMP	000005	DMKCFM DMKCFT DMKQCN DMKVCN
VMMSVC	000008	DMKCFD DMKCFS DMKCPI DMKCQR DMKLOG DMKPSA
VMMTEXT	000012	DMKBLD DMKCFS DMKCQR DMKHVC DMKLOG DMKQCN DMKVMI
VMM360	000003	DMKCFS DMKLOG
VMNDCNT	000005	DMKDSP DMKPTR
VMNEWCRO	000010	DMKCDB DMKCDS DMKDSP DMKPRV DMKVAT
VMNORUN	000003	DMKDSP DMKSCH
VMNOTRAN	000005	DMKCFP DMKCFS DMKHVC DMKVIC
VMNSHR	000003	DMKCFG DMKPGS
VMOSTAT	000130	DMKCCW DMKCDB DMKCFC DMKCFG DMKCFM DMKCFP DMKCFS DMKCNS DMKCQG DMKCQP DMKCQR DMKCSO
		DMKDIA DMKDSP DMKGRF DMKHVC DMKLNK DMKLOG DMKMCH DMKMSG DMKMSW DMKNES DMKNET DMKPGS
		DMKPRG DMKPSA DMKQCN DMKRGA DMKRSE DMKUSO DMKVCN DMKVDB DMKVDS DMKVMA DMKVSP
VMPAGES	000028	DMKBLD DMKCPI DMKDSP DMKMON DMKPGS DMKPTR DMKRPA DMKSCH DMKTHI DMKVMA
VMPAGEX	000006	DMKCFP DMKCFS DMKCQR DMKPRG DMKVAT
VMPA2APL	000008	DMKCFG DMKCPB DMKGRF DMKHVC DMKHVD DMKRGA
VMPDISK	000007	DMKMON DMKPGT DMKTHI
VMPDRUM	000007	DMKMON DMKPGT DMKTHI
VMPEND	000043	DMKCFG DMKCFM DMKCFP DMKCPB DMKDSP DMKPER DMKPRG DMKPRV DMKPSA DMKSCH DMKTMR DMKTRC
		DMKVAT DMKVCN DMKVIO DMKVSP
VMPERCM	000012	DMKDSP DMKPRG DMKPRV DMKTMR
VMPERPND	000023	DMKDSP DMKPER DMKPRG DMKPRV DMKPSA DMKTMR DMKTRC DMKVAT
VMPFUNC	000010	DMKCFS DMKCQR DMKGRF DMKRGA DMKUSO
VMPGPND	000005	DMKCFP DMKDSP DMKVAT
VMPGPNT	000006	DMKCFP DMKDSP DMKVAT
VMPGREAD	000009	DMKACO DMKMON DMKPTR DMKSCH DMKTHI
VMPGRINQ	000004	DMKMON DMKPTR DMKSCH
VMPGWAIT	000009	DMKDSP DMKPGS DMKPTR DMKRPA DMKTHI
VMPGWRIT	000004	DMKMON DMKPTR DMKTHI
VMPNCH	000005	DMKMON DMKTHI DMKVSP
VMPNT	000049	DMKBLD DMKCKP DMKCPV DMKCQP DMKCQR DMKDIA DMKDSP DMKEDM DMKLOG DMKMID DMKMON DMKMSG
		DMKSCN DMKTHI DMKUSO
VMPRGII	000010	DMKDSP DMKPRG DMKPRV DMKTMR
VMPRGPND	000003	DMKPRG
VMPRIDSP	000008	DMKCFM DMKDSP DMKHVC DMKSCH DMKVCN DMKVIO
VMPSTAT	000089	DMKBLD DMKCCW DMKCDB DMKCDS DMKCFG DMKCFP DMKCFS DMKCPB DMKCQR DMKDSP DMKHVC DMKHVD
		DMKLOG DMKMON DMKPGS DMKPRG DMKPRV DMKPTR DMKSCH DMKTMR DMKTRC DMKUSO DMKVAT DMKVIO
VMPSW	000233	DMKBLD DMKCDB DMKCDS DMKCFC DMKCFG DMKCFM DMKCFP DMKCFS DMKCPB DMKDGD DMKDRD DMKDSP
		DMKGIC DMKHVC DMKHVD DMKIOS DMKLOG DMKMCH DMKMON DMKPRG DMKPRV DMKPSA DMKTMR DMKTRA
		DMKTRC DMKUDR DMKUSO DMKVAT DMKVCN DMKVER DMKVIO DMKVSP

Label	Count	References											
VMPSWAIT	000005	DMKDSP	DMKSCH	DMKTHI									
VMPSWDCT	000011	DMKCFG	DMKLNK	DMKLOG									
VMPXINT	000017	DMKCFP	DMKCPB	DMKDSP	DMKGRF	DMKPRV	DMKRGA	DMKSCH	DMKTMR				
VMQBPNT	000003	DMKSCH											
VMQFPNT	000008	DMKSCH											
VMQLEVEL	000041	DMKBLD	DMKCFS	DMKDGD	DMKDSP	DMKGIO	DMKMON	DMKSCH	DMKTHI	DMKTMR	DMKUSO		
VMQPRIOR	000006	DMKMON	DMKSCH										
VMQSEND	000008	DMKDSP	DMKPSA	DMKSCH									
VMQSTAT	000027	DMKCFG	DMKCFM	DMKCPB	DMKDSP	DMKGRF	DMKHVC	DMKHVD	DMKLOG	DMKQCN	DMKRGA	DMKSCH	DMKVCN
		DMKVIO											
VMQ1	000011	DMKMON	DMKSCH	DMKTHI	DMKTMR								
VMRBSC	000007	DMKQCN	DMKVCN										
VMRDINQ	000003	DMKMON	DMKSCH										
VMREAL	000003	DMKBLD	DMKLOG	DMKPRV									
VMRON	000010	DMKCFS	DMKCQR	DMKDSP	DMKLOG	DMKSCH							
VMRPAGE	000006	DMKCFS	DMKPTR	DMKSCH	DMKUSO								
VMRPRIOR	000002	DMKSCH											
VMRSTAT	000176	DMKACO	DMKBLD	DMKCDB	DMKCFC	DMKCFG	DMKCFM	DMKCFP	DMKCKP	DMKCNS	DMKCPI	DMKCPV	DMKDGD
		DMKDIA	DMKDSP	DMKGIO	DMKGRF	DMKHVC	DMKIOS	DMKLNK	DMKLOG	DMKMCH	DMKMON	DMKMSG	DMKPGS
		DMKPRG	DMKPRV	DMKPSA	DMKPTR	DMKQCN	DMKRGA	DMKRGB	DMKRNH	DMKRPA	DMKSCH	DMKSCN	DMKTHI
		DMKTMR	DMKTRA	DMKTRC	DMKUSO	DMKVAT	DMKVCA	DMKVCN	DMKVER	DMKVIO	DMKVMA	DMKVSP	
VMRUN	000017	DMKDSP	DMKPRV	DMKSCH	DMKTHI	DMKTMR	DMKVCA						
VMSEG	000070	DMKBLD	DMKCCW	DMKCDB	DMKCFG	DMKCFP	DMKCFS	DMKCPI	DMKCPV	DMKCSO	DMKDRD	DMKDSP	DMKEDM
		DMKLOG	DMKPGS	DMKPRV	DMKPSA	DMKPTR	DMKSCH	DMKTMR	DMKTRA	DMKTRC	DMKUSO	DMKVAT	DMKVMA
VMSEGDSP	000002	DMKBLD											
VMSHADT	000009	DMKDSP	DMKPRG	DMKVAT									
VMSHR	000016	DMKCCW	DMKCDB	DMKCFG	DMKDSP	DMKPGS	DMKPSA	DMKUSO	DMKVMA				
VMSHRSYS	000007	DMKCFD	DMKCFG	DMKPGS	DMKVMA								
VMSIZE	000032	DMKBLD	DMKCCW	DMKCDB	DMKCDS	DMKCFD	DMKCFG	DMKCFP	DMKCPI	DMKCPV	DMKDEF	DMKEDM	DMKLOG
		DMKPGS	DMKPTR	DMKUSO	DMKVIO								
VMSLEEP	000009	DMKCFC	DMKCFM	DMKHVC	DMKLOG								
VMSTEALS	000005	DMKMON	DMKPTR	DMKSCH									
VMSTKO	000008	DMKCFM	DMKCQG	DMKCQP	DMKCQR	DMKCST	DMKNET	DMKTHI					
VMSTMPI	000010	DMKCFS	DMKSCH										
VMSTMPT	000007	DMKSCH											
VMSTOR	000022	DMKBLD	DMKCFG	DMKCFP	DMKCFS	DMKCPV	DMKCQG	DMKDEF	DMKHVC	DMKLOG	DMKPGS	DMKTRC	DMKVER
VMSVCPND	000002	DMKPRG											
VMSYSOP	000020	DMKCFM	DMKCNS	DMKCSO	DMKDSP	DMKGRF	DMKLOG	DMKPSA	DMKQCN	DMKUSO	DMKVDB	DMKVDS	DMKVSP
VMTCDEL	000006	DMKCFT	DMKCNS	DMKCQR	DMKLOG								
VMTERM	000039	DMKBLD	DMKCFC	DMKCFM	DMKCFT	DMKCPI	DMKCQG	DMKCQP	DMKCQR	DMKDIA	DMKHVC	DMKHVD	DMKLOG
		DMKMON	DMKMSW	DMKPSA	DMKQCN	DMKRGA	DMKUSO	DMKVCN	DMKVDS				
VMTESCP	000005	DMKCFT	DMKCQR	DMKLOG									

Label	Count	References
VMTIDLE	000004	DMKDSP
VMTIMECN	000004	DMKACO DMKLOG
VMTIMER	000017	DMKCDS DMKDSP DMKLOG DMKPGS DMKPTR DMKSCH
VMTIO	000007	DMKCFP DMKVCN DMKVIO
VMTIONT	000003	DMKDSP
VMTLDEL	000005	DMKCFT DMKCQR DMKLOG
VMTLEND	000016	DMKBLD DMKCFT DMKCNS DMKCQR DMKGRF DMKLOG DMKRGA DMKRGB
VMTLEVEL	000051	DMKCFS DMKCQR DMKDSP DMKLOG DMKPSA DMKSCH DMKTMR
VMTMINQ	000004	DMKDSP DMKSCH
VMTMOUTQ	000030	DMKBLD DMKDSP DMKIOS DMKLOG DMKMCH DMKPRG DMKPSA DMKSCH DMKTMR
VMTMRINT	000004	DMKPSA DMKSCH DMKTMR
VMTODINQ	000001	DMKSCH
VMTON	000006	DMKCFS DMKCQR DMKDSP DMKLOG
VMTPAGE	000004	DMKDSP
VMTRERIN	000023	DMKCDS DMKCFC DMKCFP DMKDSP DMKPRG DMKPRV DMKPSA DMKTRA DMKTRC DMKVIO
VMTRCTL	000052	DMKCDS DMKCFC DMKCFP DMKDSP DMKEDM DMKIOS DMKPER DMKPRG DMKPRV DMKPSA DMKTMR DMKTRA
		DMKTRC DMKUSO DMKVAT DMKVCA DMKVIO
VMTREX	000004	DMKDSP DMKTRA DMKTRC
VMTREXT	000027	DMKCFM DMKDSP DMKEDM DMKPGS DMKPRG DMKPRV DMKPSA DMKTMR DMKTRA DMKTRC DMKUSO DMKVIO
VMTRINT	000002	DMKTRA DMKTRC
VMTRIO	000010	DMKDSP DMKTRA DMKTRC DMKVIO
VMTRMID	000022	DMKBLD DMKCFT DMKCQG DMKCQP DMKCQR DMKDIA DMKHVC DMKHVD DMKLOG DMKPSA DMKQCN DMKRGB
		DMKRNH DMKUSO
VMTRPER	000014	DMKDSP DMKPER DMKPRG DMKPRV DMKTMR DMKTRA DMKUSO DMKVAT
VMTRPRG	000005	DMKDSP DMKPRG DMKTRA DMKTRC
VMTRPRV	000007	DMKDSP DMKPRV DMKTRA DMKTRC
VMTRQBLK	000007	DMKCFS DMKLOG DMKSCH DMKUSO
VMTRSIO	000017	DMKIOS DMKTRA DMKTRC DMKVCA DMKVIO
VMTRSVC	000005	DMKDSP DMKPSA DMKTRA DMKTRC
VMTSEND	000006	DMKDSP DMKPSA DMKSCH
VMTTIME	000268	DMKACO DMKBLD DMKCNS DMKCPS DMKCPV DMKCSU DMKDIA DMKDSP DMKGRF DMKHVC DMKIOS DMKLOG
		DMKMCH DMKMID DMKMON DMKMSG DMKMSW DMKNES DMKNLD DMKPAG DMKPRG DMKPSA DMKPTR DMKQCN
		DMKRGA DMKRGB DMKRNH DMKSCH DMKSPL DMKTHI DMKUSO DMKVCA DMKVCH DMKVDB DMKVMA
VMUPRIOR	000005	DMKCFS DMKCQR DMKLOG DMKMON DMKSCH
VMUSER	000132	DMKBLD DMKCCH DMKCDS DMKCFP DMKCFS DMKCKP DMKCPI DMKCPS DMKCPV DMKCQG DMKCQP DMKCQR
		DMKCSO DMKCSP DMKCST DMKCSU DMKDEF DMKDIA DMKDRD DMKEDM DMKHVD DMKIOE DMKIOF DMKLNK
		DMKLOG DMKMCC DMKMCH DMKMON DMKMSG DMKMSW DMKNES DMKNET DMKNLD DMKQCN DMKRNH DMKSCN
		DMKSPL DMKTHI DMKUSO DMKVCA DMKVCH DMKVDB DMKVDR DMKVER DMKVSP
VMVCR0	000019	DMKCDB DMKCDS DMKCFG DMKCFP DMKCFS DMKDSP DMKLOG DMKPRV DMKTRC DMKVIO
VMVIRCF	000015	DMKCFC DMKCFM DMKHVC DMKLNK DMKLOG DMKNES DMKNET DMKQCN
VMVTERM	000019	DMKBLD DMKCFM DMKCFP DMKCFT DMKCPS DMKDEF DMKGRF DMKHVD DMKLOG DMKQCN DMKRGA
		DMKUSO DMKVDR DMKVDS

Label	Count	References
VMVTIME	000008	DMKACO DMKLOG DMKMON DMKSCH DMKTMR
VMV370R	000058	DMKBLD DMKCDB DMKCDS DMKCFG DMKCFP DMKCFS DMKCPB DMKCQR DMKDSP DMKLOG DMKPRG DMKPRV
		DMKSCH DMKTMR DMKTRC DMKUSO DMKVIO
VMWCNT	000008	DMKPTR DMKRPA
VMWNGON	000010	DMKBLD DMKCFS DMKCQR DMKLOG DMKMSG DMKUSO
VMWSCHG	000003	DMKHVC DMKSCH
VMWSERNG	000003	DMKSCH
VMWSPRCJ	000018	DMKBLD DMKCPV DMKMON DMKSCH DMKTHI
VRALOC	000003	DMKBLD DMKDEF DMKLOG
VSPBUFBK	000016	DMKVSP
VSPBUFSZ	000006	DMKVSP
VSPCAW	000014	DMKDRD DMKVSP
VSPCCW	000118	DMKDRD DMKVSP
VSPDPAGE	000024	DMKDRD DMKVSP
VSPIDACT	000006	DMKVSP
VSPIDAL	000001	DMKVSP
VSPIDASW	000008	DMKVSP
VSPIDAW2	000002	DMKVSP
VSPLCTL	000011	DMKCKP DMKCSP DMKDRD DMKEDM DMKSPL DMKVSP
VSPMISC	000002	DMKVSP
VSPNEXT	000005	DMKVSP
VSPRECNC	000002	DMKVSP
VSPSFBLK	000024	DMKCKP DMKCSP DMKDRD DMKEDM DMKSPL DMKVSP
VSPSIZE	000009	DMKDRD DMKEDM DMKSPL DMKVSP
VSPVPAGE	000015	DMKSPL DMKVSP
VSPXBLCK	000019	DMKCKP DMKCQG DMKCSP DMKCST DMKSPL DMKVDR
VSPXDIST	000006	DMKCSP DMKCST
VSPXLEN	000006	DMKCSP DMKCST DMKVDR
VSPXSIZE	000002	DMKCSP DMKCST
VSPXSPAR	000002	DMKCSP DMKCST
VSPXTAG	000003	DMKCST DMKSPL
VSPXTGLN	000007	DMKCSP DMKCST DMKSPL
VSPXXUSR	000009	DMKCKP DMKCQG DMKCSP DMKCST DMKSPL
VSYSRES	000003	DMKCFG
WAIT	000017	DMKBLD DMKCDS DMKCFM DMKCFP DMKCPI DMKDDR DMKDMP DMKDSP DMKIOG DMKLOG DMKPRV DMKPSA
		DMKTRA DMKTRC DMKUSO
WRITBRK	000001	DMKRNH
WRITEOT	000002	DMKRNH
WRITNRM	000007	DMKRNH
XINTBLCK	000055	DMKCFP DMKCPB DMKDSP DMKGRF DMKRGA DMKSCH DMKTMR
XINTCODE	000017	DMKCPB DMKDSP DMKGRF DMKRGA DMKSCH
XINTMASK	000005	DMKDSP

Label	Count	References
XINTNEXT	000036	DMKCFP DMKCPB DMKDSP DMKGRF DMKRGA DMKSCH DMKTMR
XINTPARM	000002	DMKSCH DMKTMR
XINTSIZE	000013	DMKCFP DMKCPB DMKDSP DMKGRF DMKRGA DMKSCH DMKTMR
XINTSORT	000015	DMKCFP DMKCPB DMKDSP DMKGRF DMKRGA DMKSCH DMKTMR
XOBRCCW1	000002	DMKRSE
XOBRCCW2	000001	DMKRSE
XOBRCCW3	000001	DMKRSE
XOBRCCW4	000001	DMKRSE
XOBREXT	000003	DMKRSE
XOBRFLAG	000013	DMKIOE DMKIOF DMKRSE
XOBRMIS1	000002	DMKRSE
XOBRMIS2	000002	DMKRSE
XOBRRT1	000006	DMKRSE
XOBRRT2	000006	DMKRSE
XOBRRT3	000006	DMKRSE
XOBRRT4	000003	DMKRSE
XOBRRT5	000007	DMKRSE
XOBRRT6	000006	DMKRSE
XOBRSIZE	000002	DMKRSE
XOBRSTAT	000012	DMKRSE
XOBRT1	000011	DMKIOE DMKIOF DMKRSE
XOBRT2	000001	DMKRSE
XOBRT3	000007	DMKIOE DMKIOF DMKRSE
XOBR010	000003	DMKIOE DMKIOF DMKRSE
XOBR150	000004	DMKIOE DMKIOF DMKRSE
XOBR180	000003	DMKIOE DMKIOF DMKRSE
XOBR512	000009	DMKIOE DMKIOF DMKRSE
XPAGNUM	000047	DMKBLD DMKCCW DMKCDB DMKCDS DMKCPI DMKDGD DMKDRD DMKFRE DMKHVC DMKHVD DMKPGS DMKPSA
		DMKPTR DMKUNT DMKVMA
XRIGHT16	000028	DMKCCW DMKCDB DMKCFG DMKCPI DMKCQP DMKDAS DMKDSP DMKERM DMKMSG DMKOPR DMKRNH DMKSEV
		DMKSIX DMKSSP DMKTRA DMKVAT DMKVCN
XRIGHT24	000007	DMKCCW DMKCFP DMKHVD DMKPSA
XTNDLOCK	000011	DMKDSP DMKFRE DMKGRF DMKPAG DMKPTR DMKRGA DMKVIO
X2048BND	000015	DMKCCW DMKDRD DMKHVD DMKPSA DMKTRC DMKUNT
X40FFS	000007	DMKCFG DMKCFS DMKCFT DMKCPB DMKCPS DMKCPV DMKNLD
Y0	000011	DMKDMP DMKDSP DMKIOS DMKMCH DMKPRG DMKPSA
Y2	000011	DMKDMP DMKDSP DMKIOS DMKMCH DMKPRG DMKPSA
Y4	000011	DMKDMP DMKDSP DMKIOS DMKMCH DMKPRG DMKPSA
Y6	000011	DMKDMP DMKDSP DMKIOS DMKMCH DMKPRG DMKPSA
ZEROES	000119	DMKACO DMKBLD DMKBSC DMKCCW DMKCDB DMKCDS DMKCFD DMKCFP DMKCFS DMKCFT DMKCKS DMKCPB
		DMKCPI DMKCPS DMKCPV DMKCQG DMKCQP DMKCQR DMKCSO DMKCSP DMKCST DMKCSU DMKDAS DMKDIA
		DMKDMP DMKDRD DMKDSP DMKGRF DMKIOC DMKIOE DMKIOF DMKLNK DMKLOG DMKMCC DMKMCH DMKMON
		DMKMSW DMKNET DMKPSA DMKPTR DMKRNH DMKRSE DMKSCH DMKSCN DMKSPL DMKTAP DMKTMR DMKTRC
		DMKUDR DMKUNT DMKUSO DMKVCA DMKVDB DMKVDR DMKVDS DMKVSP DMKWRM

RSCS Module	BALR to Module	At Label	Comments
DMTAKE	DMTDSP	TAKEXIT	Resumes dispatching; processing of a TAKE request is complete.
	DMTPST	TAKEMUTE	Signals a task that it must process a TAKE request.
	DMTQRQ	TAKEMUTE	Frees a GIVE element.
DMTASK	DMTDSP	TAEXIT	Resumes dispatching; processing of a task request has completed.
	DMTPST	TAGPURGE	Signals the termination of a task.
	DMTQRQ	TAFREEOK	Frees a terminated task element.
	DMTQRQ	TAGPURGE	Frees a terminated GIVE element.
	DMTQRQ	TAMAKE	Gets a queue element for a new task.
	DMTQRQ	TAQPTEST	Frees requested elements for a terminated task.
	DMTQRQ	TASQTEST	Frees an I/O element associated with a task being purged.
DMTASY	DMTDSP	ASEXIT	Resumes dispatching; processing of an asynchronous exit request has completed.
	DMTQRQ	ASQEND	Gets a free queue element; free a terminated queue element.
	DMTQRQ	ASQGOT	Gets a free queue element; free a terminated queue element.
DMTAXS	DMTAKE	AXSACCPT	Takes a request for DMTAXS services from another task.
	DMTASY	AXSIGSET	Requests an asynchronous exit for task asynchronous alerts.
	DMTASY	AXSIGSET	Requests an asynchronous exit for reader X'001'.
	DMTCOM	GETLINK	Gets a link table entry.
	DMTCOM	OPENIRTY	Gets a page of main storage.
	DMTCOM	OPENOLNK	Gets a page of main storage.
	DMTCOM	TODEBCD	Converts a S/370 format TOD to EBCDIC date and time.
	DMTGIV	MSGDO	Gives a message element to DMTMGX for processing.
	DMTPST	AXSALRT1	Signals acceptance of a command to process.
	DMTPST	AXSASYIO	Signals arrival of a request for an asynchronous exit.

RSCS Module	BALR to Module	At Label	Comments
DMTAXS (cont)	DMTSIG	ACCEFIND	Alerts a line driver task that a newly arrived file has been accepted.
	DMTSIG	CHANDONE	Alerts a line driver task.
	DMTWAT	AXSCYCLE	Waits for a request for DMTAXS services.
	DMTWAT	MSGDO	Waits until processing by DMTGIV has completed.
DMTCMX	DMTCOM	QYOLINK	Finds a link table entry.
	DMTCOM	TODEBCD	Converts a S/370 TOD to EBCDIC date and time.
	DMTCRE	STALNGOT	Creates a line driver task, as specified in the START command.
	DMTMGX	CMXDCIT	Writes a message resulting from command processing.
	DMTMGX	CMXM001	Writes a message showing the number of free pages in storage.
	DMTMGX	CMXM003B	Writes a message showing the command currently being executed by RSCS.
	DMTMGX	DISCHARG	Writes a message resulting from DISCONN command processing.
	DMTMGX	QYM654	Writes a message resulting from QUERY command processing.
	DMTMGX	QYM655	Writes a message resulting from QUERY command processing.
	DMTMGX	QYSYMSG	Writes a message resulting from command processing.
	DMTREX	DISCONN	DIAGNOSE instruction entry to CP console function.
	DMTREX	DISCHARG	DIAGNOSE instruction entry to CP console function.
	DMTSIG	CMXALRDY	Alerts a task for command processing.
	DMTSIG	STACREAT	Alerts DMTLAX to validate a line address used in a START command.
DMTCOM	DMTDSP	MFIXIT	Requests dispatching of a task for which a message has been stacked for transmission.
	DMTDSP	MFOXIT	Requests dispatching of a task for which a message has been unstacked for transmission.
	DMTSTO	GETPTRY	Requests main storage.

RSCS Module	BALR to Module	At Label	Comments
DMTCRE	DMTASK	CREQTASK	Requests the supervisor to start a new task.
	DMTIOM	CFILDOIO	Requests the I/O manager to read one DASD block from a file on a CMS-type system disk.
	DMTSTO	CRETRYIT	Requests main storage for the creation of a task.
	DMTWAT	CFILDOIO	Waits for a read I/O request to complete.
DMTEXT	DMTDSP	EXTGC	Resumes dispatching; processing of an external interruption is complete.
DMTGIV	DMTDSP	GIVEXIT	Resumes dispatching; processing of a GIVE request is complete.
	DMTPST	GIVESNIF	Signals a task to begin processing a GIVE request.
	DMTQRQ	GIVESCAN	Gets a free queue element.
DMTINI	DMTDSP	INIQDONE	Dispatches the first task.
	DMTQRQ	INIQDONE	Initializes the queue of free elements.
DMTIOM	DMTDSP	IODISPCH	Resumes dispatching; processing of an I/O request is complete.
	DMTPST	IONORMAL	Signals completion of an I/O event.
	DMTPST	IOPUNT	Signals an error on a request for a queue element.
	DMTQRQ	DMTIOMRQ	Gets an element for an I/O request.
	DMTQRQ	IODISMIS	Frees an element used for a SENSE request.
	DMTQRQ	IONORMAL	Frees an element used in an I/O request.
	DMTQRQ	IOUNITCK	Gets an element for a SENSE request.
DMTLAX	DMTASY	LAXINIT	Sets up an asynchronous exit for DMTLAX.
	DMTWAT	LAXHANG	Terminates DMTLAX.
DMTMGX	DMTCOM	MGXBUILT	Gets a link table entry.
	DMTCOM	MGXTOLOC	Stacks a message.
	DMTREX	MGXNCPR	Writes a message to a local VM/370 userid.
	DMTREX	MGXNOVM	Writes a message to the VM/370 operator.
	DMTSIG	MGXBUILT	Alerts an originating task that a message has been handled.

RSCS Module	BALR to Module	At Label	Comments
DMTNPT	DMTASY	NPTNOPAS	Sets up an asynchronous interrupt for DMTNPT.
	DMTCCM	AXSMENQ	Enqueues a message on the message stack for processing by DMTMGX.
	DMTCCM	MSG2780	Unstacks a message for transmission to a remote station.
	DMTCCM	NPTNCPAS	Gets a page of storage for use as DMTNPT buffers.
	DMTCOM	TODEBCD	Converts S/370 TOD to EBCDIC date and time.
	DMTGIV	AXSGET	Requests DMTAXS to open a file.
	DMTGIV	AXSPURGE	Requests DMTAXS to purge a file.
	DMTGIV	COMMANDS	Passes a command element to DMTREX for processing by DMTCMX.
	DMTGIV	KLOGIT	Requests DMTAXS to open the LOG file for output.
	DMTGIV	LINEDROP	Requests DMTAXS to close a file.
	DMTGIV	LOGCLOSE	Requests DMTAXS to close the LOG file for output.
	DMTGIV	MSG1	Passes a message element to DMTMGX for processing.
	DMTGIV	PUTCLS1	Requests DMTAXS to close a file for output.
	DMTGIV	PUTOPEN	Requests DMTAXS to open a file for output.
	DMTGIV	TASKILL	Requests DMTREX to terminate the requesting NPT line driver.
	DMTIOM	LOGCONT1	Requests an I/O operation for the LOG routine.
	DMTIOM	LOGPRINT	Prints a LOG message.
	DMTIOM	XECUTE	Requests an I/O operation (general usage by DMTNPT).
	DMTPST	AXSALRT1	Signals that DMTNPT accepted a command.
	DMTWAT	AXSGET	Waits for a request to open a file to complete processing.
	DMTWAT	AXSPURGE	Waits for a request to purge a file to complete processing.
	DMTWAT	COMMANDS	Waits for DMTCMX to process a command.
	DMTWAT	KLOGIT	Waits for completion of a request to open the LOG file for processing.
	DMTWAT	LINEDROP	Waits for a request to close a file to complete processing.
	DMTWAT	LOGCLOSE	Waits for a request to close the LOG file when processing is complete.
	DMTWAT	LOGCONT1	Waits for an I/O operation to complete logging processing.
	DMTWAT	MSG1	Waits for message processing to complete.

RSCS Module	BALR to Module	At Label	Comments
DMTNPT (cont)	DMTWAT	PUTCLS1	Waits fcr a request to close a file to complete processing.
	DMTWAT	PUTOPEN	Waits for completion of a request to open a file for processing.
	DMTWAT	TASKILL	Waits for task termination processing to complete.
	DMTWAT	XECQWAIT	Waits for an I/O operation to complete.
DMTREX	DMTAKE	REXACCPT	Accepts a request to process a VM/370 file.
	DMTASK	QUIESE	Requests task termination.
	DMTASK	TERTKILL	Requests task termination.
	DMTASY	REXICGOT	Initializes an asynchronous exit.
	DMTCOM	REXFLUSH	Requests DMTMGX to write any queued messages.
	DMTCOM	REXOUTRY	Removes a message for the message stack and write it to the console.
	DMTCRE	REXICGOT	Creates the tasks DMTAXS and DMTLAX.
	DMTDSP	REXDQUIT	Terminates dispatching due tc program check.
	DMTDSP	REXHEXIT	Resumes dispatching after program check processing.
	DMTIOM	REXCONON	Requests an I/O operation (console write).
	DMTIOM	REXFCONF	Requests an I/O operation (ccnsole write).
	DMTIOM	REXQUERY	Requests an I/O operation (console read).
	DMTMGX	MSG	Passes a message element tc DMTMGX for processing.
	DMTMGX	TERMSET	Writes a task terminated message.
	DMTPST	REXASYN	Signals a console attention.
	DMTPST	REXHALT	Signals that DMTREX is undispatchable due to program check.
	DMTWAT	QUIESE	Waits for a task to terminate.
	DMTWAT	QUICK	Waits for task I/O to terminate.
	DMTWAT	REXSWAIT	Waits for a console write to complete.
	DMTWAT	REXWAIT	Waits for completion of an event.
DMTSIG	DMTDSP	ALSCAN ALNOGO	Resumes dispatching; processing of an alerted task has completed.
DMTSMI	DMTASY	SETNCBUF	Sets up an asynchroncus exit for DMTSML.
	DMTCOM	ASYNENQ	Stacks a message to be transmitted by DMTSML.
	DMTCOM	BUFSDONE	Gets a page of storage for DMTSML I/O tasks.
	DMTCOM	IBLDBUFS	Gets a page of storage for DMTSML TP buffers.

RSCS Module	BALR to Module	At Label	Comments
DMKSML (cont)	DMTCCM	MSGPBOC1	Unstacks a message fcr transmission tc a remote station.
	DMTCOM	TODEBCD	Converts S/370 TOD to EBCDIC date and time.
	DMTGIV	AXS	Requests services of DMTAXS for the SML line driver task.
DMTGIV		KLOGIT	Requests DMTAXS tc open a LOG printer.
DMTGIV		LOGCLOSE	Requests DMTAXS to close the LOG printer.
	DMTGIV	AXSGET	Requests DMTAXS to give a file fcr transmission.
	DMTGIV	AXSPURGE	Requests DMTAXS to purge a file.
	DMTGIV	EOJ	Requests termination of the SML line driver task.
	DMTGIV	MSG1	Gives a message to DMTMGX for processing.
	DMTGIV	WGET1A	Requests that a message be written to the RSCS console; pass a command tc DMTREX.
	DMTIOM	I27XXIO	Performs the initial I/O operation for the SML line driver task.
	DMTIOM	JOUT1	Requests an I/O operation; set up job processing controls.
	DMTIOM	PCONT2 PLINE	Requests an I/O operation (set up printer controls.
	DMTIOM	RSIO	Requests a start I/O for the DMTSML TRACE function.
	DMTIOM	UOUT2	Requests an I/O operation (sets up punch controls).
	DMTIOM	WRLOG1	Requests an I/O operation (log an I/O operation).
	DMTPST	ASYNRET	Posts the reader synch lock.
	DMTWAT	ALLCHK	Waits for the DMTSML synch lock to be posted (waits for a request tc process).
	DMTWAT	AXS	Waits fcr completion of an event by DMTAXS.
	DMTWAT	AXSGET	Waits for DMTAXS to GIVE a file for transmission.
	DMTWAT	AXSPURGE	Waits for DMTAXS to purge a file.
	DMTWAT	EOJ	Terminates the SML line driver task by issuing a terminal WAIT request.
		KLOGIT	Waits for DMTAXS to open a LOG printer.
		LOGCLOSE	Waits for DMTAXS to clcse a LOG printer.
	DMTWAT	MSG1	Waits until GIVE to DMTMGX is complete.
	DMTWAT	RISIC1	Waits for initial SIO for the DMTSML line driver to complete.
	DMTWAT	WGET1A	Waits until message prccessing has completed.
	DMTWAT	WRLOG1	Waits for I/O logging to complete.

RSCS Module	BALR to Module	At Label	Comments
DMTSTC	DMTDSP	MAINDONE	Resumes dispatching; a request for a page of storage has been processed.
DMTWAT	DMTDSP	WAITGO	Resumes dispatching; processing of a WAIT request has completed.

Module Name	Entry Points	Function
DMTAKE	DMTAKE	Contains the supervisor service that supplies task programs with the receiver interface tc GIVE requests issued by other tasks. A single CALL causes DMTAKE to first respond to the previously supplied GIVE request and then supply a new GIVE request to the task for its processing.
DMTASK	DMTASK	A service rcutine that creates new tasks and deletes existing tasks executed by the MSUP dispatcher. The entry to DMTASK is via a BAL instruction from task programming. Any entry into DMTASK causes the calling task's execution to be suspended through the freeze SVC function.
DMTASY	DMTASY	A supervisor service module that starts and ends asynchronous exit requests for task prcgrams. This routine handles asynchronous exit requests for asynchronous exit requests for I/O interrupticns, and ALERT exit requests.
DMTAXS	DMTAXS	Controls the interface of the line drivers to the VM/370 spool file system, enqueues files for transmission and processes commands that manipulate spool files.
	AXSINIT	Initializes the AXS task.
	AXSCYCLE	Looks for work to do by examining the synch locks asscciated with the AXS task.
	REQXEQ	Scans the request table for a match and branches to the to the apprcpriate subroutine, depending on the request code.
	CMDPROC	Executes AXS commands from the command buffer passed on by an ALERT exit from DMTREX.
	OPENIN	Starts spocl file processing.
	CLOSECUT	Ends processing for output files.
	MSG	Sets the MSG request element. A CALL GIVE instruction passes the MSG request element tc the message manager. The code associated with other entry points in this module format the MSG element variable areas in various ways and exit finally to MSG.

Module Name	Entry Points	Function
DMTAXS (cont.)	HEXGET	Converts and validates a hex string.
	DECGET	Converts and validates a decimal string.
	DECPUT	Converts a hex fullword to decimal and generates an EBCDIC representation of it, suppresses leading zeroes to a minimum count, which is optionally supplied by the caller.
	TODS370	Converts EBCDIC to the System/370 TOD value.
	TODEBCD	Converts System/370 TOD to an EBCDIC date and time.
	GSUCCESS	Gets inactive successor spool file.
	ACCEPT	Inspects newly arrived files.
	UNPEND	Brings in a link's pending tags.
	GETROUTE	Gets a routing table entry.
	GETLINK	Gets link table entry.
	GETSLOT	Gets a free tag queue element.
	FREESLOT	Returns a tag queue element.
	TAGGEN	Builds a file tag from hypervisor information.
	TAGPLACE	Sets a file tag into a link queue immediately before the first tag of numerically higher priority (lower real priority).
	FILSELEC	Selects a file to be read from a link queue.
	TAGFIND	Locates a file with spoolid matching the one supplied by the caller, within the internal file tag queues.
	DEFINE	Gets a virtual spool device.
	DETACH	Undefines a virtual spool device.
	VCHANGE	Changes VM/370 file attributes.
	VCLOSE	Issues the VM/370 CLOSE command for a device.
	VPURGE	Purges an inactive reader file from the VM/370 spool.
	VSPOOL	Sets VM/370 virtual spool device options.
	VTAGD	Sets a VM/370 tag for a virtual spool device
	VTAGF	Sets a VM/370 tag for an inactive spool file.
DMTCMX	DMTCMX	This module is part of the REX system control task. DMTCMX is called in several places in DMTREX, which is the main REX control routine. DMTCMX accepts an EBCDIC string and executes the RSCS command that the string represents.
	CMXHIT	Calls the necessary individual command proccessing routine.
	CMSALERT	Passes a command element to another task via the ALERT task-to-task communications interface.
	KEYWDGET	Decodes the next keyword on the input command line.

Module Name	Entry Points	Function
DMTCMX (cont.)	LTABGET	Finds the link table entry implied by the first keyword in the command line described by the calling routine's register parameters.
	HEXGET	Converts and validates a hex string.
	DECPUT	Converts a hex fullword to decimal and generates an EBCDIC representation of it. It suppresses leading zeros to a minimum count, which is optionally supplied by the calling routine.
	FILGET	Locates a file, within the internal file tag queues, with a spoolid matching that supplied by the calling routine.
	TODEBCD	Converts a System/370 format TOD to EBCDIC data and time.
	PARMGET	Scans an EBCDIC line and frames the next parameter on the line.
DMTCOM	DMTCOM	Contains various reentrant routines used by RSCS tasks.
	GETLINK	Scans the link table chain and returns a link table address.
	GETPAGE	Gets a free page of main storage.
	FREEPAGE	Returns a page of main storage.
	MPI	Stacks message elements in a LIFO stack for later processing. If no room is available in the current page, a new page is fetched if there are at least five free pages remaining. If five free pages are not remaining, an error condition is returned. All tasks except REX are allowed only three pages of storage to stack messages.
	MPC	Unstacks message elements from the message queue for this task. If none are queued an error condition is returned.
	GTODEBCD	Converts a System/370 format TOD to EBCDIC data and time.
DMTCRE	DMTCRE	Creates new tasks under MSUP.
	CMSFILCH	Reads one dASD block from a CMS disk.
	CMSCPEN	Does initial work prior to reading a CMS file.
	CMSGET	Gets the next CMS file item.

Module Name	Entry Points	Function
DMTDSP	DMTDSP	This module is the MSUP dispatcher. It is entered when an exit occurs from supervisor functions that were entered following an interruption or that issued the freeze SVC function. DMTDSP must be entered with all PSW masks off (except for the machine check mask).
DMTEXT	DMTEXT	This module is the MSUP external interruption handler. DMTEXT receives control directly on an external interrupt and saves the status of the executing task if one was interrupted.
DMTGIV	DMTGIV	This is a supervisor service routine that enqueues GIVE requests from tasks to be delivered to other tasks by DMTAKE.
DMTINI	DMTINI	Receives control after initial loading of RSCS, and performs general initialization functions that are common to all parts of RSCS. DMTINI writes a copy of the initial load to DASD, acording to operator instructions, when RSCS is initial program loaded from the generation IPL deck. When itital program loaded from disk, DMTINI finishes reading the saved RSCS load. When IPL disk reading or writing is comlete, DMTINI initializes RSCS storage areas.
DMTION	DMTION	This module contains both the MSUP I/O interrupt handler and the task I/O service routine. The I/O service provided by DMTION to the task programs includes sequential subchannel scheduling, channel program execution, automatic sense execution on unit check when requested, return of all pertinent information regarding the execution of the channel program, and notification via a POST upon completion of the channel program.
DMTLAX	DMTLAX	This routine is the line allocation task for RSCS. The major part of this routine functions as an asynchronous exit being alerted by DMTREX.

Module Name	Entry Points	Function
DMTMAP	DMMAP	Describes the non-fixed address MSUP status storage areas in main storage. This module contains no executable code.
DMTMGX	DMTMGX	Takes a message request buffer and constructs the message from the information in that buffer and the message text found in DMTMSG.
DMTMSG	DMTMSG	Contains a list of error messages to be used externally by DMTMGX. This module contains no executable code.
DMTNPT	DMTNPT	This module is a line driver that provides support for the 2770, 2780, 3770, and 3780 nonprogrammable terminals.
	NPTGET	Maintains a cyclic control of the DMTPT task on both sending and receiving operations.
	SENDOFF	Sends the BSC end-of-transmission character (EOT) on the line to the remote terminal.
	BUFFINIT	Initializes the line output buffer with the correct BSC character set, depending on the type of output file and and features available at the terminal.
	XECUTE	Requests the supervisor to execute I/O operations. After starting the I/O operations, XECUTE waits for either a command to be entered or the completion of the requested I/O operation.
	LINEIO	Executes (by calling XECUTE) I/O operations on the BSC line and checks the final state. LINEIO then sets the IOERR flag in the DEVFLAG byte.
	GETBLOCK	prepares the line output buffer to be transmitted to the remote terminal.
	GETVRFY	Analyses the response obtained from each buffer transmission and takes the appropriate error action.
	PUTBLOCK	Deblocks received TP buffers and writes the deblocked record to the VM/370 spool file system.
	PUTVRFY	Verifies the content of each received TP buffer and constructs an appropriate reply if the buffer is found in error.

Module Name	Entry Points	Function
DMTNPT (cont.)	COMMANDS	Passes commands received from the remote card reader to the RSCS command processor for execution.
	CMDPROC	Executes commands passed to it in the CMDRESP buffer after an ALERT from DMTREX indicates a command has been entered.
	MSGPROC	Unstacks messages from the task MSG queue and transmits them to the remote terminal printer. Prepares and sends requests to the specialized task REX to write console messages.
	MSG	Prepares and sends requests to the specialized task REX to write console messages.
	HEADPREP	Provides, one record after the other, the separator and header for print files and the header card for punch files.
	MAKEBLOC	Saves the caller's registers for a call to VMSB2CP. Upon return from VMSB2CP, it sets the return code and returns to the original caller.
	VMSB2CP	Deblocks the VM/370 spool page buffers into an unpacked buffer (PACKBLK).
	AXSGET	Requests the specialized task AXS to open, close, and delete the spool files that the NPT task is processing.
	TODEBCD	Converts System/370 TOD to EBCDIC date and time.
	PARMGET	Scans character strings to find delimiter characters.
	NPTINIT	Initialization routine for NPT.
	NPTLINK	NPT sign-on routine.
	NPTERROR	Writes the terminal I/O error message and terminates tha task.
	NPTTERM	Terminates the NPT task.
DMTPST	DMTPST	A service routine that may be called from anywhere in RSCS. DMTPST signals the completion of an event by posting the event's associated synch lock. This routine is entirely reentrant and does not change the state of running PSW.
DMTQRQ	DMTQRQ	Manages the MSUP supervisor status queue for other MSUP functions. DMTQRQ is for use within the supervisor and be entered with all PSW masks off (except machine check).

Module Name	Entry Points	Function
DMTREX	DMTREX	This routine is the controlling supervisor task and together with DMTCMX, DMTMGX, DMTSYS, DMTCOM, DMTMSG, and DMTCRE make up the REX supervisor task.
	REXINIT	Performs the initialization for the DMTREX task.
	REXCYCLE	Monitors a list of synch locks when looking for work for DMTREX to perform.
	REXPCHEX	Processes program checks.
	REXITERM	Entered when RSCS initialization fails. Issues the initialization failure message, dumps the contents of main storage, types any remaining messages, and loads a disabled wait state PSW.
	REQXEQ	Scans the function table and calls the appropriate routine based on that code (either DMTCMX or DMTMGX).
	DEACT	Deactivates the link table entry.
	MSG	Writes messages.
	TERMINAT	Terminates a specified task.
	QUIESCE	Becomes the task code for a task in the process of termination. Looks for any outstanding I/O for the terminating task. If any outstanding I/O is found, issues HIO and waits for completion. When all I/O is completed, it terminates the task.
DMTSIG	DMTSIG	Performs a task alert exit for a requesting task.
DMTSML	DMTSML	Functions as an RJE work station into a remote system using the MULTI-LEAVING transmission protocol. It can also function as a host to a remote programmable work station supporting a System/370, System/3, Model 20, 1130, or a 2922.
	SMLINIT	Initializes various parameters needed by DMTSML. Saves the link table address, initializes output tags, and constructs the sign-on card from information in the operand field of the START command.
	ISIO	Performs the enable sequence on the communications line, analyzes the response received, and, if the response is correct, writes the line connected message.
	ASYNEXIT	This is the alert exit entered by DMTSIG. Two tasks may alert this line driver: • DMTREX--When a command has been entered for pro- Processing by the DMTSML line driver. • DMTAXS--When DMTAXS must asynchronously notify DMSML that a file has arrived for transmission.

Module Name	Entry Points	Function
DMTSML (cont.)	&START	This is the supervisor routine for DMTSML. The commutator cycles while looking for a routine to enter until all commutator entries are closed. It then waits for a synch lock list to be posted.
	&CTRN1	Dequeues tasks from its task queue and performs the action requested by the control record in the dequeued task.
	&PRTN1	Dequeues tasks from its task queue, obtains a new output spool device, if needed, from DMTAXS, and sends the task to a virtual printer.
	&URTN1	Dequeues tasks from its task queue, obtains a new output spool device, if needed, from DMTAXS, and sends the task to a virtual punch.
	&JRTN1	Dequeues tasks from its task queue, obtains a new output spool device, if needed, from DMTAXS, and sends the task to a virtual device.
	&USREXIT	Validates the ID card in the front of decks read in from a remote card reader.
	&PRTN1	Reads in files from the VM/370 spool file system, deblocks the files into 132 byte records, and issues a call to PUT to block the record into a transmission buffer.
	AXSGET	This routine is the interface to DMTAXS. It gets files ready to transmit and purges those files when transmission is complete.
	VMDEBLOK	This is the deblock routine for the VM/370 page spool buffers. It returns the deblocked record in the RDTTDTA1 buffer.
	HEADPREP	Provides, one record after the other, the separator and header for print files and the header card for punch files.
	TODEBCD	Converts System/370 TOD to EBCDIC data and time.
	&WRTN1	Writes received messages to the RSCS operator, if in RJE mode. Passes commands to DMTREX for execution, if in HOST mode. These commands or messages are dequeued from console TCT.
	CMDPROC	Executes commands passed to it in the CMDRESP buffer after an alert from DMTREX indicating a command was entered.
	MSGPROC	Entered when the MSGECB is posted by this task's asynchronous exit indicating messages are in the message queue for this task. These messages are unstacked from the message queue by repeated calls to GMSGREQ and queued for transmission.

Module Name	Entry Points	Function
DMTSML (cont.)	MSG	Prepares and sends requests to the specialized task REX to writes messages on the operator's console.
	PARMGET	Scans lines and tests for delimiter characters.
	&TPPUT	Takes a line and packs it into a teleprocessing buffer. When the buffer is filled, it is queued onto OUTBUF for processing by COMSUP.
	&TPGET	Deblocks received telecommunications buffers into tasks and queues the task onto the appropriate processors TCTTASK queue.
	COMSUP	Processes all I/O on the communications line. It deque-ues TP buffers from OUTBUF for transmission and queues received TP buffers onto the &INBUF queue for deblocking by TPGET.
	CERROR	Analyses all errors on the communications line. The ap-propriate corrective action is taken depending on the on the type of error.
DMTSTO	DMTSTO	Reserves pages of free storage for use by calling task programs. Task programs free storage pages by clearing the associated map byte to zero in the main storage map.
DMTSVC	DMTSVC	This module is the MSUP interrupt handler and receives control directly when an SVC interrupt occurs.
DMTSYS	DMTSYS	The common system control information area that is shared by all task level functions of RSCS. All instal-lation variable information used by an RSCS system is reflected in the assembly of this module. This module is the only module that must be assembled as part of an RSCS system generation.
DMTVEC	DMTVEC	Describes the fixed address storage utilization for MSUP, beginning at main storage address X'200'. System/370 architecture defies the first 512 bytes of main storage and MSUP uses this area as it is defined. This area is not included in the DMTVEC module to facilitate initial system loading. This area is ini-tialized by DMTINI at IPL time.
DMTWAT	DMTWAT	Called directly from task programs by a BAL instruction. It provides event synchronization by means of suspending a task's execution until some specified event is sig-nalled complete by another process in the system.

Module	External References (Labels and Modules)												
DMTAKE	ACTIVE	DISPATCH	GIVEADDR	GIVEE	GIVENAME	GIVENEXT	GIVENID	GIVEQ	GIVERID	POSTREQ	QREQ	R1	R11
	R12	R13	R14	R15	R2	R3	R4	R5	R6	SVECTORS	TAREA	TASKE	TASKID
	TASKNAME	TASKNEXT	TASKQ	TGREG1	TGREG15	TREQLOCK							
DMTASK	ACTIVE	ALERTQ	DISPATCH	EXTQ	FREEE	FREEID	FREENEXT	GIVEADDR	GIVEE	GIVENEXT	GIVENID	GIVEQ	IOE
	IOEXITQ	IOID	IONEXT	IOSUBQ	LIMEC	MAINMAP	MAINSIZE	MPXIOQ	POSTREQ	QREQ	R0	R1	R12
	R13	R14	R15	R2	R3	R4	R5	R6	R7	R8	R9	SELIOQ	SVECTORS
	TAREA	TASKE	TASKID	TASKNAME	TASKNEXT	TASKQ	TASKSAVE	TASKSTAT	TGREG0	TGREG13	TGREG15		
DMTASY	ACTIVE	ALERTQ	ASYNCODE	ASYNE	ASYNEXIT	ASYNID	ASYNNEXT	ASYNTASK	DISPATCH	EXTQ	IOEXITQ	QREQ	R0
	R1	R12	R13	R14	R15	R2	R3	R4	SVECTORS	TAREA	TASKE	TASKID	TGREG0
	TGREG15												
DMTAXS	ALERTREQ	ASYNREQ	COMDSECT	CSW	DE	DEVCODE	DEVCUU	GIVEREQ	GLINKREQ	GPAGEREQ	GTODEBCD	IOTABLE	LACTCLS1
	LACTIVE	LACTTNME	LALERT	LFLAG	LINKID	LINKLEN	LINKTABL	LPENDING	LPOINTER	LRESERVD	LSPARE	LTAKEN	MAINMAP
	POSTREQ	PRCGADDR	ROUTDEST	ROUTE	ROUTNEXT	RCUTSIZE	R0	R1	R10	R11	R12	R13	R14
	R15	R2	R3	R4	R5	R6	R7	R8	R9	SFBCLAS	SFBCOPY	SFBDATE	SFBDIST
	SFBFILID	SFBFLAG	SFBFLAG2	SFBFNAME	SFBFTYPE	SFBINUSE	SFBLOK	SFBORIG	SFBRECNO	SFBRECSZ	SFBREQUE	SFBSHOLD	SFBTYPE
	SFBUHOLD	SVECTORS	TAG	TAGBLOCK	TAGCLASS	TAGCOPY	TAGDEV	TAGDIST	TAGFLAG	TAGFLAG2	TAGID	TAGINDEV	TAGINLOC
	TAGINTOD	TAGINVM	TAGLEN	TAGLINK	TAGNAME	TAGNEXT	TAGPRIOR	TAGRECLN	TAGRECNM	TAGTOLOC	TAGTOVM	TAGTYPE	TAKEREQ
	TASKE	TASKSAVE	TCOM	TLINKS	TROUTE	TTAGQ	TYPPRT	TYPPUN	TYP1403	TYP2540P	TYP3211	WAITREQ	
DMTCMX	ALERTREQ	COMDSECT	DEVCODE	DEVCUU	DMTCRE	DMTCREDA	DMTMGX	DMTREXCN	DMTREXHC	DMTREXID	GLINKREQ	GTODEBCD	IOTABLE
	LACTCLS1	LACTDRVR	LACTIVE	LACTLINE	LACTTNME	LDEFCLS1	LDEFDRVR	LDEFLINE	LDEFTNME	LDRAIN	LFLAG	LHOLD	LINKID
	LINKLEN	LINKTABL	LPENDING	LPOINTER	LRESERVD	LTAKEN	LTRALL	LTRERR	MAINMAP	MAINSIZE	R0	R1	R10
	R11	R12	R13	R14	R15	R2	R3	R4	R5	R6	R7	R8	R9
	SFBSHOLD	SFBUHOLD	SVECTORS	TAG	TAGBLOCK	TAGCLASS	TAGCOPY	TAGDIST	TAGFLAG	TAGID	TAGINLOC	TAGINTOD	TAGINVM
	TAGLINK	TAGNAME	TAGNEXT	TAGPRIOR	TAGRECNM	TAGTOLOC	TAGTOVM	TCOM	TLINKS	TPORTS	TTAGQ		
DMTCOM	ACTIVE	DISPATCH	LACTTNME	LINKID	LINKLEN	LINKTABL	LMSGQ	MAINMAP	MAINREQ	MAINSIZE	R0	R1	R10
	R11	R12	R13	R14	R15	R2	R3	R4	R5	R6	R7	R8	R9
	SVECTORS	TAREA	TASKE	TASKID	TASKNAME	TASKNEXT	TASKQ	TGREG0	TGREG1	TGREG15	TGREG2	TLINKS	TPSW

Module	External References (Labels and Modules)												
DMTCRE	CC	CE	CUE	DE	DEVCODE	ENDCSW	IOREQ	IOTABLE	LACTDRVR	LACTTNME	LINKTABL	MAINMAP	MAINREQ
	MAINSIZE	R0	R1	R12	R14	R15	R2	R3	R4	R5	R6	R7	R9
	SILI	SIOCOND	SVECTORS	TAREA	TASKREQ	TGREG0	TGREG1	TGREG2	TYP2314	WAITREQ			
DMTDSP	ACTIVE	LIMBC	LOCKLIST	NEWPSW	R0	R1	R15	R2	R3	R4	SVECTORS	TAREA	TASKE
	TASKID	TASKNEXT	TASKQ	TASKSAVE	TASKSTAT	TGREG0	TGREG1	TPSW	WAITING				
DMTEXT	ACTIVE	ASYNCODE	ASYNE	ASYNEXIT	ASYNNEXT	ASYNTASK	DISPATCH	EXTQ	NEWEXT	OLDEXT	R0	R1	R13
	R14	R15	R2	R3	R4	SSAVE	SVECTORS	TAREA	TASKE	TASKSAVE	TGREG0	TGREG14	TPSW
DMTGIV	ACTIVE	DISPATCH	GIVEADDR	GIVEE	GIVENAME	GIVENEXT	GIVENID	GIVEQ	GIVERID	POSTREQ	QREQ	R0	R1
	R12	R13	R14	R15	R2	R3	R4	SVECTORS	TAREA	TASKE	TASKID	TASKNAME	TASKNEXT
	TASKQ	TASKSAVE	TGREG15	TREQLOCK									
DMTINI	CAW	CC	CE	CLASDASD	CLASTERM	CSW	DE	DEVCODE	DEVCUU	DISPATCH	DMTCREDA	DMTIOMIN	DMTMAPME
	DMTMAPCE	DMTREXVL	FREEE	FREENEXT	FREEQ	IOTABLE	IPLCCW1	IPLPSW	MAINMAP	MAINSIZE	MCHEK	NEWEXT	NEWIO
	OLDIO	QREQ	QUEUE	R0	R1	R10	R11	R12	R13	R14	R15	R2	R3
	R4	R5	R6	R7	R8	R9	SILI	SVECTORS	TASKE	TASKID	TASKNAME	TASKNEXT	TASKQ
	TASKSAVE	TASKSTAT	TIMER	TYP2314	TYP3210	TYP3330	TYP3340	WAIT					
DMTIOM	ACTIVE	ASYNCODE	ASYNE	ASYNEXIT	ASYNNEXT	ASYNTASK	BUSY	CAW	CE	CHANDONE	CSW	DE	DEVCUU
	DISPATCH	ENDCSW	ENDSENSE	IOADDR	IOE	IOEXITQ	IOID	IONEXT	IOSBCHAN	IOSTAT	IOSUBQ	IOSYNCH	IOTABLE
	IOTABLEA	MPXIOQ	NEWIO	OLDIO	PCI	POSTREQ	PROGADDR	QREQ	R0	R1	R12	R13	R14
	R15	R2	R3	R4	R5	R6	SELIOQ	SENSING	SENSREQ	SIOCOND	SM	SSAVE	SVECTORS
	TAREA	TASKE	TASKID	TASKSAVE	TGREG0	TGREG14	TPSW	UC					
DMTLAX	ASYNREQ	CLASTERM	LACTIVE	LACTLINE	LFLAG	LINKID	LINKLEN	LINKTABL	R0	R1	R12	R14	R15
	R2	R3	R4	R5	R6	R7	R8	R9	SVECTORS	TLINKS	TPORTS	TYPBSC	TYP2700
	WAITREQ												
DMTMGX	ALERTREQ	COMDSECT	DMTMSG	DMTREXHC	GLINKREQ	LACTIVE	LACTTNME	LFLAG	LINKID	LINKTABL	PMSGREQ	R0	R1
	R10	R12	R13	R14	R15	R2	R3	R4	R5	R6	R7	R8	R9
	SVECTORS	TCOM	TLINKS										
DMTNPT	ASYNREQ	BUSOUT	CC	CMDREJ	COMDSECT	EQCHK	GIVEREQ	GMSGREQ	GPAGEREQ	GTODEBCD	INTREQ	IOREQ	LACTLINE
	LDRAIN	LERRCNT	LFLAG	LHOLD	LINKID	LINKTABL	LTOCNT	LTRALL	LTRERR	LTRNSCNT	PMSGREQ	POSTREQ	R0
	R1	R10	R11	R12	R13	R14	R15	R2	R3	R4	R5	R6	R7
	R8	R9	SILI	SKIP	SPLINK	SPRECNUM	SVECTORS	TAG	TAGDEV	TAGDIST	TAGID	TAGINDEV	TAGINLOC
	TAGINTOD	TAGINVM	TAGLINK	TAGNAME	TAGNEXT	TAGRECNM	TAGTOLOC	TAGTOVM	TASKE	TASKSAVE	TCOM	TLINKS	TYPPRT
	TYPPUN	TYP2700	TYP3210	UC	UE	WAITREQ							
DMTPST	R0	R1	R14	TASKE	TASKSTAT	WAITING							
DMTQRQ	FREEE	FREEID	FREENEXT	FREEQ	R1	R14	R15	SVECTORS					

Module	External References (Labels and Modules)												
DMTREX	ACTIVE	ASYNREQ	ATTN	COMDSECT	CSW	DEVCODE	DEVCUU	DISPATCH	DMTCMX	DMTCCMVC	DMTCRE	DMTMGX	DMTSYSLK
	DMTSYSND	DMTSYSPT	DMTSYSRT	DMTSYSTQ	ENDCSW	GMSGREQ	IOADDR	IOE	IOID	IONEXT	IOREQ	IOSYNCH	IOTABLE
	IOTABLEA	LACTDRVR	LACTIVE	LACTLINE	LACTTNME	LDEFDRVR	LFLAG	LHALT	LIMBC	LINKID	LINKLEN	LINKTABL	LMSGQ
	LOCKLIST	MAINMAP	MAINSIZE	MPXIOQ	NEWPROG	OLDPROG	POSTREQ	PROGADDR	R0	R1	R12	R13	R14
	R15	R2	R3	R4	R5	SELIOQ	SILI	SSAVE	SVECTORS	TAKEREQ	TAREA	TASKE	TASKID
	TASKNAME	TASKNEXT	TASKQ	TASKREQ	TASKSAVE	TASKSTAT	TCOM	TGREG0	TGREG12	TGREG13	TGREG2	TGREG4	TLINKS
	TPORTS	TPSW	TVECTOR0	TYP3210	UE	WAITING	WAITREQ						
DMTSIG	ACTIVE	ALERTQ	ASYNE	ASYNEXIT	ASYNNEXT	ASYNTASK	DISPATCH	R0	R13	R14	R15	R2	R3
	SVECTORS	TAREA	TASKE	TASKNAME	TGREG15	TGREG2							
DMTSML	ASYNREQ	CC	CCC	CD	COMDSECT	DEVCUU	ENDCSW	GIVEREQ	GMSGREQ	GPAGEREQ	GTODEBCD	IOREQ	IOSYNCH
	IOTABLE	LACTLINE	LDRAIN	LERRCNT	LFLAG	LHOLD	LINKID	LINKTABL	LTOCNT	LTRALL	LTRERR	LTRNSCNT	PMSGREQ
	POSTREQ	PROGADDR	R0	R1	R10	R11	R12	R13	R14	R15	R2	R3	R4
	R5	R6	R7	R8	R9	SILI	SKIP	SPLINK	SPRECNUM	SVECTORS	TAG	TAGDEV	TAGDIST
	TAGID	TAGINDEV	TAGINLOC	TAGINTOD	TAGINVM	TAGLINK	TAGNAME	TAGRECNM	TAGTCLOC	TAGTOVM	TASKE	TASKSAVE	TCOM
	TLINKS	TYPPRT	TYPPUN	TYP2700	TYP3210	UC	UE	WAITREQ					
DMTSTO	ACTIVE	DISPATCH	MAINMAP	R0	R1	R14	R15	R2	R3	R4	SVECTORS	TAREA	TASKE
	TASKID	TGREG1	TGREG15										
DMTSVC	ACTIVE	NEWPSW	NEWSVC	OLDSVC	R0	R13	R14	R15	SSAVE	SVECTORS	TAREA	TASKE	TASKSAVE
	TGREG0	TGREG13	TGREG14	TPSW									
DMTSYS	LINKLEN	ROUTSIZE	TAGLEN										
DMTVEC	DMTAKE	DMTASK	DMTASY	DMTDSP	DMTGIV	DMTIOMRQ	DMTMAPMS	DMTMAPQE	DMTMAPQU	DMTPST	DMTQRQ	DMTSIG	DMTSTO
	DMTWAT												
DMTWAT	ACTIVE	DISPATCH	LOCKLIST	R1	R14	R15	R2	R3	R4	R5	R6	SVECTORS	TASKE
	TASKSTAT	WAITING											

RSCS LABEL-TO-MODULE CROSS REFERENCE

Label	Count	References
ACTIVE	000030	DMTAKE DMTASK DMTASY DMTCOM DMTDSP DMTEXT DMTGIV DMTIOM DMTREX DMTSIG DMTSTO DMTSVC DMTWAT
ALERTQ	000003	DMTASK DMTASY DMTSIG
ALERTREQ	000005	DMTAXS DMTCMX DMTMGX
ASYNCODE	000004	DMTASY DMTEXT DMTIOM
ASYNE	000011	DMTASY DMTEXT DMTIOM DMTSIG
ASYNEXIT	000005	DMTASY DMTEXT DMTIOM DMTSIG
ASYNID	000002	DMTASY
ASYNNEXT	000011	DMTASY DMTEXT DMTIOM DMTSIG
ASYNREQ	000006	DMTAXS DMTLAX DMTNPT DMTREX DMTSML
ASYNTASK	000006	DMTASY DMTEXT DMTIOM DMTSIG
ATTN	000001	DMTREX
BUSOUT	000001	DMTNPT
BUSY	000001	DMTIOM
CAW	000006	DMTINI DMTIOM
CC	000087	DMTCRE DMTINI DMTNPT DMTSML
CCC	000001	DMTSML
CD	000001	DMTSML
CE	000004	DMTCRE DMTINI DMTIOM
CHANDONE	000004	DMTIOM
CLASDASD	000001	DMTINI
CLASTERM	000005	DMTINI DMTLAX
CMDREJ	000001	DMTNPT
COMDSECT	000006	DMTAXS DMTCMX DMTMGX DMTNPT DMTREX DMTSML
CSW	000026	DMTAXS DMTINI DMTIOM DMTREX
CUE	000001	DMTCRE
DE	000006	DMTAXS DMTCRE DMTINI DMTIOM
DEVCODE	000013	DMTAXS DMTCMX DMTCRE DMTINI DMTREX
DEVCUU	000008	DMTAXS DMTCMX DMTINI DMTIOM DMTREX DMTSML
DISPATCH	000015	DMTAKE DMTASK DMTASY DMTCOM DMTEXT DMTGIV DMTINI DMTIOM DMTREX DMTSIG DMTSTO DMTWAT
DMTAKE	000001	DMTVEC
DMTASK	000001	DMTVEC
DMTASY	000001	DMTVEC

Label	Count	References			
DMTCMX	000001	DMTREX			
DMTCOMVC	000001	DMTREX			
DMTCRE	000003	DMTCMX	DMTREX		
DMTCRELA	000003	DMTCMX	DMTINI		
DMTDSP	000001	DMTVEC			
DMTGIV	000001	DMTVEC			
DMTIOMIN	000001	DMTINI			
DMTIOMRQ	000001	DMTVEC			
DMTMAPME	000001	DMTINI			
DMTMAPMS	000001	DMTVEC			
DMTMAPQE	000002	DMTINI	DMTVEC		
DMTMAPQU	000001	DMTVEC			
DMTMGX	000010	DMTCMX	DMTREX		
DMTMSG	000001	DMTMGX			
DMTPST	000001	DMTVEC			
DMTQRQ	000001	DMTVEC			
DMTREXCN	000001	DMTCMX			
DMTREXHC	000004	DMTCMX	DMTMGX		
DMTREXID	000001	DMTCMX			
DMTREXVL	000001	DMTINI			
DMTSIG	000001	DMTVEC			
DMTSTO	000001	DMTVEC			
DMTSYSLK	000001	DMTREX			
DMTSYSND	000001	DMTREX			
DMTSYSPT	000001	DMTREX			
DMTSYSRT	000001	DMTREX			
DMTSYSTQ	000001	DMTREX			
DMTWAT	000001	DMTVEC			
ENDCSW	000014	DMTCRE	DMTIOM	DMTREX	DMTSML
ENDSENSE	000001	DMTIOM			
EQCHK	000001	DMTNPT			
EXTQ	000004	DMTASK	DMTASY	DMTEXT	
FREEE	000008	DMTASK	DMTINI	DMTQRQ	
FREEID	000002	DMTASK	DMTQRQ		
FREENEXT	000009	DMTASK	DMTINI	DMTQRQ	
FREEQ	000005	DMTINI	DMTQRQ		
GIVEADDR	000004	DMTAKE	DMTASK	DMTGIV	
GIVEE	000013	DMTAKE	DMTASK	DMTGIV	
GIVENAME	000003	DMTAKE	DMTGIV		
GIVENEXT	000014	DMTAKE	DMTASK	DMTGIV	
GIVENID	000005	DMTAKE	DMTASK	DMTGIV	
GIVEQ	000005	DMTAKE	DMTASK	DMTGIV	
GIVEREQ	000018	DMTAXS	DMTNPT	DMTSML	
GIVERID	000002	DMTAKE	DMTGIV		

Label	Count	References								
GLINKREQ	000003	DMTAXS	DMTCMX	DMTMGX						
GMSGREQ	000004	DMTNPT	DMTREX	DMTSML						
GPAGEREQ	000005	DMTAXS	DMTNPT	DMTSML						
GTODEBCD	000004	DMTAXS	DMTCMX	DMTNPT	DMTSML					
INTREQ	000001	DMTNPT								
IOADDR	000008	DMTIOM	DMTREX							
IOE	000024	DMTASK	DMTIOM	DMTREX						
IOEXITQ	000003	DMTASK	DMTASY	DMTIOM						
IOID	000004	DMTASK	DMTIOM	DMTREX						
IONEXT	000015	DMTASK	DMTIOM	DMTREX						
IOREQ	000013	DMTCRE	DMTNPT	DMTREX	DMTSML					
IOSBCHAN	000006	DMTIOM								
IOSTAT	000009	DMTIOM								
IOSUBQ	000007	DMTASK	DMTIOM							
IOSYNCH	000021	DMTIOM	DMTREX	DMTSML						
IOTABLE	000034	DMTAXS	DMTCMX	DMTCRE	DMTINI	DMTIOM	DMTREX	DMTSML		
IOTABLEA	000009	DMTIOM	DMTREX							
IPLCCW1	000001	DMTINI								
IPLPSW	000005	DMTINI								
LACTCLS1	000005	DMTAXS	DMTCMX							
LACTDRVR	000008	DMTCMX	DMTCRE	DMTREX						
LACTIVE	000019	DMTAXS	DMTCMX	DMTLAX	DMTMGX	DMTREX				
LACTLINE	000013	DMTCMX	DMTLAX	DMTNPT	DMTREX	DMTSML				
LACTTNME	000021	DMTAXS	DMTCMX	DMTCOM	DMTCRE	DMTMGX	DMTREX			
LALERT	000005	DMTAXS								
LDEFCLS1	000004	DMTCMX								
LDEFDRVR	000005	DMTCMX	DMTREX							
LDEFLINE	000004	DMTCMX								
LDEFTNME	000004	DMTCMX								
LDRAIN	000013	DMTCMX	DMTNPT	DMTSML						
LERRCNT	000008	DMTNPT	DMTSML							
LFLAG	000073	DMTAXS	DMTCMX	DMTLAX	DMTMGX	DMTNPT	DMTREX	DMTSML		
LHALT	000003	DMTREX								
LHOLD	000018	DMTCMX	DMTNPT	DMTSML						
LIMBO	000005	DMTASK	DMTDSP	DMTREX						
LINKID	000045	DMTAXS	DMTCMX	DMTCOM	DMTLAX	DMTMGX	DMTNPT	DMTREX	DMTSML	
LINKLEN	000017	DMTAXS	DMTCMX	DMTCOM	DMTLAX	DMTREX	DMTSYS			
LINKTABL	000015	DMTAXS	DMTCMX	DMTCOM	DMTCRE	DMTLAX	DMTMGX	DMTNPT	DMTREX	DMTSML
LMSGQ	000005	DMTCOM	DMTREX							
LOCKLIST	000004	DMTDSP	DMTREX	DMTWAT						
LPENDING	000018	DMTAXS	DMTCMX							
LPOINTER	000015	DMTAXS	DMTCMX							
LRESERVD	000006	DMTAXS	DMTCMX							
LSPARE	000002	DMTAXS								

Label	Count	References											
LTAKEN	000006	DMTAXS	DMTCMX										
LTOCNT	000008	DMTNPT	DMTSML										
LTRALL	000016	DMTCMX	DMTNPT	DMTSML									
LTRERR	000013	DMTCMX	DMTNPT	DMTSML									
LTRNSCNT	000008	DMTNPT	DMTSML										
MAINMAP	000016	DMTASK	DMTAXS	DMTCMX	DMTCOM	DMTCRE	DMTINI	DMTREX	DMTSTO				
MAINREQ	000002	DMTCOM	DMTCRE										
MAINSIZE	000007	DMTASK	DMTCMX	DMTCOM	DMTCRE	DMTINI	DMTREX						
MCHEK	000004	DMTINI											
MPXIOQ	000007	DMTASK	DMTIOM	DMTREX									
NEWEXT	000003	DMTEXT	DMTINI										
NEWIO	000004	DMTINI	DMTIOM										
NEWPROG	000004	DMTREX											
NEWPSW	000006	DMTDSP	DMTSVC										
NEWSVC	000001	DMTSVC											
OLDEXT	000002	DMTEXT											
OLDIO	000006	DMTINI	DMTIOM										
CLDPROG	000001	DMTREX											
OLDSVC	000004	DMTSVC											
PCI	000001	DMTIOM											
PMSGREQ	000003	DMTMGX	DMTNPT	DMTSML									
POSTREQ	000011	DMTAKE	DMTASK	DMTAXS	DMTGIV	DMTIOM	DMTNPT	DMTREX	DMTSML				
PROGADDR	000012	DMTAXS	DMTIOM	DMTREX	DMTSML								
QREQ	000014	DMTAKE	DMTASK	DMTASY	DMTGIV	DMTINI	DMTIOM						
QUEUE	000001	DMTINI											
ROUTDEST	000001	DMTAXS											
ROUTE	000001	DMTAXS											
ROUTNEXT	000002	DMTAXS											
ROUTSIZE	000003	DMTAXS	DMTSYS										
R0	000513	DMTASK	DMTASY	DMTAXS	DMTCMX	DMTCOM	DMTCRE	DMTDSP	DMTEXT	DMTGIV	DMTINI	DMTIOM	DMTLAX
		DMTMGX	DMTNPT	DMTPST	DMTREX	DMTSIG	DMTSML	DMTSTO	DMTSVC				
R1	001048	DMTAKE	DMTASK	DMTASY	DMTAXS	DMTCMX	DMTCOM	DMTCRE	DMTDSP	DMTEXT	DMTGIV	DMTINI	DMTIOM
		DMTLAX	DMTMGX	DMTNPT	DMTPST	DMTQRQ	DMTREX	DMTSML	DMTSTO	DMTWAT			
R10	000058	DMTAXS	DMTCMX	DMTCOM	DMTINI	DMTMGX	DMTNPT	DMTSML					
R11	000033	DMTAKE	DMTAXS	DMTCMX	DMTCOM	DMTINI	DMTNPT	DMTSML					
R12	000050	DMTAKE	DMTASK	DMTASY	DMTAXS	DMTCMX	DMTCOM	DMTCRE	DMTGIV	DMTINI	DMTIOM	DMTLAX	DMTMGX
		DMTNPT	DMTREX	DMTSML									
R13	000190	DMTAKE	DMTASK	DMTASY	DMTAXS	DMTCMX	DMTCOM	DMTEXT	DMTGIV	DMTINI	DMTIOM	DMTMGX	DMTNPT
		DMTREX	DMTSIG	DMTSML	DMTSVC								
R14	001066	DMTAKE	DMTASK	DMTASY	DMTAXS	DMTCMX	DMTCOM	DMTCRE	DMTEXT	DMTGIV	DMTINI	DMTIOM	DMTLAX
		DMTMGX	DMTNPT	DMTPST	DMTQRQ	DMTREX	DMTSIG	DMTSML	DMTSTO	DMTSVC	DMTWAT		
R15	000938	DMTAKE	DMTASK	DMTASY	DMTAXS	DMTCMX	DMTCOM	DMTCRE	DMTDSP	DMTEXT	DMTGIV	DMTINI	DMTIOM
		DMTLAX	DMTMGX	DMTNPT	DMTQRQ	DMTREX	DMTSIG	DMTSML	DMTSTO	DMTSVC	DMTWAT		

Label	Count	References
R2	000739	DMTAKE DMTASK DMTASY DMTAXS DMTCMX DMTCOM DMTCRE DMTDSP DMTEXT DMTGIV DMTINI DMTIOM DMTLAX DMTMGX DMTNPT DMTREX DMTSIG DMTSML DMTSTO DMTWAT
R3	000723	DMTAKE DMTASK DMTASY DMTAXS DMTCMX DMTCOM DMTCRE DMTDSP DMTEXT DMTGIV DMTINI DMTIOM DMTLAX DMTMGX DMTNPT DMTREX DMTSIG DMTSML DMTSTO DMTWAT
R4	000620	DMTAKE DMTASK DMTASY DMTAXS DMTCMX DMTCOM DMTCRE DMTDSP DMTEXT DMTGIV DMTINI DMTIOM DMTLAX DMTMGX DMTNPT DMTREX DMTSML DMTSTO DMTWAT
R5	000418	DMTAKE DMTASK DMTAXS DMTCMX DMTCOM DMTCRE DMTINI DMTIOM DMTLAX DMTMGX DMTNPT DMTREX DMTSML DMTWAT
R6	000460	DMTAKE DMTASK DMTAXS DMTCMX DMTCOM DMTCRE DMTINI DMTIOM DMTLAX DMTMGX DMTNPT DMTSML DMTWAT
R7	000309	DMTASK DMTAXS DMTCMX DMTCOM DMTCRE DMTINI DMTLAX DMTMGX DMTNPT DMTSML
R8	000368	DMTASK DMTAXS DMTCMX DMTCOM DMTINI DMTLAX DMTMGX DMTNPT DMTSML
R9	000122	DMTASK DMTAXS DMTCMX DMTCOM DMTCRE DMTINI DMTLAX DMTMGX DMTNPT DMTSML
SELIOQ	000007	DMTASK DMTIOM DMTREX
SENSING	000003	DMTIOM
SENSREQ	000002	DMTIOM
SFBCLAS	000001	DMTAXS
SFBCOPY	000001	DMTAXS
SFBDATE	000001	DMTAXS
SFBDIST	000001	DMTAXS
SFBFILID	000010	DMTAXS
SFBFLAG	000002	DMTAXS
SFBFLAG2	000001	DMTAXS
SFBFNAME	000001	DMTAXS
SFBFTYPE	000001	DMTAXS
SFBINUSE	000001	DMTAXS
SFBLOK	000002	DMTAXS
SFBORIG	000002	DMTAXS
SFBRECNO	000001	DMTAXS
SFBRECSZ	000001	DMTAXS
SFBREQUE	000004	DMTAXS
SFBSHOLD	000004	DMTAXS DMTCMX
SFBTYPE	000001	DMTAXS
SFBUHOLD	000005	DMTAXS DMTCMX
SILI	000130	DMTCRE DMTINI DMTNPT DMTREX DMTSML
SIOCOND	000005	DMTCRE DMTIOM
SKIP	000002	DMTNPT DMTSML
SM	000001	DMTIOM
SPLINK	000006	DMTNPT DMTSML
SPRECNUM	000018	DMTNPT DMTSML
SSAVE	000011	DMTEXT DMTIOM DMTREX DMTSVC
SVECTORS	000022	DMTAKE DMTASK DMTASY DMTAXS DMTCMX DMTCOM DMTCRE DMTDSP DMTEXT DMTGIV DMTINI DMTIOM DMTLAX DMTMGX DMTNPT DMTQRQ DMTREX DMTSIG DMTSML DMTSTO DMTSVC DMTWAT
TAG	000038	DMTAXS DMTCMX DMTNPT DMTSML

Label	Count	References											
TAGBLOCK	000019	DMTAXS	DMTCMX										
TAGCLASS	000009	DMTAXS	DMTCMX										
TAGCOPY	000010	DMTAXS	DMTCMX										
TAGDEV	000016	DMTAXS	DMTNPT	DMTSML									
TAGDIST	000015	DMTAXS	DMTCMX	DMTNPT	DMTSML								
TAGFLAG	000008	DMTAXS	DMTCMX										
TAGFLAG2	000004	DMTAXS											
TAGID	000022	DMTAXS	DMTCMX	DMTNPT	DMTSML								
TAGINDEV	000022	DMTAXS	DMTNPT	DMTSML									
TAGINLCC	000012	DMTAXS	DMTCMX	DMTNPT	DMTSML								
TAGINTOD	000007	DMTAXS	DMTCMX	DMTNPT	DMTSML								
TAGINVM	000008	DMTAXS	DMTCMX	DMTNPT	DMTSML								
TAGLEN	000002	DMTAXS	DMTSYS										
TAGLINK	000020	DMTAXS	DMTCMX	DMTNPT	DMTSML								
TAGNAME	000009	DMTAXS	DMTCMX	DMTNPT	DMTSML								
TAGNEXT	000051	DMTAXS	DMTCMX	DMTNPT									
TAGPRIOR	000012	DMTAXS	DMTCMX										
TAGRECLN	000001	DMTAXS											
TAGRECNM	000004	DMTAXS	DMTCMX	DMTNPT	DMTSML								
TAGTOLOC	000013	DMTAXS	DMTCMX	DMTNPT	DMTSML								
TAGTOVM	000017	DMTAXS	DMTCMX	DMTNPT	DMTSML								
TAGTYPE	000001	DMTAXS											
TAKEREQ	000002	DMTAXS	DMTREX										
TAREA	000021	DMTAKE	DMTASK	DMTASY	DMTCOM	DMTCRE	DMTDSP	DMTEXT	DMTGIV	DMTIOM	DMTREX	DMTSIG	DMTSTO
		DMTSVC											
TASKE	000044	DMTAKE	DMTASK	DMTASY	DMTAXS	DMTCOM	DMTDSP	DMTEXT	DMTGIV	DMTINI	DMTIOM	DMTNPT	DMTPST
		DMTREX	DMTSIG	DMTSML	DMTSTO	DMTSVC	DMTWAT						
TASKID	000021	DMTAKE	DMTASK	DMTASY	DMTCOM	DMTDSP	DMTGIV	DMTINI	DMTIOM	DMTREX	DMTSTO		
TASKNAME	000016	DMTAKE	DMTASK	DMTCOM	DMTGIV	DMTINI	DMTREX	DMTSIG					
TASKNEXT	000023	DMTAKE	DMTASK	DMTCOM	DMTDSP	DMTGIV	DMTINI	DMTREX					
TASKQ	000010	DMTAKE	DMTASK	DMTCOM	DMTDSP	DMTGIV	DMTINI	DMTREX					
TASKREQ	000003	DMTCRE	DMTREX										
TASKSAVE	000015	DMTASK	DMTAXS	DMTDSP	DMTEXT	DMTGIV	DMTINI	DMTIOM	DMTNPT	DMTREX	DMTSML	DMTSVC	
TASKSTAT	000014	DMTASK	DMTDSP	DMTINI	DMTPST	DMTREX	DMTWAT						
TCOM	000019	DMTAXS	DMTCMX	DMTMGX	DMTNPT	DMTREX	DMTSML						
TGREG0	000014	DMTASK	DMTASY	DMTCOM	DMTCRE	DMTDSP	DMTEXT	DMTIOM	DMTREX	DMTSVC			
TGREG1	000005	DMTAKE	DMTCOM	DMTCRE	DMTDSP	DMTSTO							
TGREG12	000001	DMTREX											
TGREG13	000005	DMTASK	DMTREX	DMTSVC									
TGREG14	000003	DMTEXT	DMTIOM	DMTSVC									
TGREG15	000020	DMTAKE	DMTASK	DMTASY	DMTCOM	DMTGIV	DMTSIG	DMTSTO					
TGREG2	000005	DMTCOM	DMTCRE	DMTREX	DMTSIG								
TGREG4	000001	DMTREX											
TIMER	000001	DMTINI											

Label	Count	References							
TLINKS	000030	DMTAXS	DMTCMX	DMTCOM	DMTLAX	DMTMGX	DMTNPT	DMTREX	DMTSML
TPORTS	000003	DMTCMX	DMTLAX	DMTREX					
TPSW	000014	DMTCOM	DMTDSP	DMTEXT	DMTIOM	DMTREX	DMTSVC		
TREQLOCK	000004	DMTAKE	DMTGIV						
TROUTE	000001	DMTAXS							
TTAGQ	000005	DMTAXS	DMTCMX						
TVECTOR0	000001	DMTREX							
TYPBSC	000002	DMTLAX							
TYPPRT	000009	DMTAXS	DMTNPT	DMTSML					
TYPPUN	000010	DMTAXS	DMTNPT	DMTSML					
TYP1403	000001	DMTAXS							
TYP2314	000005	DMTCRE	DMTINI						
TYP2540P	000001	DMTAXS							
TYP2700	000004	DMTLAX	DMTNPT	DMTSML					
TYP3210	000009	DMTINI	DMTNPT	DMTREX	DMTSML				
TYP3211	000001	DMTAXS							
TYP3330	000002	DMTINI							
TYP3340	000002	DMTINI							
UC	000010	DMTIOM	DMTNPT	DMTSML					
UE	000003	DMTNPT	DMTREX	DMTSML					
WAIT	000002	DMTINI							
WAITING	000005	DMTDSP	DMTPST	DMTREX	DMTWAT				
WAITREQ	000030	DMTAXS	DMTCRE	DMTLAX	DMTNPT	DMTREX	DMTSML		

CP INTERNAL TRACE TABLE

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

- External interruptions
- SVC interruptions
- Program interruptions
- Machine check interruptions
- I/O interruptions
- Free storage requests
- Release of free storage
- Entry into scheduler
- Queue drop
- Run user requests
- Start I/O
- Unstack I/O interruptions
- Storing a virtual CSW
- Test I/O
- Halt device
- Unstack IOBLOK or TRQBLOK
- NCP BTU (Network Control Program Basic Transmission Unit)

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

The size of the trace table depends on the amount of real storage available at IPL time. For each 256K bytes (or part thereof) of real storage available at IPL time, one page (4096 bytes) is allocated to the CP trace table. Each entry in the CP trace table is 16 bytes long. There are 17 possible types of trace table entries; one for each type of event recorded. The first byte of each trace table entry, the identification code, identifies the type of event being recorded.

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

The CP internal trace table is initialized during IPL. If you do not wish to record events in the trace table, issue the MONITOR STOP CPTRACE command to suppress recording. The pages allocated to the trace table are not released and recording can be restarted at any time by issuing the MONITOR START CPTRACE command. If the VM/370 system should abnormally terminate and automatically restart, the tracing of events on the real machine will be active. After a VM/370 IPL (manual or automatic), CP internal tracing is always active.

There are 17 possible types of trace table entries, each uniquely identified by the value of the first byte. Figure 62 describes the format of each type of trace table entry.

Type of Event	Module	Identification Code (hexadecimal)	Format of Trace Table Entry
External interrupt	DMKPSA	01	0: X'01' / 1: X'0000000000' / 6: Interrupt Code / 8: External Old PSW / 15
SVC interrupt	DMKPSA	02	0: X'02 / 1: GR14 or GR15 (See Note 1) / 4: Instruction Length Code / 6: Interrupt Code / 8: SVC Old PSW / 15
Program interrupt	DMKPRG	03	0: X'03' / 1: First 3 bytes of VMPSW / 4: Instruction Length Code / 6: Interrupt Code / 8: Program Old PSW / 15
Machine Check Interrupt	DMKMCH	04	0: X'04' / 1: Address of VMBLOK / 4: First 4 bytes of 8-byte Interrupt Code / 8: Machine Check Old PSW / 15
I/O interrupt	DMKIOS	05	0: X'05' / 1: X'00' / 2: Device Address / 4: I/O Old PSW + 4 / 8: CSW / 15
Free Storage (FREE)	DMKFRE	06	0: X'06 / 1: Address of VMBLOK / 4: GR 0 at entry / 8: GR 1 at exit / 12: GR 14 / 15
Return storage (FRET)	DMKFRE	07	0: X'07' / 1: Address of VMBLOK / 4: GR 0 at entry / 8: GR 1 at entry / 12: GR 14 / 15
Enter Scheduler	DMKSCH	08	0: X'08' / 1: Address of VMBLOK / 4: Value of VMRSTAT, VMDSTAT, VMOSTAT, and VMQSTAT / 8: VMQLEVEL VMTLEVEL / 10: VMIOINT / 12: VMPEND / 13 / 15
Queue drop	DMKSCH	09	0: X'09' / 1: Address of VMBLOK / 4: X'0000' / 6: New Priority / 8: Number of Resident Pages / 10: Projected Working Set / 12: Number of Referenced Pages / 14: Current Page load (PSA) / 15
Run user	DMKDSP	0A	0: X'0A' / 1: X'000000' / 4: RUNUSER value from PSA / 8: RUNPSW value from PSA / 15
Start I/O	DMKCNS DMKIOS DMKVIO	0B	0: X'0B' / 1: Condition Code / 2: Device Address / 4: Address of IOBLOK / 8: CAW / 12: For CC = 1, CSW + 4 otherwise this field is not used / 15
Unstack I/O interrupt	DMKDSP	0C	0: X'0C' / 1: X'00' / 2: Virtual Device Address / 4: Address of VMBLOK / 8: Virtual CSW / 15
Virtual CSW store	DMKVIO	0D	0: X'0D' / 1: Instruction Operation Code / 2: Virtual Device Address / 4: Address of VMBLOK / 8: Virtual CSW / 15
Test I/O	DMKCNS DMKIOS DMKVIO	0E	0: X'0E' / 1: Condition Code / 2: Device Address / 4: Address of IOBLOK / 8: CAW / 12: For CC = 1, CSW + 4 otherwise this field is not used / 15
Halt Device	DMKCNS DMKIOS DMKVIO	0F	0: X'0F' / 1: Condition Code / 2: Device Address / 4: Address of IOBLOK / 8: CAW / 12: For CC = 1, CSW + 4 otherwise this field is not used / 15
Unstack IOBLOK or TRQBLOK	DMKDSP	10	0: X'10' / 1: Address of VMBLOK / 4: Value of VMRSTAT, VMDSTAT, VMOSTAT, and VMQSTAT / 8: Address of IOBLOK or TRQBLOK / 12: Interrupt Return Address / 15
NCP BTU (See Note 2)	DMKRNH	11	0: X'11' / 1: X'00' / 2: CONSRID / 4: CONDEST / 6: CONRTAG / 8: CONSYSR CONEXTR / 10: CONTCMD / 12: CONFUNC CONDFLG / 14: CONDCNT / 15

Notes: 1. If the interrupt code (bytes 6 and 7) is 0C, the contents of GR 14 are displayed. For all other interrupt codes, the contents of GR 15 are displayed.
2. Bytes 2 through 15 of a code 11 trace record represent a Basic Transmission Unit, sent or received by a 3704/3705. If CONSYSR/CONEXTR are zero, the BTU was transmitted to the 3704/3705. If they are non-zero, the BTU was received. If CONTCMD equals X'7700', this is an unsolicited BTU response.

Figure 62. CP Trace Table Entries

CP COMMANDS USED TO DEBUG THE VIRTUAL MACHINE

The VM/370 Control Program has a set of interactive commands that control the VM/370 system and enable the user to control his virtual machines and associated control program facilities. The virtual machine operator using these commands can gather much the same information about his virtual machine that an operator of a real machine gathers using the CPU console.

The CP commands are eight characters or less in length. The commands can be abbreviated by truncating them to the minimum permitted length shown in the format description. When truncation is permitted, the shortest acceptable version of the command is represented by capital letters, with the optional part represented by lowercase letters. Note, however, that you can enter any CP command with any mixture of uppercase and lowercase letters.

The operands, if any, follow the command on the same line and must be separated from the command by a blank. Lines cannot be continued. Generally, the operands are positional, but some commands have reserved words and keywords to assist processing. Blanks must separate the command from any operands and the operands from each other.

Several of these commands (for example, STORE or DISPLAY) examine or alter virtual storage locations. When CP is in complete control of virtual storage (as in the case of DOS, MFT, MVT, PCP, CMS, and RSCS) these commands execute as expected. However, when the operating system in the virtual machine itself manipulates virtual storage (OS/VS1, OS/VS2, or DOS/VS), these CP commands should not be used.

Each CP user has one or more privilege classes as indicated in his VM/370 directory entry. Class G commands useful for debugging are discussed in the following paragraphs. For a discussion of all the CP Class G commands and the CP command privilege classes, refer to the _VM/370: CP Command Reference for General Users_. The remainder of this section discusses the CP Class G commands that provide material and techniques that are useful in debugging.

ADSTOP

Privilege Class: G

Use the ADSTOP command to halt the execution of a virtual machine at a virtual instruction address. Execution halts when the instruction at the address specified in the command is the next instruction to be executed. The format of the ADSTOP command is:

ADSTOP	{ hexloc } { OFF }

where:

hexloc is the hexadecimal representation of the virtual instruction address where execution is to be halted. Since ADSTOP modifies storage, an address specified within a shared segment results in the virtual machine being placed in nonshared mode with its own copy of the shared segment. A fresh copy of the shared segment is then loaded for the use of the other users.

OFF cancels any previous ADSTOP setting.

Usage Notes

1. When execution halts, the CP command mode is entered and a message is displayed. At this point, you may invoke other CP debugging commands. To resume operation of the virtual machine, issue the BEGIN command. Once an ADSTOP location is set, it may be removed by one of the following:

   - Reaching the virtual storage location specified in the ADSTOP command
   - Performing a virtual IPL or SYSTEM RESET
   - Issuing the ADSTOP OFF command
   - Specifying a different location with a new ADSTOP hexloc command

2. Since the ADSTOP function modifies storage by placing a CP SVC X'B3' at the specified location, you should not:

   - Examine the two bytes at the instruction address because CP does not verify that the location specified contains a valid CPU instruction.
   - Use the TRACE command with the INSTRUCT, BRANCH, or ALL operands if any traced instruction is located at the ADSTOP address.

3. Address stops may not be set in an OS/VS or DOS/VS virtual machine's virtual storage; address stops may be set only in the virtual=real partitions or regions of those virtual machines.

4. If the SVC handling portion of the virtual machine assist feature is enabled on your virtual machine, CP turns it off when an ADSTOP is set. When the address stop is removed, CP returns the assist feature SVC handling to its previous status.

<u>Response</u>

ADSTOP AT xxxxxx

The instruction whose address is xxxxxx is the next instruction scheduled for execution. The virtual machine is in a stopped state. Any CP command (including an ADSTOP command to set the next address stop) can be issued. Enter the CP command BEGIN to resume execution at the instruction location xxxxxx, or at any other location desired.

BEGIN

Privilege Class: G

Use the BEGIN command to continue or resume execution in the virtual machine at either a specified storage location or the location pointed to by the virtual machine's current program status word (PSW). The format of the BEGIN command is:

Begin	[hexloc]

where:

hexloc is the hexadecimal storage location where execution is to begin.

Usage Notes

1. When BEGIN is issued without hexloc, execution begins at the storage address pointed to by the current virtual machine PSW. Unless the PSW has been altered since the CP command mode was entered, the location stored in the PSW is the location where the virtual machine stopped.

2. When BEGIN is issued with a storage location specified, execution begins at the specified storage location. The specified address replaces the instruction address in the PSW, then the PSW is loaded.

Responses

None. The virtual machine begins execution.

DISPLAY

Privilege Class: G

Use the DISPLAY command to display the following virtual machine components at your terminal:

- Virtual storage locations (1st level virtual storage only; see Usage Notes.)
- Storage keys
- General registers
- Floating-point registers
- Control registers
- Program status word (PSW)
- Channel address word (CAW)
- Channel status word (CSW)

Note: Use the NETWORK DISPLAY command to display the content of 3704/3705 storage.

The format of the DISPLAY command is:

```
Display  | { [ hexloc1 ]  [ {-} [hexloc2] ] }
         | { [Khexloc1 ]  [ {:} [END    ] ] }
         | { [Lhexloc1 ]  [                ] }
         | { [Thexloc1 ]  [ {.} [bytecount] ] }
         | { [   0     ]  [     [END      ] ] }
         | {                                  }
         | { Greg1 [ {-} [reg2]             ] }
         | { Yreg1 [ {:} [END ]             ] }
         | { Xreg1 [                        ] }
         | {       [ {.} [regcount]         ] }
         | {       [     [END     ]         ] }
         | {                                  }
         | { Psw                              }
         | { CAW                              }
         | { CSW                              }
```

where:

hexloc1
Lhexloc1
Thexloc1
Khexloc1
0

is the first, or only, hexadecimal storage location that is to be displayed at the terminal. If L or no letter prefix is specified, the storage contents are displayed in hexadecimal. If T is specified, the storage contents are displayed in hexadecimal, with EBCDIC translation. If K is specified, the storage keys are displayed in hexadecimal.

If hexloc1 is not on a fullword boundary, it is rounded down to the next lower fullword.

If hexloc1 is not specified, the display begins at storage location 0. If L, T, or K are entered either without any operands, or followed immediately by a blank, the contents of all storage locations or all the storage keys are displayed. If L, T, or K are not specified and this is the first operand, then the default value of zero is assumed. The address, hexloc1, may be one to six hexadecimal digits; leading zeros are optional.

{-} hexloc2 {:} END — is the last of the range of hexadecimal storage locations whose contents are to be displayed at the terminal. Either a - or a : must be specified to display the contents of more than one location by storage address. If hexloc2 is not specified, the contents of all storage locations from hexloc1 to the end of virtual storage are displayed. If specified, hexloc2 must be equal to or greater than hexloc1 and within the virtual storage size. (See Usage Notes below for a discussion on discontiguous shared segments.) The address, hexloc2, may be from one to six hexadecimal digits; leading zeros are optional.

{.}bytecount END — is a hexadecimal integer designating the number of bytes of storage (starting with the byte at hexloc1) to be displayed at the terminal. The period (.) must be specified to display the contents of more than one storage location by bytecount. The sum of hexloc1 and bytecount must be an address that does not exceed the virtual machine size. (See Usage Notes below for a discussion on discontiguous shared segments.) If this address is nct on a fullword boundary, it is rounded up to the next higher fullword. The value, bytecount, must have a value of at least one and may be from one to six hexadecimal digits; leading zeros are optional.

Greg1 — is a decimal number from 0 to 15 or a hexadecimal integer from 0 to F representing the first, or only, general register whose contents are to be displayed at the terminal. If G is specified without a register number, the contents of all the general registers are displayed at the terminal.

Yreg1 — is an integer (0, 2, 4, or 6) representing the first, or only, floating-point register whose contents are to be displayed at the terminal. If Y is specified without a register number, the contents of all of the floating-point registers are displayed at the terminal.

Xreg1 — is a decimal number from 0 to 15 or a hexadecimal number from 0 to F representing the first, or only, control register whose contents are to be displayed at the terminal. If X is specified without a register number, the contents of all of the control registers are displayed at the terminal. If Xreg1 is specified for a virtual machine without extended mode operations available, only control register 0 is displayed.

{-}reg2 {:} END — is a number representing the last register whose contents are to be displayed at the terminal. Either a - or a : must be specified to display the contents of more than one register by register number. If reg2 is not specified, the contents of all registers from reg1 through the last register of this type are displayed. The operand, reg2, must be equal to or greater than reg1. If Greg1 or Xreg1 are specified, reg2 may be a decimal number from 0-15 or a hexadecimal number from 0-F. If Yreg1 is specified, reg2 may be 0, 2, 4, or 6. The contents of registers reg1 through reg2 are displayed at the terminal.

{.}regcount
END
: is a decimal number from 1 to 16 or a hexadecimal number from 1 to F specifying the number of registers (starting with reg1) whose contents are to be displayed at the terminal. If the display type G or X is specified, regcount can be a decimal number from 1 to 16 or a hexadecimal number from 1 to F. If display type Y is specified, regcount must be 1, 2, 3, or 4. The sum of reg1 and regcount must be a number that does not exceed the maximum register number for the type of registers being displayed.

PSW
: displays the current virtual machine PSW (program status word) as two hexadecimal words.

CAW
: displays the contents of the CAW (channel address word at hexadecimal location 48) as one hexadecimal word.

CSW
: displays the contents of the CSW (channel status word at hexadecimal location 40) as two hexadecimal words.

Usage Notes

1. Only first level storage (storage that is real to the virtual machine) can be displayed. Operating systems such as DOS/VS and OS/VS have virtual storage of their own. This second level virtual storage cannot be displayed directly. The user or the virtual operating system is responsible for converting any second level storage locations to first level storage locations before issuing the command.

2. If a command line with an invalid operand is entered, the DISPLAY command terminates when it encounters the invalid operand; however, any previous valid operands are processed before termination occurs. Multiple storage locations, registers, and control words can be displayed using a single command line.

3. When multiple operands are entered on a line for location or register displays, the default display type is the same as the previous explicit display type. The explicit specification of a display type defines the default for subsequent operands for the current display function. Blanks are used to separate operands or sets of operands if more than one operand is entered on the same command line. Blanks must not be used to the right or left of the range or length delimiters (: or - or .), unless it is intended to take the default value of the missing operand defined by the blank. For example:

```
display 10 20 T40 80 G12 5 L60-100
```

displays the following, respectively:

hexadecimal location 10
hexadecimal location 20
hexadecimal location 40 with EBCDIC translation
hexadecimal location 80 with EBCDIC translation
general register 12
general register 5
hexadecimal locations 60 through 100

4. To terminate the DISPLAY function while data is being displayed at the terminal, press the Attention key (or its equivalent). When the display terminates, another command may be entered.

5. The DISPLAY command does not distinguish between shared and non-shared storage; it displays any of the virtual machine's addressable storage whether shared or not.

6. Use the DISPLAY command to display the contents of various storage locations, registers, and control words at the terminal. By examining this type of information during the program's execution, you may be able to determine the cause of program errors. Usually, an address stop is set to stop the program execution at a specified point. The system enters the CP environment and you may then issue the DISPLAY command.

7. When you must examine large portions of storage, use the DUMP command rather than the DISPLAY command. Because the terminal operates at a much slower speed than the printer, only limited amounts of storage should be printed (via the DISPLAY command) at the terminal.

8. When running with a discontiguous saved segment (DCSS), you can display storage locations outside the range of your virtual machine size if they are within the DCSS. If there exist locations between the upper limit of your virtual machine and the address at which the DCSS was saved, an attempt to display those locations (or associated keys) will result in a "non-addressable storage" message.

Responses

One or more of the following responses is displayed, depending upon the operands specified.

Displaying Storage Locations

xxxxxx word1 word2 word3 word4 [key] *EBCDIC TRANSLATION*

This is the response you receive when you display storage locations; xxxxxx is the hexadecimal storage location of word1. Word1 is displayed (word-aligned) for a single location specification. Up to four words are displayed on a line, followed, optionally, by an EBCDIC translation of those four words. Periods are represented by nonprintable characters. Multiple lines are used (if required) for a range of locations. If translation to EBCDIC is requested (Thexloc), alignment is made to the next lower 16-byte boundary; otherwise, alignment is made to the next lower fullword boundary. If the location is at a 2K page boundary, the key for that page is also displayed.

Displaying Storage Keys

xxxxxx TO xxxxxx   KEY = kk

This is the response you receive when you display storage keys; xxxxxx is a storage location and kk is the associated storage key.

Displaying General Registers

GPR n = genreg1 genreg2 genreg3 genreg4

This is the response you receive when you display general registers; n is the register whose contents are genreg1. The contents of the following consecutive registers are genreg2, genreg3, and so on. The contents of the registers are displayed in hexadecimal. Up to four registers per line are displayed for a range of registers. Multiple lines are displayed if required, with a maximum of four lines needed to display all 16 general registers.

Displaying Floating-Point Registers

FPR n = xxxxxxxxxxxxxxxx .xxxxxxxxxxxxxxxxx E xx

This is the response you receive when you display floating-point registers; n is the even-number floating-point register whose contents are displayed on this line. The contents of the requested floating-point registers are displayed in both the internal hexadecimal format and the E format. One register is displayed per line. Multiple lines are displayed for a range of registers.

Displaying Control Registers

ECR n = ctlreg1 ctlreg2 ctlreg3 ctlreg4

This is the response you receive when you display control registers; n is the register whose contents are ctlreg1. The contents of the following consecutive registers are ctlreg2, ctlreg3, and so on. The contents of the requested control registers are displayed in hexadecimal. Up to four registers per line are displayed. Multiple lines are displayed if required.

Displaying the PSW

PSW = xxxxxxxx xxxxxxxx

The contents of the PSW are displayed in hexadecimal.

Displaying the CAW

CAW = xxxxxxxx

The contents of the CAW (hexadecimal location 48) are displayed in hexadecimal.

Displaying the CSW

CSW = xxxxxxxx xxxxxxxx

The contents of the CSW (hexadecimal location 40) are displayed in hexadecimal.

DUMP

Privilege Class: G

Use the DUMP command to print the contents of various components of the virtual machine on the virtual spooled printer. The following items are printed:

- Virtual program status word (PSW)
- General registers
- Floating-point registers
- Control registers (if you have the ECMODE option specified in your VM/370 directory entry)
- Storage keys
- Virtual storage locations (1st level virtual storage only; see Usage Notes.)

Note: Use the NETWORK DUMP command to dump the contents of 3704/3705 storage. This command is described in the *VM/370: Operator's Guide*.

The format of the DUMP command is:

```
+------------------------------------------------------------------------+
| DUMP  | / [Lhexloc1] [{-} [hexloc2   ]] \                            |
|       | | [Thexloc1] [{:} [END       ]] |      [*dumpid]             |
|       | { [ hexloc1] [                ] }                            |
|       | | [   0    ] [{.} [bytecount ]] |                              |
|       | \            [    [END       ]] /                              |
+------------------------------------------------------------------------+
```

where:

Lhexloc1
Thexloc1
hexloc1
0 (default)
: is the first or only hexadecimal storage location to be dumped. If you enter L or T without operands, the contents of all virtual storage locations are dumped.

  The address, hexloc1, may be one to six hexadecimal digits; leading zeros are optional. If hexlcc1 is not specified, the dump begins at storage location 0.

  If hexloc1 is not on a fullword boundary, it is rounded down to the next lower fullword.

{-} hexloc2
{:} END (default)
: is the last hexadecimal storage location whose contents are to be dumped to the printer. The operand, hexloc2, must be equal to or greater than hexloc1 and within the virtual storage size. To dump to the end of storage, you can specify END instead of hexloc2 or you can leave the field blank, since the default is END. If you specify :END or -END, the contents of storage from hexloc1 to END are dumped. The contents of storage locations hexloc1 through hexloc2 are printed with EBCDIC translation at the printer. The operand, hexloc2, may be from one to six hexadecimal digits; leading zeros are optional.

{.}bytecount END — is a hexadecimal integer designating the number of bytes of storage (starting with the byte at hexloc1) to be dumped to the printer. The period (.) must be specified to dump the contents of more than one storage location by bytecount. The sum of hexloc1 and bytecount must be an address that does not exceed the virtual machine size. If this address is not on a fullword boundary, it is rounded up to the next highest fullword. The value, bytecount, must be one or greater and can be no longer than six hexadecimal digits. Leading zeros are optional.

*dumpid — can be entered for descriptive purposes. If specified, it becomes the first line printed preceding the dump data. Up to 100 characters, with or without blanks, may be specified after the asterisk prefix. No error messages are issued, but only 100 characters are used, including asterisks and embedded blanks.

Usage Notes

1. Only first level storage (storage that is real to the virtual machine) can be dumped. Operating systems such as DOS/VS and OS/VS have virtual storage of their own. This second level virtual storage cannot be dumped directly. The user or the virtual operating system is responsible for converting any second level storage locations to first level storage locations before issuing the command.

2. The CP DUMP command executes in an area of storage separate from your virtual machine storage and does not destroy any portion of your storage.

3. The DUMP command prints the virtual PSW and the virtual registers (general, floating-point, and control). If only this information is desired, at least one virtual address must be specified, such as

```
DUMP 0
```

4. The output format for the virtual storage locations is eight words per line with the EBCDIC translation on the right. Each fullword consists of eight hexadecimal characters. All the rest of the information (PSW, general and floating-point registers, and storage keys) is printed in hexadecimal. If you have the ECMODE option in your VM/370 directory entry, the control registers are also printed. To print the dump on the real printer, a CLOSE command must be issued for the spooled virtual printer.

5. Normally, you should define beginning and ending dump locations in the following manner:

```
dump Lhexloc1-hexloc2
dump Lhexloc1.bytecount
dump Lhexloc1-hexloc2 hexloc1.bytecount * dumpid
```

If, however, a blank follows the type character (L or T) or the character and the hexloc, the default dump starting and ending locations are assumed to be the beginning and/or end of virtual storage. Blanks are used to separate operands or sets of operands if more than one operand is entered on the same command line. Blanks must not be used to the right or left of range or length delimiters (: or - or .), unless it is intended to take the default value of the missing operand defined by the blank. Thus, all of the following produce full storage dumps:

```
dump 1      dump t:       dump 0-end
dump t      dump 1.       dump 1:end
dump -      dump t.       dump t:end
dump :      dump 0-       dump 0:end
dump .      dump 0:       dump 1.end
dump 1-     dump 0.       dump t.end
dump t-     dump 1-end    dump 0.end
dump 1:     dump t-end
```

The following produces three full dumps:

```
dump 1 . t
dump - . :
```

6. When running with a discontiguous saved segment (DCSS), you can dump storage locations outside the range of your virtual machine size if they are within the DCSS. If there exist locations between the upper limit of your virtual machine and the address at which the DCSS was saved, an attempt to dump those locations (or associated keys) will result in a "non-addressable storage" message appearing in the printer output.

Responses

As the dump progresses, the following message is displayed at the terminal; indicating that the dump is continuing from the next 64k boundary:

DUMPING LOC hexloc

where hexloc is the segment (64K) boundary address for the dump continuation, such as 020000, 030000, or 040000.

If you press the Attention key, or its equivalent, on the terminal while the message is being displayed, the dump function is terminated.

COMMAND COMPLETE

is the response indicating normal completion of the dump function.

Privilege Class: G

Use the SET command to control various functions within your virtual system. The format of the SET command is:

```
SET   | ACNT      {ON }
      | MSG       {OFF}
      | WNG
      | IMSG
      | RUN
      | LINEDit
      | ECmode
      | ISAM
      | NOTRans
      | PAGEX
      |
      | EMSG      {ON  }
      |           {OFF }
      |           {CODE}
      |           {TEXT}
      |
      | TIMER     {ON  }
      |           {OFF }
      |           {REAL}
      |
      | ASsist    { [ON]  [SVC  ] }
      |           {       [NOSVC] }
      |           {               }
      |           {  OFF          }
      |
      | PFnn [IMMed  ] [pfdata1#pfdata2#...pfdatan]
      |      [DELayed]
      |
      | PFnn [TAB n1 n2 ...   ]
      |
      | PFnn COPY [resid]
      |
      | PFnn COPY [cuu]
```

where:

ACNT {ON | OFF} controls whether accounting information is displayed at the terminal or not (ON and OFF, respectively) when the operator issues the CP ACNT command. When you log on VM/370, ACNT is set on.

MSG {ON | OFF} controls whether messages sent by the MSG command from other users are to be received at the terminal. If ON is specified, the messages are displayed. If OFF is specified, no messages are received. In addition to controlling messages generated by the MSG command, spooling messages generated by users sending punch, printer or reader files to another virtual machine are also suppressed if OFF is specified. When you log on VM/370, MSG is set on.

WNG {ON | OFF} controls whether warning messages are displayed at the terminal. If ON is specified, all warning messages sent via the CP WARNING command from the system operator are received at the terminal. If OFF is specified, no warning messages are received. When you log on VM/370, WNG is set on.

IMSG {ON | OFF} controls whether certain informational responses issued by the CP CHANGE, DEFINE, DETACH, ORDER, PURGE, and TRANSFER commands are displayed at the terminal or not. The descriptions of these CP commands tell which responses are affected. If ON is specified the informational responses are displayed. If OFF is specified, they are not. The SET IMSG ON or OFF command line has no effect on the handling of error messages set by the SET EMSG command. When you log on VM/370, IMSG is set on.

RUN {ON | OFF} controls whether the virtual machine stops when the Attention key is pressed. ON allows you to activate the Attention key (causing a read of a CP command) without stopping your virtual machine. When the CP command is entered, it is immediately executed and the virtual machine resumes execution. OFF places the virtual machine in the normal CP environment, so that when the Attention key is pressed, the virtual machine stops. When you log on VM/370, RUN is set off.

LINEDIT {ON | OFF} controls the line editing functions. ON specifies that the line editing functions and the symbols of the VM/370 system are to be used to edit virtual CPU console input requests. This establishes line editing features in systems that do not normally provide them. OFF specifies that no character or line editing is to be used for the virtual machine operating system. When you log on VM/370, LINEDIT is set on.

ECMODE {ON | OFF} controls whether the virtual machine operating system may use System/370 extended control mode and control registers 1 through 15. Control register zero may be used with ECMODE either ON or OFF. When you log on VM/370, ECMODE is set according to the user's directory option; ON if ECMODE was specified and OFF if not.

Note: Execution of the SET ECMODE {ON|OFF} command always causes a virtual system reset.

ISAM {ON | OFF} controls whether additional checking is performed on virtual I/O requests to DASD in order to support the OS Indexed Sequential Access Method (ISAM). When you log on VM/370, ISAM is set according to the user's directory options; ON if ISAM was specified and OFF if not.

NOTRANS {ON | OFF} controls CCW translation for CP. NOTRANS can be specified only by a virtual machine that occupies the virtual=real space. It causes all virtual I/O from the issuing virtual machine to bypass the CP CCW translation except under the following conditions:

- SIO tracing active
- 1st 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 non-dedicated device
- Pending device status

Any of the above conditions will force CCW translation.

To be in effect in the virtual=real environment, SET NOTRANS ON must be issued after the virtual=real machine is loaded via the IPL command. (IPL sets the NOTRANS option to an OFF condition.)

PAGEX {ON | OFF} controls the pseudo page fault portion of the VM/VS Handshaking feature. PAGEX ON or OFF should only be issued for an OS/VS1 virtual machine that has the VM/VS Handshaking feature active. It can only be specified for a virtual machine that has the extended control mode (ECMODE) option. PAGEX ON sets on the pseudo page fault portion of handshaking; PAGEX OFF sets it off. When you log on to VM/370, PAGEX is set OFF. Also, each time you IPL VS1 in your virtual machine PAGEX is set off. If you want to use the pseudo page fault handling portion of handshaking you must issue SET PAGEX ON after you IPL VS1.

EMSG {ON | OFF | CODE | TEXT} controls error message handling. ON specifies that both the error code and text are displayed at the terminal. TEXT specifies that only text is displayed. CODE specifies that only the error code is to be displayed. OFF specifies that no error message is to be displayed. When you log on VM/370, EMSG is set to TEXT.

If the console is being spooled, the OFF setting is ignored for the spooled output and the full error message appears in the spooled output. The other three settings result in spooled output that matches the console printout.

Note: CMS recognizes EMSG settings for all error (E), information (I), and warning (W) messages, but ignores the EMSG setting and displays the complete message (error code and text) for all response (R), severe error (S), and terminal (T) messages.

TIMER {ON | OFF | REAL} controls the virtual timer. ON specifies that the virtual timer is to be updated only when the virtual CPU is running. OFF specifies that the virtual timer is not to be updated. REAL specifies that the virtual timer is to be updated during virtual CPU run time and also during virtual wait time. If the REALTIMER option is specified in your VM/370 directory entry, TIMER is set to REAL when you log on; otherwise it is set to ON when you log on.

```
ASSIST ⎧ ┌    ┐ ┌      ┐ ⎫
       ⎨ |ON | |SVC   | ⎬
       ⎪ |   | |NOSVC | ⎪
       ⎪ └    ┘ └      ┘ ⎪
       ⎩  OFF            ⎭
```

controls the availability of the virtual machine assist feature for your virtual machine. The assist feature is available to your virtual machine when you log on if (1) the real CPU has the feature installed and (2) the system operator has not turned the feature off. The SVC handling portion of the assist feature is invoked when you log on unless your VM/370 directory entry has the SVCOFF option. Issue the QUERY SET command line to see if the assist feature is activated and whether the assist feature or VM/370 is handling SVC interruptions. All SVC 76 requests are passed to CP for handling, regardless of the SVC and NOSVC operands. If you issue the SET ASSIST command line and specify SVC or NOSVC while the virtual machine assist feature is turned off, the appropriate bits are set. Later, if the feature is turned on again, the operand you specified while it was off becomes effective. ON sets the assist feature on for the virtual machine; OFF turns it off. SVC specifies that the assist feature handles all SVC interruptions except SVC 76 for the virtual machine; NOSVC means VM/370 handles all the SVC interruptions. See the _VM/370: System Programmer's Guide_ for information on how to use the assist feature.

```
     ┌         ┐
PFnn |IMMED    | [pfdata1#pfdata2#...pfdatan]
     |DELAYED  |
     └         ┘
```

defines a program function for a program function key on a 3277 Display Station and indicates when that function is to be executed. See the _VM/370: Terminal User's Guide_ for a description of how to use the 3277 program function keys.

nn
: is a number from 1 (or 01) to 12 that corresponds to a key on a 3277. The program function is a programming capability you create by defining a series of VM/370 commands or data you want executed. This series of commands executes when you press the appropriate program function key.

IMMED
: specifes that the program function is executed immediately after you press the program function key.

DELAYED
: specifies that execution of the program function is delayed for a display terminal. When the program function is entered, it is displayed in the input area and not executed until you press the Enter key. DELAYED is the default value for display terminals.

pfdata1#pfdata2#...pfdatan
: defines the VM/370 command or data lines that constitute the program function. If more than one command line is to be entered, the pound sign (#) must separate the lines. If you use the pound sign (#) to separate commands that you want executed with the designated PF key, you must precede the command

line with #CP, turn line editing off, or precede each pound sign with the logical escape character ("). For further explanation, see the "Usage Notes" section that follows. If no command lines are entered, PFnn is a null command. Program functions cannot be embedded within one another.

PFnn TAB n1 n2 ...
: specifies a program function number to be associated with tab settings on a terminal. The number of the PF key, nn can be a value from 1 (or 01) to 12. For examples of how this feature is used, see the VM/370: CMS User's Guide.

TAB
: is a keyword identifying the tab function. The tab settings (n1 n2 ...) may be entered in any order.

PFnn COPY [resid]
: specifies that the program function key, numbered nn, performs a COPY function for a remote 3270 terminal. nn must be a value from 1 (or 01) to 12. The COPY function produces a printed output of the entire screen display at the time the PF key is actuated. The output is printed on an IBM 3284, 3286 or 3288 printer connected to the same control unit as your display terminal.

resid
: may be specified if more than one printer is connected to the same control unit as your display terminal. It is a three-character hexadecimal resource identification number assigned to a specific printer. If resid is entered, the printed copy is directed to a specific printer; if not, the copy is printed on the printer with the lowest resid number. The resid numbers of the printers available to your display terminal can be obtained from your system operator. If only one printer is available, as with the 3275 Display Station, resid need not be specified.

PFnn COPY [cuu]
: specifies that the program function key, numbered nn, performs a COPY function for a local 3270 terminal. nn must be a value from 1 (or 01) to 12. When the PF key is actuated, the COPY function produces a printed output of the entire local screen display except for the status field which is replaced with blanks.

cuu
: is the real hardware address of the 3284, 3286, or 3288 printer, and may specify a printer that is on a different control unit than the one to which your 3270 is attached. If you do not specify cuu, the printer with the lowest cuu that is available on the same control unit as your 3270 will be selected.

Note: For both remote and local COPY functions:

You will receive a NOT ACCEPTED message, displayed in the screen status field of your 3270, if any of the the following situations occur:

- The printer is already busy, or all printers are busy.
- The printer is turned off.
- The printer is out of paper or is in any other intervention required condition.

* The designated device is not a 328X type printer.
* The SET PFnn COPY command is invalid.

You may include your own identification on the printed output by entering the data into the user input area of the screen before you press the PF key. The identification appears on the last two lines of the printed copy.

Usage Notes

1. Both SET PFnn TAB and SET PFnn COPY are immediate commands: their function is executed immediately upon pressing the appropriate program function key. If you insert the keyword DELAYED after the PFnn operand, the command will be accepted, however, the program function will still be executed immediately.

2. If you use the SET PFnn command to set up a series of concatenated commands, you should be aware of the following situation:

   If you enter one of the following commands while in CMS mode:

   ```
   SET PF02 IMMED Q RDR#Q PRT#Q PUN

           -- or --

   CP SET PF02 IMMED Q RDR#Q PRT#Q PUN
   ```

   and then press the Enter key:

   1. The Enter key causes immediate execution,
   2. Only the Q PRT and Q PUN commands execute, and
   3. Q PRT and Q PUN are stripped from the PF02 key assignment leaving Q RDR, which was not executed.

The following examples demonstrate two methods for avoiding the problem.

Example 1

Enter one of the following commands while in CMS mode:

```
#CP SET PF02 IMMED Q RDR#Q PRT#Q PUN

           -- or --

CP SET PF02 IMMED Q RDR"#Q PRT"#Q PUN

           -- or --

SET PF02 IMMED Q RDR"#Q PRT"#Q PUN
```

Now press the Enter key.

CP assigns the three QUERY commands as functions of the PF02 key. Pressing the PF02 key executes the three QUERY commands.

Example 2

Enter the following command while in CMS mode:

SET LINEDIT OFF

and press the Enter key.

Then enter:

SET PF02 IMMED Q RDR#Q PRT#Q PUN

-- or --

CP SET PF02 IMMED Q RDR#Q PRT#Q PUN

and press the Enter key.

CP assigns the three QUERY commands as functions of the PF02 key.

Then enter:

SET LINEDIT ON

and press the Enter key.

Pressing the PF02 key executes the three QUERY commands.

Responses

None

STORE

Privilege Class: G

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

- Virtual storage locations (1st level virtual storage only; see Usage Notes)
- General registers
- Floating-point registers
- Control registers (if available)
- Program status word

The STORE command can also save virtual machine data in low storage. The format of the STORE command is:

```
STore  | ( hexloc                                       )
       | ( Lhexloc    hexword1 [hexword2...]            )
       |                                                 
       | ( Shexloc    hexdata...                        )
       |                                                 
       | ( {Greg}                                       )
       | ( {Xreg}     hexword1 [hexword2...]            )
       |                                                 
       | ( {Yreg}     hexdword1 [hexdword2...]          )
       |                                                 
       | ( Psw      [hexword1] hexword2                 )
       |                                                 
       | ( STATUS                                       )
```

where:

hexloc
Lhexloc hexword1 [hexword2...]
stores the specified data (hexword1 [hexword2...]) in successive fullword locations starting at the address specified by hexloc. The smallest group of hexadecimal values that can be stored using this form is one fullword. Either form (hexloc or Lhexloc) can be used.

If hexloc is not on a fullword boundary, it is rounded down to the next lower fullword.

hexword1 [hexword2...]
each represents up to sixteen hexadecimal digits. If the value being stored is less than a fullword (eight hexadecimal digits), it is right-adjusted in the word and the high order bytes of the word are filled with zeros. If two or more hexwords are specified, they must be separated by one or more blanks.

Shexloc hexdata...
stores the data specified (hexdata...) in the address specified by hexloc, without word alignment. The shortest string that can be stored is one byte (two hexadecimal digits). If the string contains an odd number of characters, the last character is not stored, an error message is sent, and the function is terminated.

hexdata...
is a string of two or more hexadecimal digits with no embedded blanks.

Greg hexword1 [hexword2...]
stores the hexadecimal data (hexword1 [hexword2...]) in successive general registers starting at the register specified by reg. The reg operand must be either a decimal number from 0-15 or a hexadecimal digit from 0-F.

hexword1 [hexword2...]
each represents up to eight hexadecimal digits. If the value being stored is less than a fullword (eight hexadecimal digits), it is right-adjusted in the word and the high order bytes of the word are filled with zeros. If two or more hexwords are specified, they must be separated by one or more blanks.

Xreg hexword1 [hexword2...]
stores the hexadecimal data (hexword1 [hexword2...]) in successive control registers starting at the register specified by reg. The reg operand must either be a decimal number from 0-15 or a hexadecimal digit from 0-F. If the virtual machine is in basic control mode, you can store data in register 0 only.

hexword1 [hexword2...]
each represents up to eight hexadecimal digits. If the value being stored is less than a fullword (eight hexadecimal digits), it is right-adjusted in the word and the high order bytes of the word are filled with zeros. If two or more hexwords are specified, they must be separated by one or more blanks.

Yreg hexdword1 [hexdword2...]
stores the hexadecimal data (hexdword1 [hexdword2...]) in successive floating-point registers starting at the register specified by reg. The reg operand must be a digit from 0-7. If reg is an odd number, it is adjusted to the preceding even number.

hexdword1 [hexdword2...]
each represents up to sixteen hexadecimal digits. If the value being stored is less than a doubleword (sixteen hexadecimal digits), it is left justified in the doubleword and low order positions are filled with zeros. If two more more hexdwords are specified, they must be separated by one or more blanks.

PSW [hexword1] hexword2
stores the hexadecimal data in the first and second words of the virtual machine's program status word (PSW). If only hexword2 is specified, it is stored into the second word of the PSW.

[hexword1] hexword2
each represents up to eight hexadecimal digits. These operands must be separated by one or more blanks. If the value being stored is less than a fullword (eight hexadecimal digits), it is right-adjusted in the word and the high order bytes of the word are filled with zeros.

STATUS
stores selected virtual machine data in certain low storage locations of the virtual machine, simulating the hardware

store status facility. These locations are permanently assigned locations in real storage. To use the STATUS operand, your virtual machine must be in the extended control mode. The STATUS operand should not be issued for CMS virtual machines or for DOS virtual machines generated for a CPU smaller than a System/360 Model 40. The STATUS operand stores the following data in low storage:

Decimal Address	Hexadecimal Address	Length in Bytes	Data
216	D8	8	CPU Timer
224	E0	8	Clock Comparator
256	100	8	Current PSW
352	160	32	Floating-point registers 0-6
384	180	64	General registers 0-15
448	1C0	64	Control registers 0-15

Usage Notes

1. Only first level storage (storage that is real to the virtual machine) can be stored into. Operating systems such as DOS/VS and OS/VS have virtual storage of their own. This second level virtual storage cannot be stored into directly. The user or the virtual operating system is responsible for converting any second level storage locations to first level storage locations.

2. The operands may be combined in any order desired, separated by one or more blanks, for up to one full line of input. If an invalid operand is encountered, an error message is issued and the store function is terminated. However, all valid operands entered, before the invalid one, are processed properly.

3. If you combine the operands for storing into storage, registers, the PSW, or the status area on a single command line, all operands must be specified; default values do not apply in this case.

4. If the STORE command is used by your virtual machine to alter the contents of a shared segment, your virtual machine will be placed in non-shared mode with your own copy of the shared segment. A fresh copy of the shared segment is then loaded for use by the other users.

5. With the STORE command, data is stored either in units of one word with fullword boundary alignment or in units of one byte without alignment.

6. The STORE STATUS command stores data in the extended logout area. The STORE STATUS command stores CPU Timer and Clock Comparator values that may then be displayed at the terminal via the DISPLAY command. The procedure is the only way to get timer information at the terminal.

Response

STORE COMPLETE

is the response at the successful completion of the command.

SYSTEM

Privilege Class: G

Use the SYSTEM command to simulate the action of the RESET and RESTART buttons on the real computer console, and to clear storage. The format of the SYSTEM command is:

```
┌─────────────────────────────────────────────────────────────────┐
│ SYStem  │ ⎧CLEAR  ⎫                                              │
│         │ ⎨RESET  ⎬                                              │
│         │ ⎩RESTART⎭                                              │
└─────────────────────────────────────────────────────────────────┘
```

where:

CLEAR clears virtual storage and virtual storage keys to binary zeros.

RESET clears all pending interruptions and conditions in the virtual machine.

RESTART simulates the hardware system RESTART function by storing the current PSW at virtual location eight and loading, as the new PSW, the doubleword from virtual location zero. Interrupt conditions and storage remain unaffected.

Usage Notes

1. The RESET function and the CLEAR function leave the virtual machine in a stopped state.

2. After issuing the SYSTEM command with RESET or CLEAR specified, either STORE a PSW and issue BEGIN or issue BEGIN with a hexadecimal storage location specified, to resume operation. The virtual machine automatically restarts at the location specified in the new PSW (which is loaded from the doubleword at location zero) after the SYSTEM RESTART command is processed.

Responses

STORAGE CLEARED - SYSTEM RESET

is the response given if the command SYSTEM CLEAR is entered.

SYSTEM RESET

is the response given if the command SYSTEM RESET is entered.

If the command SYSTEM RESTART is entered, no response is given; the virtual machine resumes execution at the address in the virtual PSW loaded from virtual storage location zero.

TRACE

Privilege Class: G

Use the TRACE command to trace specified virtual machine activity and to record the results at the terminal, on a virtual spooled printer, or on both terminal and printer. The format of the TRACE command is:

```
┌──────────────────────────────────────────────────────────────────────┐
│                                                                      │
│  TRace  │ ┌ SVC      ┐¹ ┌ PRINTer                    ┐ ┐            │
│         │ │ I/O      │  │ ┌          ┐ ┌       ┐      │ │            │
│         │ │ PROgram  │  │ │TERMinal  │ │NORun  │      │ │            │
│         │ │ EXTernal │  │ │BOTH      │ │RUN    │      │ │            │
│         │ │ PRIV     │  │ └          ┘ └       ┘      │ │            │
│         │ < SIO      >  │                             │ >            │
│         │ │ CCW      │  │  OFf                        │ │            │
│         │ │ BRanch   │  └                             ┘ │            │
│         │ │ INSTruct │                                  │            │
│         │ │ ALL      │                                  │            │
│         │ └ CSW      ┘                                  │            │
│         │                                               │            │
│         │   END                                         ┘            │
│──────────────────────────────────────────────────────────────────────│
│ ¹More than one of these activities may be traced by using a single   │
│  TRACE command. For example:                                         │
│                                                                      │
│     TRACE SVC PROGRAM SIO PRINTER                                    │
└──────────────────────────────────────────────────────────────────────┘
```

where:

SVC traces virtual machine SVC interruptions.

I/O traces virtual machine I/O interruptions.

PROGRAM traces virtual machine program interruptions.

EXTERNAL traces virtual machine external interruptions.

PRIV traces all virtual machine non-I/O privileged instructions.

SIO traces TIO, CLRIO, HIO, HDV, and TCH instructions to all virtual devices. Also traces SIO and SIOF instructions for nonconsole and nonspool devices only.

CCW traces virtual and real CCWs for nonspool nonconsole device I/O operations. When CCW tracing is requested, SIO and TIO instructions to all devices are also traced.

BRANCH traces virtual machine interruptions, PSW instructions, and successful branches.

INSTRUCT traces all instructions, virtual machine interruptions, and successful branches.

ALL traces all instructions, interruptions, succesful branches, privilege instructions, and virtual machine I/O operations.

CSW provides contents of virtual and real channel status words at I/O interruption.

END	terminates all tracing activity and prints a termination message.
PRINTER	directs tracing output to a virtual spooled printer.
<u>TERMINAL</u>	directs tracing output to the terminal (virtual machine console).
BOTH	directs tracing output to both a virtual spooled printer and the terminal.
OFF	halts tracing of the specified activities on both the printer and terminal.
<u>NORUN</u>	stops program execution after the trace output to the terminal and enters the CP command environment. <u>Note</u>: If a Diagnose code X'008' is being traced, NORUN has no effect and program execution does not stop.
RUN	continues the program execution after the trace output to the terminal has completed and does not enter the CP command environment.

<u>Usage Notes</u>

1. If your virtual machine has the virtual=real option and NOTRANS set on, CP forces CCW translation while tracing either SIO or CCW. When tracing is terminated with the TRACE END command, CCW translation is bypassed again.

2. If the virtual machine assist feature is enabled on your virtual machine, CP turns it off while tracing SVC, PRIV, BRANCH, INSTRUCT, or ALL activities. After the tracing is terminated with the TRACE END command line, CP turns the assist feature on again.

3. If trace output is being recorded at the terminal, the virtual machine stops execution and CP command mode is entered after each output message. This simulates the instruction step function.

   However, all processing associated with the event being traced will be completed and, therefore, execution may have stopped after an instruction has executed and the PSW has been updated.

   For example, a privileged instruction traced with the PRIV operand will stop after the privileged instruction executes, whereas the same instruction traced with the ALL operand will stop before the instruction executes.

   To determine whether the traced instruction has executed, display the virtual machine PSW.

   To resume operation of the virtual machine, the BEGIN command must be entered. If the RUN operand is specified, the virtual machine is not stopped after each output message.

4. If trace output is being recorded on a virtual spooled printer, a CLOSE command must be issued to that printer in order for the trace output to be printed on the real printer.

5. Successful branches to the next sequential instruction and branch-to-self instructions are not detected by TRACE.

6. Instructions that modify or examine the first two bytes of the next sequential instruction cause erroneous processing for BRANCH and INSTRUCT tracing.

7. When tracing on a virtual machine with only one printer, the trace data is intermixed with other data sent to the virtual printer. To separate trace information from other data, define another printer with a lower virtual address than the previously defined printer. For example, on a system with 00E defined as the only printer, define a second printer as 00B. The regular output goes to 00E and the trace output goes to 00B.

8. If the BRANCH, INSTRUCT, or ALL activities are being traced by a virtual machine using a shared system, the virtual machine is placed in nonshared mode with its own copy of the shared segment. A fresh copy of the shared segment is then loaded for use by the other users.

9. I/O operations for virtual channel-to-channel adapters, with both ends connected to the same virtual machine, cannot be traced.

10. Use the TRACE command to trace specified virtual machine activity and to record the results at the terminal, at a virtual printer, or at both. This command is useful in debugging programs because it allows you to trace only the information that pertains to a particular problem.

11. If your virtual machine is doing I/O that results in program controlled interruptions (PCIs), and you are tracing I/O or CSW activity, some of the PCIs may not be traced. This situation arises when the system is extending its free storage area and the additional demand on available free storage would cause a system abend.

Responses

The following symbols are used in the responses received from TRACE:

Symbol	Meaning
vvvvvv	virtual storage address
tttttt	virtual transfer address or new PSW address
rrrrrr	real storage address
xxxxxxxx	virtual instruction, channel command word, CSW status
yyyyyyyy	real instruction, CCW
ss	argument byte (SSM-byte) for SSM instruction
ns	new system mask after execution of STOSM/STNSM
zz	low order byte of R1 register in an execute instruction (not shown if R1 register is register 0)
zzzzzzzz	referenced data
type	virtual device name (DASD, TAPE, LINE, CONS, RDR, PRT, PUN, GRAF, DEV)
V vadd	virtual device address
R radd	real device address
mnem	mnemonic for instruction
int	interruption type (SVC, PROG, EXT, I/O)
code	interruption code number (in hexadecimal)
CC n	condition-code number (0, 1, 2, or 3)
IDAL	Indirect data address list
***	virtual machine interrupt
:::	privileged operations
==>	transfer of control

TRACE STARTED

This response is issued when tracing is initiated.

TRACE ENDED

This response is issued when tracing is suspended.

TCH, TIO, CLRIO, HIO, HDV, SIO, or SIOF

TCH

```
I/O vvvvvv TCH xxxxxxxx type vadd CC n
```

TIO, CLRIO, HIO, or HDV

```
I/O vvvvvv mnem xxxxxxxx type vadd CC n type radd CSW xxxx
```

SIO or SIOF

```
I/O vvvvvv mnem xxxxxxxx type vadd CC n type radd CSW xxxx CAW vvvvvvvv
```

CCW:

```
CCW vvvvvv xxxxxxxx xxxxxxxx rrrrrr yyyyyyyy yyyyyyyy
CCW  IDAL  vvvvvvvv vvvvvvvv  IDAL  00rrrrrr 00rrrrrr
CCW  SEEK  xxxxxxxx xxxxxx    SEEK  yyyyyyyy yyyy
```

The IDAL or SEEK line is included only if applicable. The virtual IDAL is not printed if the real CCW operation code does not match the real CCW.

INSTRUCTION TRACING:

Privileged Instruction:

```
::: vvvvvv SSM   xxxxxxxx ss             (normal SSM)
::: vvvvvv SSM   xxxxxxxx ss  tttttt     (switch to/from translate mode)
::: vvvvvv STOSM xxxxxxxx ns             (normal STOSM)
::: vvvvvv STOSM xxxxxxxx ns  tttttt     (switch to translate mode)
::: vvvvvv STNSM xxxxxxxx ns             (normal STNSM)
::: vvvvvv STNSM xxxxxxxx ns  tttttt     (switch from translate mode)
::: vvvvvv LPSW  xxxxxxxx     tttttttt  tttttttt   (WAIT bit on)
::: vvvvvv LPSW  xxxxxxxx ==> tttttttt  tttttttt   (WAIT bit not on)
::: vvvvvv mnem  xxxxxxxx                          (all others)
```

Executed Instructions:

```
vvvvvv EX xxxxxxxx zz vvvvvv mnem xxxx xxxxxxxx
```

For an executed instruction, where zz (see preceding explanation of symbols) is nonzero, the mnemonic for the executed instruction is given as if the zz byte had been put into the instruction with an OR operation.

All Other Instructions:

```
vvvvvv mnem xxxxxxxx xxxx
```

SUCCESSFUL BRANCH:

```
vvvvvv mnem xxxxxxxx ==> tttttt
```

INTERRUPTION (SVC, PROGRAM, or EXTERNAL)

```
*** vvvvvv int code ==> tttttt
```

I/O INTERRUPTION (First line given only if "CSW" was specified):

```
CSW V vadd xxxxxxxx xxxxxxxx R radd yyyyyyyy yyyyyyyy
*** vvvvvv I/O vadd ==> tttttt CSW xxxx
```

BRANCH TRACE: (ALL option selected)

Entry for 'branch from' instruction

```
vvvvvv mnem xxxxxxxx     tttttt
```

Entry for 'branch to' instruction

```
==> vvvvvv mnem xxxxxxxxxxxx
```

CP COMMANDS FOR SYSTEM PROGRAMMERS AND SYSTEM ANALYSTS

CP real machine debugging is reserved for Class C users (system programmers) and Class E users (system analysts). CP has facilities to examine data in real storage (via the DCP and DMCP commands) and to store data into real storage (via the STCP command). There is no facility to examine or alter real machine registers, PSW, or storage words.

Remember, real storage is changing even as you issue the CP commands to examine and alter it.

System programmers and analysts may also want to use the CP internal trace table. This table records events that occur on the real machine.

DCP

Privilege Classes: C and E

Use the DCP command to display the contents of real storage locations at the terminal.

If an invalid operand is entered, the DCP command terminates. However, any previous valid operands are processed before termination occurs. The format of the DCP command is:

```
┌──────────────────────────────────────────────────────────────────────┐
│                                                                      │
│ DCP          │ ┌         ┐┌    ┌          ┐ ┐                         │
│              │ │Lhexloc1 ││{-}│ hexloc2  │ │                         │
│              │ │Thexloc1 ││{:}│ END      │ │                         │
│              │ │ hexloc1 ││   └          ┘ │                         │
│              │ │    0    ││                │                         │
│              │ └         ┘│                │                         │
│              │            │    ┌         ┐ │                         │
│              │            │{.}│bytecount│ │                         │
│              │            │    │END      │ │                         │
│              │            └    └         ┘ ┘                         │
└──────────────────────────────────────────────────────────────────────┘
```

where:

Lhexloc1
Thexloc1
hexloc1
0
: specifies the first storage location to be displayed. If hexloc1 is the only operand, it specifies the only storage location to be displayed. If hexloc1 is not specified, L or T must be specified and the display begins with storage location 0. If hexloc1 is specified and L or T is not specified, the display is in hexadecimal. T specifies that an EBCDIC translation is to be included with the hexadecimal display. L specifies that the display is to be in hexadecimal only. If hexloc1 is followed by a period and is not on a fullword boundary, it is rounded down to the next lower fullword.

{-|:} [hexloc2 | END]
: specifies that a range of locations is to be displayed. To display the contents of one or more storage locations by specified storage address location the "-" or ":" must be used. The hexloc2 operand must be 1- to 6-hexadecimal digits; leading zeroes need not be specified. In addition, The hexloc2 operand must be equal to hexloc1 and it should not exceed the size of real storage. If END is specified, real storage from hexloc1 through the end of real storage is displayed. If hexloc2 is not specified, END is the default. Note that this occurs only if "-" or ":" follows the first operand.

{.} [bytecount | END]
: is a hexadecimal integer designating the number of bytes of real storage (starting with the byte at hexloc1) to be displayed on the terminal. The sum of hexloc1 and the bytecount must be an address that does not exceed the size of real storage. If this address is not on a fullword boundary, it is rounded up to the next higher fullword. The bytecount operand must be a value of 1 or greater and may not exceed six hexadecimal digits.

Usage:

Normally, a user will or should define the beginning and ending locations of storage in the following manner:

```
dcp Lhexloc1-hexloc2
dcp Thexloc1-hexloc2
dcp hexloc1:hexloc2
dcp hexloc1.bytecount
dcp hexloc1:hexloc2 hexloc1.bytecount
```

Note that no blanks can be entered between the limit or range symbols (: or - r .) or any of the operands except for the blank or blanks between the command name and the first operand. A blank is also required between each set of operands when more than one set cf operands are entered on one command line.

However, if a blank immediately follows the designated type character (T or L), DCP displays all of real storage. If the next operand is either a colon (:), a hyphen (-), or a period (.) followed by a blank character, the system again defaults to a display of all storage locations as this operand assumes a second set of operands.

Note: Blanks separate operands or sets of operands if more than one operand is entered on the same command line. Blanks should not occur on the right or left of range or length symbols, unless it is intended to take the default value of the missing operand defined by the blank.

The following are examples of DCP entries that produce full storage displays.

```
dcp l          dcp 1-         dcp 0-         dcp t:end
dcp t          dcp 1:         dcp 0:         dcp t:end
dcp -          dcp t:         dcp 1-end      dcp 0:end
dcp :          dcp 1.         dcp t-end      dcp 1.end
dcp .          dcp t.         dcp 0-end      dcp 0.end
```

The following displays all of storage three times because of the embedded blanks:

```
dcp l . t
```

Response

Requested locations are displayed in the following format:

xxxxxx = word1 word2 word3 word4 [key] *EBCDIC translation*

where xxxxxx is the real storage location of word1. "word1" is displayed (word aligned) for a single hexadecimal specification. Up to four words are displayed on a line. If required, multiple lines are displayed. The EBCDIC translation is displayed aligned to the next lower 16-byte boundary if Thexloc is specified. Nonprintable characters display as a ".". If the location is at a 2K page boundary, the key for that page is also displayed. The output can be stopped and the command terminated by pressing the ATTN key (or its equivalent).

DMCP

Privilege Classes: C and E

Use the DMCP command to print the contents of real storage locations on the user's virtual spooled printer. The output format is eight words per line with EBCDIC translation. Multiple storage locations and ranges may be specified. To get the output printed on the real printer, the virtual spooled printer must be terminated with a CLOSE command. The format of the DMCP command is:

```
+-------------------------------------------------------------------------+
| DMCP     | ┌           ┐┌ ┌           ┐ ┐ [*dumpid]                      |
|          | |Lhexloc1   ||{-}| hexloc2 | |                                |
|          | |Thexloc1   ||{:}| END     | |                                |
|          | | hexloc1   ||   └         ┘ |                                |
|          | |    0      ||               |                                |
|          | └           ┘|               |                                |
|          |              |   ┌         ┐ |                                |
|          |              |{.}|bytecount| |                                |
|          |              |   |END      | |                                |
|          |              └   └         ┘ ┘                                |
+-------------------------------------------------------------------------+
```

where:

Lhexloc1
Thexloc1
hexloc1
0
: specifies the first storage location to be dumped. If hexloc1 is the only operand, it specifies the only storage location to be dumped. If hexloc1 is not specified, L or T must be specified and dumping starts with storage location 0. An EBCDIC translation is included with the dump contents. If hexloc1 is followed by a period and is not on a fullword boundary, it is rounded down to the next lower fullword.

{-/:} [hexloc2 | END]
: is a range of real storage locations to be dumped. To dump to the end of real storage, hexloc2 may be specified as END or not specified at all, in which case END is assumed by default.

{.} [bytecount | END]
: is a hexadecimal integer designating the number of bytes of real storage (starting with the byte at hexloc1) to be typed at the printer. The sum of hexloc1 and the bytecount must be an address that does not exceed the size of real storage. If this address is not on a fullword boundary, it is rounded up to the next higher fullword. If the "." is used for a range, hexloc2 is defined as the number of hexadecimal storage locations (in bytes) to be dumped starting at hexloc1. If hexloc2 is specified as a length in this way, it must have a value such that when added to hexloc1 it will not exceed the storage size.

***dumpid**
: is specified for identification purposes. If specified, it becomes the first line printed preceding the dump data. Up to 100 characters with or without blanks may be specified after the asterisk prefix. If dumpid is specified, hexloc2 or bytecount must be specified. The asterisk (*) is required to identify the dumpid.

Usage:

Normally, a user would define beginning and ending dump locations in the following manner:

```
dmcp Lhexloc-hexloc

   -- or --

dmcp hexloc.bytecount
```

Note that there are no blanks between length or range symbols (: or - or .) or between any of the operands except for the blank(s) between the command and the first operand. A blank is also required between each set of operands when more than one set of operands are entered. Note, only one period (.), colon (:), dash (-) or no delimiter may be used within each set of operands.

If, however, a blank immediately follows the designated type character, the default dump starting and ending locations are assumed to be the beginning and/or end of virtual storage. Similarly, if the range or length symbol separates the first character from a blank or END, all of real storage is dumped.

Note: Blanks separate operands or sets of operands if more than one operand is entered on the same command line. Blanks should not occur on the right or left of the range or length symbol, unless it is intended to take the default value of the missing operand defined by the blank. Thus, all of the following produce full storage dumps.

```
dmcp l      dmcp l-     dmcp t.      dmcp t-end
dmcp t      dmcp t-     dmcp 0-      dmcp 0:end
dmcp -      dmcp l:     dmcp 0:      dmcp l.end
dmcp :      dmcp t:     dmcp 0.      dmcp l.end
dmcp .      dmcp l.     dmcp l-end   dmcp 0.end
```

Each of the following produces three full dumps because of the embedded blanks:

```
dmcp l . t
dmcp - : .
```

Note: In cases where multiple storage ranges or limits are specified on one command line and the line contains errors, command execution successfully processes all correct operands to the encountered error. The encountered error and the remainder of the command line is rejected and an appropriate error message is displayed.

Responses

As the dump proceeds, the following message appears at the terminal indicating that the dump is continuing from the next 64K boundary:

```
DUMPING LOC hexloc
```

where "hexloc" is the segment (64K) address for the dump continuation, such as 020000, 030000, 040000.

If the user signals attention on the terminal while the above message is displayed, the dump ends.

COMMAND COMPLETE

indicates normal completion of the dump.

LOCATE

Privilege Classes: C and E

Use the LOCATE command to find the addresses of CP control blocks associated with a particular user, a user's virtual device, or a real system device. The control blocks and their use are described in the VM/370: Data Areas and Control Block Logic. The format of the LOCATE command is:

```
| LOCate    | {userid [vaddr]} |
|           | {raddr         } |
```

where:

userid
: is the user identification of the logged on user. The address of this user's virtual machine block (VMBLOK) is printed.

vaddr
: causes the virtual channel block (VCHBLOK), virtual control unit block (VCUBLOK), and virtual device block (VDEVBLOK) addresses associated with this virtual device address to be printed with the VMBLOK address.

raddr
: causes the real channel block (RCHBLOK), real control unit block (RCUBLOK), and the real device block (RDEVBLOK) addresses associated with this real device address to be printed.

Responses

LOCATE userid

VMBLOK = xxxxxx

LOCATE userid vaddr

```
VMBLOK    VCHBLOK   VCUBLOK   VDEVBLOK
xxxxxx    xxxxxx    xxxxxx    xxxxxx
```

LOCATE raddr

```
RCHBLOK   RCUBLOK   RDEVBLOK
xxxxxx    xxxxxx    xxxxxx
```

MONITOR

Privilege Classes: A or E

Use the MONITOR command to initiate or terminate the recording of events that occur in the real machine. This recording is always active after a VM/370 IPL (manual or automatic). The events that are recorded in the CP internal trace table are:

- External interruptions
- SVC interruptions
- Program interruptions
- Machine check interruptions
- I/O interruptions
- Free storage requests
- Release of free storage
- Entry into scheduler
- Queue drop
- Run user requests
- Start I/O
- Unstack I/O interruptions
- Storing a virtual CSW
- Test I/O
- Halt device
- Unstack IOBLOK or TRQBLOK
- NCP BTU (Network Control Program Basic Transmission Unit)

Use the trace table to determine the events that preceded a CP system failure. The format of the MONITOR command for tracing events in the real machine is:

```
| MONitor  | { STArt CPTRACE } |
|          | { STOP  CPTRACE } |
```

where:

START CPTRACE
: starts the tracing of events that occur on the real machine. The events are recorded on the CP internal trace table in chronological order. When the end of the table is reached, recording continues at the beginning of the table, overlaying data previously recorded.

STOP CPTRACE
: terminates the internal trace table event tracing. Event recording ceases but the pages of storage containing the CP internal trace table are not released. Tracing can be restarted at any time by issuing the MONITOR START CPTRACE command.

Response:

COMMAND COMPLETE

The MONITOR command was processed successfully.

QUERY

Privilege Classes: A, B, C, D, E, and F

Use the QUERY command to request system status and machine configuration information. (For 3704 or 3705 Communication Controllers and remote 3270 resources see the Class A and B NETWORK command.) Not all operands are available in every privilege class.

Operands available to the specified privilege classes are given below. The format of the Class A and E QUERY command is:

```
| Query    | { PAGing            }
|          | { PRIORity userid   }
|          | { SASsist           }
```

where:

PAGING — displays the current system paging activity.

PRIORITY userid — displays the current priority of the specified userid. This is established in the VM/370 directory but can be overridden by the SET PRIORITY nn command.

SASSIST — displays the current status of the Virtual Machine Assist feature for the VM/370 system.

Responses to the Class A and E Query Commands

QUERY PAGING

PAGING nn, SET mm, RATE nnn/SEC INTERVAL=xx:xx:xx

where:

nn — specifies the percentage of time the system was in page wait during this time interval.

mm — is the system paging activity index (threshold value). This value affects the paging rate and degree of multiprogramming that VM/370 tries to attain. The value mm is normally 16.

nnn/SEC — is the current CP paging rate in pages per second.

xx:xx:xx — is the time interval between the issuance of QUERY PAGING commands.

QUERY PRIORITY userid

userid PRIORITY = nn

nn is the the assigned priority of the specified user. The lower the value, the higher the priority.

QUERY SASSIST

SASSIST {ON | OFF}

ON or OFF is indicates that the Virtual Machine Assist feature is enabled or disabled from the system.

The format of the Class B QUERY command is:

```
Query    | {DAsd   [ACTive ] }
         | {TApes  |OFFline| }
         | {LINES1 |FREe   | }
         | {UR     |ATTach | }
         | {GRaf   |ALL    | }
         | {ALL    [       ] }
         |
         | DAsd volid
         | TDsk
         | STORage
         | raddr
         | SYStem raddr
         | DUMP
```

[1]Query LINES is not effective for 3704/3705 resources unless the 3704/3705 is operating in 2701, 2702, 2703 Emulation Program (EP) mode. For 3704/3705 Communications Controllers operating in Network Control Program (NCP) or Partitioned Emulator Program (PEP) mode use the NETWORK QUERY command.

where:

DASD	displays the real addresses of disk or drum devices.
TAPES	displays the real addresses of magnetic tape units.
LINES	displays the real addresses of communication lines.
UR	displays the real addresses of unit record devices (card reader, card punches, printers).
GRAF	displays the locally attached display devices.
ALL	(used as a first operand) displays all devices and the size of real storage.
DASD volid	displays the active or free status of the specified DASD volume.

TDSK	displays all the currently allocated temporary disk space (TDSK) from all available system owned volumes assigned to virtual machine users.
STORAGE	displays the size of real storage.
raddr	displays the status of the device at the specified address.
SYSTEM raddr	displays the userid, virtual address, and access mode of virtual disks which reside on the specified channel and control unit address raddr belonging to logged on users.
DUMP	displays at the operator's terminal the type of device and device address of the unit designated to receive abnormal termination dumps.
ACTIVE	displays the status of only the active devices within the group specified. This is the default. Active devices do not include devices that are "free" or "offline". An active device is one that is in use by a user or the system.
OFFLINE	displays only the devices in an "offline" status within the group specified. An offline device is one that is not available for access by any user or the system.
FREE	displays all the devices that are not currently in use by the system or a user on the system. Free devices do not include "offline" devices. A free device is one that is not in use by a user or the system.
ATTACH	displays all the devices that are dedicated to any user on the system. An attached device is also an active device.
ALL	(as the second operand) displays the status of all devices within the group specified. The status is typed in the order of "active", "free" and "offline" and is equivalent to the response from entering QUERY type ACTIVE QUERY type FREE QUERY type OFFLINE

Responses to the Class B QUERY Command

QUERY ALL

Produces the same results as if the following commands were issued:

```
QUERY STORAGE
QUERY UR
QUERY LINES
QUERY DASD
QUERY TAPES
QUERY GRAF
```

QUERY DASD

DASD raddr ATTACH TO userid vaddr

is displayed if the real device specified by raddr is attached to a user's (userid) virtual machine at virtual address vaddr.

DASD raddr CP SYSTEM volid nnn

is displayed if the real device designated by raddr is allocated to the system for use as user's minidisks. nnn is the number of active user's minidisks on the physical disk and volid is the volume serial number of the real disk.

DASD raddr CP OWNED volid nnn

is displayed if the real device designated by raddr is used by the system for paging and spooling activity. nnn is the number of active user's minidisks (if any) on the physical disk and volid is the volume serial number of the real disk.

QUERY TAPES

TAPE raddr CP SYSTEM

is displayed if the real tape device designated by raddr is attached to CP for its exclusive use.

TAPE raddr ATTACH TO userid vaddr

is displayed if the real tape device designated by raddr is attached to a user's (userid) virtual machine at virtual address vaddr.

QUERY UR

$$\left\{\begin{array}{l}\text{PRT}\\\text{PUN}\end{array}\right\}\ \text{raddr}\ \left\{\begin{array}{l}\text{STARTED}\\\text{DRAINED}\end{array}\right\}\ \text{SYSTEM CLASS = a...}\ \left\{\begin{array}{l}\text{SEP}\\\text{NOSEP}\end{array}\right\}$$

$$\text{RDR}\ \ \text{raddr}\ \left\{\begin{array}{l}\text{STARTED}\\\text{DRAINED}\end{array}\right\}\ \text{SYSTEM}$$

is displayed for each unit record device assigned to the system for spooling activity.

where:

raddr is the real device address (cuu).

DRAINED indicates that the device is not currently available for processing. A START command must be issued to activate the device.

STARTED indicates that the device is available for spooling activity.

a... specifies the classes serviced by the output device. Up to four classes may be serviced by an output device. No blanks or commas are allowed between classes.

NOSEP indicates the device was started with the NOSEP option.

SEP indicates the device was started without the NOSEP option.

<u>Note</u>: The separator (SEP) option applies to printer output where the edge of the fanfolded continuous forms are heavily printed. This indicates to the spooling operator the beginning and end of adjacent spool files.

$\left\{\begin{array}{l}\text{PRT}\\\text{PUN}\\\text{RDR}\end{array}\right\}$ raddr ATTACH TO userid vaddr

is displayed if the device is attached to a user's virtual machine at vaddr.

If the unit record device is currently active with a spool file, the following additional response is also given:

$\left\{\begin{array}{l}\text{PRT}\\\text{PUN}\end{array}\right\}$ raddr $\left\{\begin{array}{l}\text{PRINTING}\\\text{PUNCHING}\end{array}\right\}$ userid FILE = file RECDS = norecs COPY = nn a typ

RDR raddr READING userid FILE = file

<u>where</u>:

userid is the name of the spool file owner.

file is the spool file spoolid number.

norecs is the total file logical record count.

nn is the number of copies remaining for output, where 01 indicates the last copy.

a is the spool file class.

typ is the originating device type (PRT, PUN, CON).

<u>QUERY LINES</u>

$\left\{\begin{array}{l}\text{LINE}\\\text{CONS}\end{array}\right\}$ raddr LOGON AS userid

indicates that the user represented by userid is currently logged on at the terminal located at the address raddr.

LINE raddr ATTACH TO userid vaddr

indicates that the communication line at raddr is attached to the virtual machine represented by userid at virtual address vaddr.

QUERY GRAF

GRAF raddr LOGON AS userid

indicates that the user represented by userid is currently logged on at the terminal located at real address raddr.

GRAF raddr ATTACH TO userid vaddr

indicates that the display device at real address raddr is attached to the virtual machine represented by userid at the virtual address vaddr.

QUERY type OFFLINE

This command produces a response for each offline device in the following format:

type raddr OFFLINE

Multiple responses are displayed in the following format:

```
type raddr OFFLINE, ...
  .      .      .
  .      .      .
  .      .      .
```

__Note__: In the above responses the term type refers to one or more of the following device types:

Type	Meaning
DASD	Direct access device
TAPE	Magnetic tape units
LINE	Communication line
RDR	Card reader
PRT	Line printer
PUN	Card punch
GRAF	Graphics device
CONS	Console
CTCA	Channel to channel adapter
CTLR	3704/3705 communications controller
DEV	Any other device

QUERY type FREE

This command produces a response for each device that is not active or offline in the following format:

type raddr FREE

For unit record devices the response is:

```
type raddr DRAINED
```

Note: This response implies that no spool files are queued for this device.

For communication devices the response is:

```
            {ENABLED }
type raddr  {DISABLED}
```

For DASD devices with mounted volumes the response is:

```
type raddr {FREE }
           {volid}
```

Multiple responses are displayed in the following format:

```
type raddr FREE, ...
  .      .    .
  .      .    .
  .      .    .
```

QUERY DASD volid

The command response is given in either the "active" or "free" format depending upon the device status.

QUERY TDSK

This command displays all the currently allocated user TDSK space from all available system-owned volumes. One entry of the following format is produced for each TDSK:

```
userid vaddr nnn
```

where:

userid is the virtual machine identification.

vaddr is the user's virtual device address.

nnn is the number of cylinders allocated.

Note: If the operator does a QUERY to any real device or group of devices (such as QUERY DASD) the following message occurs for all devices in a not-ready status and the CPU alarm rung:

```
type raddr INT REQ
```

QUERY STORAGE

STORAGE = xxxxxK

displays the size of real storage (xxxxx) in multiples of 1024 bytes.

QUERY raddr

The response to this command depends upon the type of device located at raddr.

See the QUERY DASD, TAPES, UR, GRAF, and LINES responses.

QUERY SYSTEM raddr

This command requests the number of user minidisks residing on the physical disk located at raddr. The response for each minidisk is given in the following format:

```
userid vaddr mode, ...
   .      .     .
   .      .     .
   .      .     .
```

where:

userid is the identification of the user who owns the minidisk.

vaddr is the virtual address by which the user refers to the minidisk.

mode is the type of access the user has: either R/O or R/W, or nnn for the number of cylinders of TDSK space allocated.

QUERY DUMP

type raddr DUMP UNIT $\left\{\begin{matrix}\text{CP}\\\text{ALL}\end{matrix}\right\}$

indicates that the device of device type "type" located at raddr is the system dump unit.

The format of the Class D QUERY command is:

```
┌──────────────────────────────────────────────────────────────┐
│           │ ⎧                   ┌      ┐  ⎫                  │
│  Query    │ ⎪  Files [CLass c]  │userid│  ⎪                  │
│           │ ⎪                   │  *   │  ⎪                  │
│           │ ⎪                   └      ┘  ⎪                  │
│           │ ⎪                             ⎪                  │
│           │ ⎨  Reader  ┌┌        ┐          ┐⎬                  │
│           │ ⎪  Printer ││ALL     │ [userid]│⎪                  │
│           │ ⎪  PUnch   ││CLass a│          │⎪                  │
│           │ ⎪          │└        ┘          │⎪                  │
│           │ ⎪          │  spoolid           │⎪                  │
│           │ ⎪          └                    ┘⎪                  │
│           │ ⎩  HOld                         ⎭                  │
└──────────────────────────────────────────────────────────────┘
```

where:

FILES — displays the number of spooled input and output files. The Class D user receives the total count in the system. Files that are currently being processed are not included in the totals.

CLASS c — displays only the spool files of the specified class. If CLASS is omitted then all spool classes are examined.

userid — displays only the spool files owned by the specified userid. If userid is omitted then spool files owned by all users are examined.

* — displays only the spool files of the logon user who issued the QUERY command.

READER
RDR — displays basic information concerning reader spool files.

PRINTER
PRT — displays basic information concerning printer spool files.

PUNCH
PCH — displays basic information concerning punch spool files.

Note: The basic information displayed is:

- Userid of the owner of the spool file. If examining files for a specific user (userid option), the userid indicates the originator of the spool file.
- Spool file spoolid number
- Class and originating device type
- Number of logical records in the file
- Number of copies specified for the file
- File hold status

HOLD — displays a list of users whose output is being held by the HOLD command.

ALL — displays additional information for all spool files examined.

spoolid displays additional information for the specified spool file. The spool identification (spoolid) is a VM/370 generated sequential number assigned to each spool file.

The additional information displayed is:

- Date and time the file was created
- Filename and filetype of the file (if any)
- Distribution code of the file

Responses to the Class D QUERY Command

QUERY FILES [CLass c] [userid]

```
FILES: {NO } RDR, { NO } PRT, {NO } PUN
       {nnn}      { nnn}       {nnn}
```

displays the total number of spool files in the system, of a particular class, or for a particular userid.

```
                  ┌                              ┐
                  │  ┌         ┐                 │
       {READER }  │  │ALL      │                 │
QUERY  {PRINTER}  │  │Class a  │ [userid]        │
       {PUNCH  }  │  └         ┘                 │
                  │   spoolid                    │
                  └                              ┘
```

```
┌─────────Basic Information─────────┐ ┌─────Additional Information─────┐
v                                   v v                                v
                                      ┌                                ┐
OWNERID[1] FILE CLASS RECDS  CPY HOLD │DATE  TIME     NAME  TYPE  DIST │
userid     file a typ norecs nn  stat │mm/dd hh:mm:ss name  type  code │
                                      └                                ┘
   .       .    . .   .      .   .     .     .         .     .     .
   .       .    . .   .      .   .     .     .         .     .     .
   .       .    . .   .      .   .     .     .         .     .     .
```

Only one file is listed for a QUERY READER, QUERY PRINTER, or QUERY PUNCH command if the spoolid operand is specified.

The DATE, TIME, NAME, TYPE, and DIST information is displayed only when the following commands are issued:

```
QUERY {READER } {ALL    }
      {PRINTER} {spoolid}
      {PUNCH  }
```

where:

userid is the identification of the user who owns the file.

file is a unique, system assigned number which is used by VM/370 to identify the file.

a is the spool file class.

---

[1]OWNERID heading the title line for the spool file data is altered to ORIGINID when the userid operand is used. In that event, ORIGINID represents the originator of the file.

typ — is the originating device type (PRT, PUN, CON, or RDR).

norecs — is the number of logical records contained in the file.

nn — is the number of copies specified for the file. (Has no effect for reader files.)

stat — is the file hold status and is either

NONE - no hold
USER - user hold

mm/dd — is the date the file was created in month/day.

hh:mm:ss — is the actual time of the creation of the file in hours:minutes:seconds.

filename — is the filename assigned to the file (if any). If the file has a 24-character data set name (dsname), only 20 characters are displayed. These characters extend from the "name" field through the "type" field.

filetype — is the filetype assigned to the file (if any).

distcode — is the distribution code of the file.

QUERY HOLD

```
HOLD : { NO  } RDR, { NO  } PRT, { NO  } PUN
       { nnn }      { nnn }      { nnn }

         ( ALL )
userid - ( RDR ) , ...
         ( PRT )
         ( PUN )
```

The first response displays the total number of files within the system which are retained in the SYSTEM HOLD status. The second response indicates the type of hold (if any) for any user in the system for which HOLD is in effect. The user who issues QUERY HOLD may receive, depending upon the status of his spooled files, the first response, the second response, or both responses.

The format of the Class A, B, C, D, E, F, and G QUERY command is:

```
| Query      | (LOGmsg              )
|            | )Names               (
|            | )Users    [userid]   (
|            | (userid              )
```

where:

LOGMSG displays the log messages of the day.

NAMES displays a list of all the users logged on and the real address of the line to which each is connected. If the user is disconnected, DSC is displayed instead of the line address.

USERS displays the number of logged on users and the number of users dialed to other virtual machines. If userid is specified, the userid and device address of the user's terminal are displayed if he is logged on. If the specified user is not logged on, a message to that effect occurs. Use the USERS operand if the userid is the same as an operand (or its minimum truncation) of the QUERY command.

Note: It is possible for the number of users logged on as indicated by the 'NAMES' operand to differ from the number logged on as indicated by the 'USERS' operand. The number of users in the process of logging on and logging off accounts for this difference.

userid displays the userid and the device address of the user's terminal if he is logged on. If the user is not logged on, a message to this effect occurs.

Responses to the A, B, C, D, E, F, and G QUERY Command

QUERY LOGMSG

```
* logmsg text line 1
  .             .
  .             .
  .             .
* logmsg text line n
  logmsg additional text lines
  .                          .
  .                          .
  .                          .
```

All lines (both those with and without an asterisk) in the log message file are displayed.

QUERY NAMES

```
userid - {DSC  }, ...
.        {raddr}
.
.
userid - {DSC  }, ...
         {raddr}
```

Lists all logged on users. If the user is currently connected, the real address to which he is connected is displayed (raddr). If he is not connected to the system, DSC is displayed.

QUERY USERS

nnn USERS, mmm DIALED

nnn is the total number of logged on users.

mmm is the total number of users logically attached via the DIAL command to virtual machines.

Note: The term DIALED means that the line is not available to CP because it is logically attached to a logged-on user and is a part of that user's virtual machine operation.

QUERY userid

userid - raddr

displays the real address (raddr) to which the specified user is connected.

STCP

Privilege Class: C

Use the STCP command to alter the contents of real storage. The real PSW or real registers cannot be altered with this command. The format of the STCP command is:

```
| STCP     | { { hexloc }  hexword1 [hexword2...] } |
|          | { { Lhexloc}                         } |
|          | {                                    } |
|          | { Shexloc   hexdata                  } |
```

where:

hexloc
Lhexloc
: stores the data given in hexword1 [hexword2...] in successive fullword locations starting at the address specified by hexloc. The smallest group of hexadecimal values that can be stored using this specification is one fullword. Data is aligned to the nearest fullword boundary. If the data being stored is less than a fullword (8-hexadecimal digits), it is right-adjusted in the word and the high order bytes of the word are filled with zeros. Either specification (hexloc or Lhexloc) may be used.

Shexloc
: stores the data given in hexdata in the address specified by hexloc without word alignment. The shortest string that can be stored is one byte (2-hexadecimal digits). If the string contains an odd number of characters, the last character is not stored. An error message occurs and the function ends.

hexword
: specifies up to 8-hexadecimal digits. If less than eight digits are specified, the string is right justified in a fullword and left-filled with zeros. If two or more hexwords are specified, they must be separated by at least one blank.

hexdata
: specifies a string of two or more hexadecimal digits with no embedded blanks.

Response

STORE COMPLETE

DASD DUMP RESTORE (DDR) SERVICE PROGRAM AND HOW TO USE IT

Use the DASD Dump Restore (DDR) program to dump, restore, copy, or print VM/370 user minidisks. The DDR program may run as a standalone program, or under CMS via the DDR command.

The DDR program has five functions:

1. Dumps part or all of the data from a DASD device to tape.
2. Transfers data from tapes created by the DDR dump function to a direct access device. The direct access device must be the same as that which originally contained the data.
3. Copies data from one device to another of the same type. Data may be reordered, by cylinder, when copied from disk to disk. In order to copy one tape to another, the original tape must have been created by the DDR DUMP function.
4. Prints selected parts of DASD and tape records in hexadecimal and EBCDIC on the virtual printer.
5. Displays selected parts of DASD and tape records in hexadecimal and EBCDIC on the terminal.

To generate the VM/370 starter system from the distribution tape, the standalone RESTORE function must be used.

INVOKING DDR UNDER CMS

The format of the DDR command is:

```
┌──────────────────────────────────────────────────────────────┐
│             │            ┌  ┐                                │
│   DDR       │ [fn   ft   │fm│  ]                             │
│             │            │* │                                │
│             │            └  ┘                                │
└──────────────────────────────────────────────────────────────┘
```

where:

fn ft [fm | *]	is the identification of the file containing the control statements for the DDR program. If no file identification is provided, the DDR program attempts to obtain control statements from the console. The filemode defaults to * if a value is not provided.

Note: If you use the CMS DDR command, CMS ignores the SYSPRINT control statement and directs the output to the CMS printer 00E.

INVOKING DDR AS A STANDALONE PROGRAM

To use DDR as a standalone program, the operator should IPL it from a real or virtual IPL device as he would any other standalone program. Then indicate where the DDR program is to obtain its control statements by responding to prompting messages at the console.

Note: Be aware that DDR when run as a standalone program does not have error recovery support. However, when DDR is invoked in CMS, in a virtual machine environment, the I/O operation is performed by CP (CP has built-in error recovery facilities).

DDR CONTROL STATEMENTS

DDR control statements describe the intended processing and the needed I/O devices. I/O definition statements must be specified first.

All control statements may be entered from either the console or the card reader. Only columns 1 to 71 are inspected by the program. All data after the last operand in a statement is ignored. An output tape must have the DASD cylinder header records in ascending sequences; therefore, the extents must be entered in sequence by cylinder. Only one type of function -- dump, restore, or copy -- may be performed in one execution, but up to 20 statements describing cylinder extents may be entered. The function statements are delimited by an INPUT or OUTPUT statement, or by a null line if the console is used for input. If additional functions are to be performed, the sequence of control cards must be repeated. If you do not use INPUT or OUTPUT control statements to separate the functions you specify when the input is read from a card reader or CMS file, an error message (DMKDDR702E) is displayed. However, the remainder of the input stream will be checked for proper syntax, but no further DDR operations will be performed. Only those statements needed to redefine the I/O devices are necessary for subsequent steps. All other I/O definitions remain the same.

To return to CMS, enter a null line (carriage return) in response to the prompting message (ENTER:). To return directly to CP, key in #CP.

The PRINT and TYPE statements work differently from other DDR control statements in that they operate on only one data extent at a time. If the input is from a tape created by the dump function, it must be positioned at the header record for each step. The PRINT and TYPE statements have an implied output of either the console (TYPE) or system printer (PRINT). Therefore, PRINT and TYPE statements need not be delimited by an INPUT or OUTPUT statement.

I/O DEFINITION STATEMENTS

The I/O definition statements describe the tape, DASD, and printer devices used while executing the DASD Dump Restore program.

INPUT/OUTPUT Control Statement

An INPUT or OUTPUT statement describes each tape and DASD unit used. The format of the INPUT/OUTPUT statement is:

```
┌──────────────────────────────────────────────────────────────────────┐
│              │                                                       │
│  INput       │  cuu  type  ┌volser┐  [(options...)]                  │
│  OUTput      │             │altape│                                  │
│              │             └      ┘                                  │
│              │       Options:                                        │
│              │       ┌          ┐ ┌            ┐ ┌       ┐            │
│              │       │SKip  nn  │ │MOde  6250  │ │REWind │            │
│              │       │SKip  0   │ │MOde  1600  │ │UNload │            │
│              │       └          ┘ │MOde   800  │ │LEave  │            │
│              │                    └            ┘ └       ┘            │
└──────────────────────────────────────────────────────────────────────┘
```

where:

INPUT indicates that the device described is an input device.

OUTPUT indicates that the device described is an output device.

cuu is the unit address of the device.

type is the device type (2314, 2319, 3330, 3330-11, 3340-35, 3340-70, 3350, 2305-1, 2305-2, 2400, 2420, or 3420) (no 7-track support for any tape devices). Specify a 3410 device as a 3420, a 3340-70F as a 3340-70, and a 3333 as a 3330. Specify a 3350 that is in 3330-1 or 3330-11 compatibility mode as a 3330 or 3330-11. Specify a 3344 as a 3340-70, and specify 3350 for a 3350 operating in native mode (as opposed to compatibility mode).

Note: The DASD Dump Restore (DDR) program, executing in a virtual machine, uses I/O DIAGNOSE 20 to perform I/O operations on tape and DASD devices. DDR under CMS requires that the device type entered agree with the device type of the real device as recognized by VM/370. If there is a conflict with device types, the following message is issued:

DMKDDR708E INVALID OPTION

However, if DDR executes standalone in a virtual machine, DDR uses DIAGNOSE 20 to perform the I/O operation if the device types agree. If the device types do not agree, error message DMKDDR708E is issued.

volser is the volume serial number of a DASD device. If the keyword "SCRATCH" is specified instead of the volume serial number, no label verification is performed.

altape is the address of an alternate tape drive.

Note: If multiple reels of tape are required and "altape" is not specified, DDR types the following at the end of the reel:

    END OF VOLUME CYL xxx HD xxx, MOUNT NEXT TAPE

After the new tape is mounted, DDR continues automatically.

Options:

SKIP nn
     0 — forward spaces nn files on the tape. nn is any number up to 255. The SKIP option is reset to zero after the tape has been positioned.

MODE {6250 | 1600 | 800} — causes all output tapes that are opened for the first time and at the load point to be written or read in the specified density. All subsequent tapes mounted are also set to the specified density. If no mode option is specified, then no mode set is performed and the density setting remains as it previously was.

REWIND — rewinds the tape at the end of a function.

UNLOAD — rewinds and unloads the tape at the end of a function.

LEAVE — leaves the tape positioned at the end of the file at the end of a function.

Note: When the wrong input tape is mounted, the message DMKDDR709E is displayed and the tape will rewind and unload regardless of options REWIND, UNLOAD, or LEAVE being specified.

SYSPRINT Control Statement

Use the SYSPRINT control statement (in the standalone DDR virtual machine only) to describe the printer that is to print data extents specified by the PRINT statement. It also can print a map of the cylinder extents from the DUMP, RESTORE, or COPY statement. If the SYSPRINT statement is not provided, the printer assignment defaults to 00E. CMS ignores the SYSPRINT statement when you invoke DDR as a command under CMS, and CMS always directs the output to 00E. The format of the SYSPRINT control statement is:

SYsprint	cuu

where:

cuu — specifies the unit address of the device.

Function Statements

The function statements tell the DDR program what action to perform. The function commands also describe the extents to be dumped, copied, or restored. The format of the DUMP/COPY/RESTORE control statement is:

```
┌──────────────────────────────────────────────────────────────────────┐
│              │ ┌                                           ┐          │
│   DUmp       │ │cyl1 [To]  [cyl2 [Reorder] [To] [cyl3]]    │          │
│   COpy       │ │CPvol                                      │          │
│   REstore    │ │ALL                                        │          │
│              │ │NUcleus                                    │          │
│              │ └                                           ┘          │
└──────────────────────────────────────────────────────────────────────┘
```

where:

DUMP requests the program to move data from a direct access volume onto a magnetic tape or tapes. The data is moved cylinder by cylinder. Any number of cylinders may be moved. The format of the resulting tape is:

Record 1: a volume header record, consisting of data describing the volumes.

Record 2: a track header record, consisting of a list of count fields to restore the track, and the number of data records written on tape. After the last count field the record contains key and data records to fill the 4K buffer.

Record 3: track data records, consisting of key and data records packed into 4K blocks, with the last record truncated.

Record 4: either the end-of-volume (EOV) or end-of-job (EOJ) trailer label. The end-of-volume label contains the same information as the next volume header record, except that the ID field contains EOV. The end-of-job trailer label contains the same information as record 1 except that the cylinder number field contains the disk address of the last record on tape and the ID field contains EOJ.

COPY requests the program to copy data from one device to another device of the same or equivalent type. Data may be recorded on a cylinder basis from input device to output device. A tape-to-tape copy can be accomplished only with data dumped by this program.

RESTORE requests the program to return data that has been dumped by this program. Data can be restored only to a DASD volume of the same or equivalent device type from which it was dumped. It is possible to dump from a real disk and restore to a minidisk as long as the device types are the same.

cyl1 [TO] [cyl2 [REORDER] [TO] [cyl3]]
Only those cylinders specified are moved, starting with the first track of the first cylinder (cyl1), and ending with the last track of the second cylinder (cyl2). The REORDER operand causes the output to be reordered, that is, moved to different cylinders, starting at the specified cylinder (cyl3) or at the starting cylinder (cyl1) if "cyl3" is not specified. The REORDER operand must not be specified unless specified limits are defined for the operation; the starting and, if required, ending cylinders (cyl1 and cyl2) must be specified.

CPVOL specifies that cylinder 0 and all active directory and permanent disk space are to be copied, dumped, or restored. This indicates that both source and target disk must be in CP format, that is, the CP Format/Allocate program must have formatted them.

ALL specifies that the operation is to be performed on all cylinders.

NUCLEUS specifies that record 2 on cylinder 0, track 0 and the nucleus cylinders are dumped, copied, or restored.

Restrictions:

- Each track must contain a valid home address, containing the real cylinder and track location.

- Record zero must not contain more than eight key and/or data characters.

- Flagged tracks are treated just as any other track for all 2314, 2319, 3340, and 2305 devices. That is, no attempt is made to substitute the alternate track data when a defective primary track is read. In addition, tracks are not inspected to determine whether they were previously flagged when written. Therefore, volumes containing flagged tracks should be restored to the same cylinders of the volume from which they were dumped. The message DMKDDR715E occurs each time a defective track is dumped, copied or restored, and the operation continues.

- Flagged tracks for 3330 and 3350 devices are handled automatically by the control unit and may never be detected by the program. The program may detect a flagged track if, for example, no alternate track is assigned to the defective primary track. If a flagged track is detected by the program, the message DMKDDR715E occurs and the operation terminates.

Example:

```
INPUT 191 3330 SYSRES
OUTPUT 180 2400 181 (MODE 800
SYSPRINT 00F
DUMP CPVOL
INPUT 130 3330 MINI01
DUMP 1 TO 50 REORDER 51
60 70 101
```

This example sets the density to 800 bpi, then dumps all pertinent data from the volume labeled SYSRES onto the tape that is mounted on unit 180. If the program runs out of space on the first tape, it continues dumping onto the alternate device (181). A map of the dumped cylinders is printed on unit 00F while the program is dumping. When the first function is complete, the volume labeled MINI01 is dumped onto a new tape. Its cylinder header records are labeled 51 to 100. A map of the dumped cylinders is printed on unit 00F. Next, cylinders 60 to 70 are dumped and labeled 101 to 111. This extent is added to the cylinder map on unit 00F. When the DDR processing is complete, the tapes are unloaded and the program stops.

If cylinder extents are being defined from the console, the following is displayed:

```
ENTER CYLINDER EXTENTS
ENTER:
```

For any extent after the first extent, the message

```
ENTER NEXT EXTENT OR NULL LINE
ENTER:
```

is displayed.

The user may then enter additional extents to be dumped, restored, or copied. A null line causes the job step to start.

Note: When a cylinder map is printed on the virtual printer (00F as in the previous example) a heading precedes the map information. Module DMKDDR controls the disk, time and zone printed in the heading. Your installation must apply a local modification to DMKDDR to insure that local time, rather than GMT (Greenwich Meridian Time), is printed in the heading.

PRINT/TYPE Function Statement

Use the PRINT and TYPE function statement to print or type (display) a hexadecimal and EBCDIC translation of each record specified. The input device must be defined as direct access or tape. The output is directed to the system console for the TYPE function, or to the SYSPRINT device for the PRINT function. (This does not cause redefinition of the output unit definition.) The format of the PRINT/TYPE control statement is:

```
PRint  | cyl1 [hh1 [rr1]] [To cyl2 [hh2 [rr2 ]]] [(options...[) ]]
TYpe   |
       |        options:
       |        [Hex] [Graphic] [Count]
```

where:

cyl1 is the starting cylinder.

hh1 is the starting track. If present, it must follow the cyl1 operand. The default is track zero.

rr1 is the starting record. If present, it must follow the hh1 operand. The default is home address and record zero.

TO cyl2 is the ending cylinder. If more than one cylinder is to be printed or typed "TO cyl2" must be specified.

hh2 is the ending track. If present, it must follow the cyl2 operand. The default is the last track on the ending cylinder.

rr2 is the record ID of the last record to print. The default is the last record on the ending track.

Options:

HEX prints or displays a hexadecimal representation of each record specified.

GRAPHIC prints or displays an EBCDIC translation of each record specified.

COUNT prints or displays only the count field for each record specified.

Examples:

PRINT 0 TO 3

Prints all of the records from cylinders 0, 1, 2, and 3.

PRINT 0 1 3

Prints only one record, from cylinder 0, track 1, record 3.

PRINT 1 10 3 TO 1 15 4

Prints all records starting with cylinder 1, track 10, record 3, and ending with cylinder 1, track 15, record 4.

The example in Figure 63 shows the information displayed at the console (TYPE function) or system printer (PRINT function) by the DDR program. The listing is annotated to describe some of the data fields.

Responses

DMKDDR725R ORIGINAL INPUT DEVICE WAS(IS) LARGER THAN OUTPUT DEVICE. DO YOU WISH TO CONTINUE? RESPOND YES OR NO:

Explanation:
RESTORE function - The number of cylinders on the original DASD input unit is compared with the number of cylinders on the output device.

COPY function - The input device contains more cylinders than the output device.

Operator Action: The operator must determine if the COPY or RESTORE function is to continue. The response is either yes or no.

DMKDDR711R VOLID READ IS volid2 [NOT volid1] DO YOU WISH TO CONTINUE? RESPOND YES NO OR REREAD:

Explanation:

volid1 - The volume serial number from the input or output control statement; volid1 is displayed only if it was entered.

volid2 - is the volume serial number from the VOL1 label on the DASD device specified by the control statement.

System Action: The system waits for a response.

If you respond "yes", the operation will continue.

If you respond "no", and the input is from cards or a CMS file, the program is terminated after scanning the remaining statements for syntax. Otherwise, the next statement is solicited from the console.

If you respond "reread", the volume specified will be read again.

Note: A new volume may have been mounted in the interim.

User Action: Respond "yes", "no", or "reread."

DMKDDR716R NO VOL1 LABEL FOUND FOR volser
DO YOU WISH TO CONTINUE? RESPOND YES NO OR REREAD:

Explanation: The program was unable to find a record with the key of VOL1 on cylinder 0 track 0 and was not able to read record 3 on cylinder 0 track 0 for the specified volume serial number (volid). The volume serial number is displayed only if specified in the INPUT or OUTPUT control statement.

System Action: The system waits for a response.

If you respond "yes", the system will continue with the job steps.

If you respond "no" and the input is from cards or a CMS file, the program will be terminated after scanning the remaining statements for syntax. Otherwise, the next statement will be solicited from the console.

If you respond "reread", the program will attempt to reread the specified device.

User Action: Respond to the message as indicated.

DMKDDR717R DATA DUMPED FROM volid1 TO BE RESTORED to volid2
DO YOU WISH TO CONTINUE? RESPOND YES NO OR REREAD:

Explanation:
volid1 - The volume serial number of the input tape.

volid2 -- The volume serial number of the output DASD device that is to receive the data from volid1.

System Action: The system waits for a response.

If you respond "yes". the restore function will continue.

If you respond "no" and the input is from cards or a CMS file, the program will be terminated after scanning the remaining statement for syntax. Otherwise, the correct statement will be solicited from the console.

If you respond "reread", the input tape will be backspaced to the start of the file, and the volume header label will be reread.

User Action: If the wrong input tape is mounted, replace the tape and respond REREAD. Otherwise, respond in the appropriate manner.

ENTER CYLINDER EXTENTS
ENTER:

This message is received only if you are entering input from your terminal.

END OF VOLUME CYL xxx HD xx, MOUNT NEXT TAPE

DDR continues processing, after the mounting of the next tape reel.

RESTORING volser

where:

volser is the volume serial number of the disk dumped. The RESTORE operation has begun.

COPYING volser

where:

volser is the volume serial number described by the input unit. The COPY operation has begun.

DUMPING volser

where:

volser is the volume serial number described by the input unit. The dumping operation has begun.

PRINTING volser

where:

volser is the volume serial number described by the input unit. The PRINT opration has begun.

END OF DUMP

The DUMP operation has ended.

END OF RESTORE

The RESTORE operation has ended.

END OF COPY

The COPY operation has ended.

END OF PRINT

The PRINT operation has ended.

END OF JOB

All specified operations have completed.

ENTER:

Prompts input from the terminal. A null line (Press the Enter key or equivalent) causes control to return to CMS, if the virtual machine is in the CMS environment.

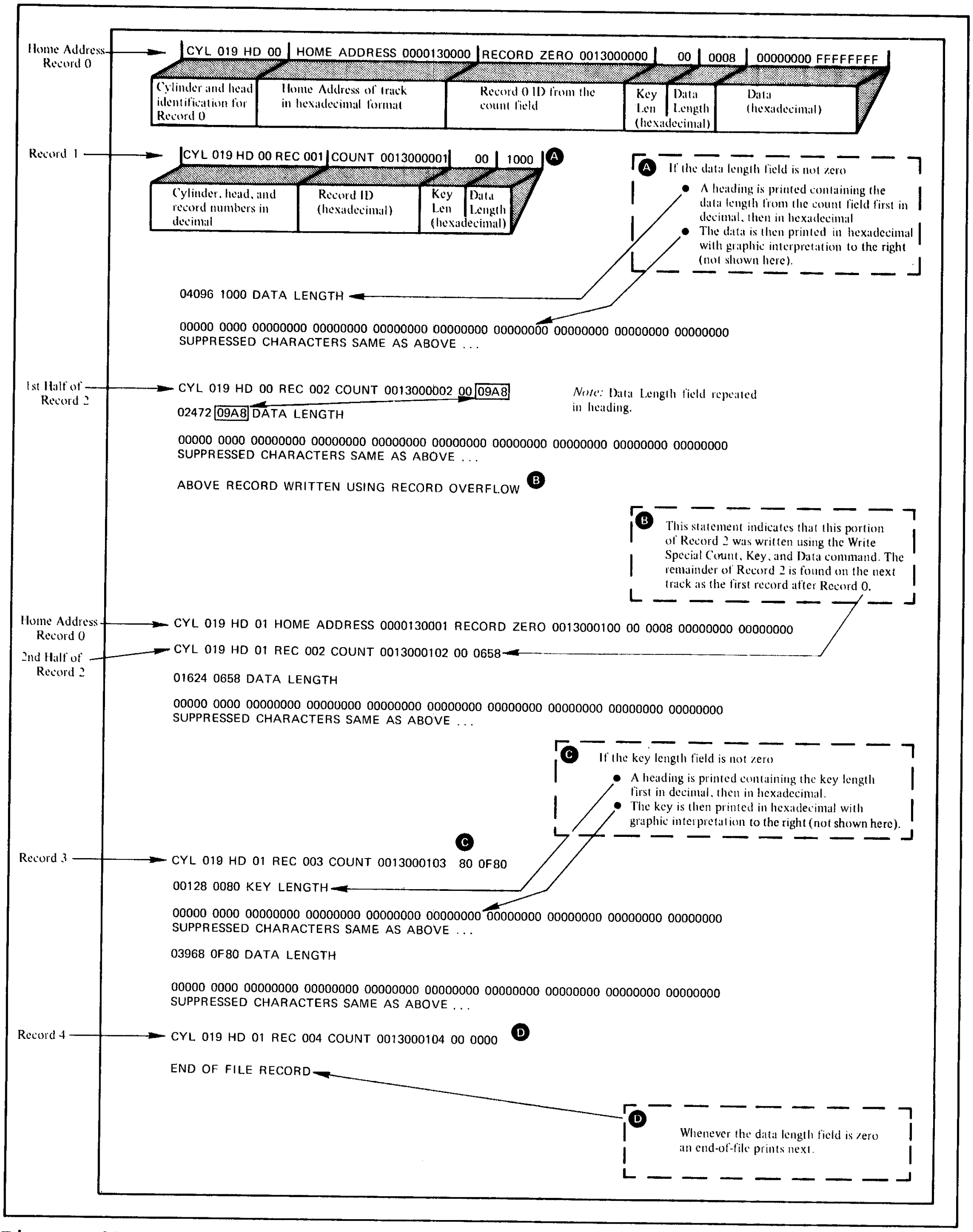


Figure 63. Annotated Sample of Output from the TYPE and PRINT Functions of the DDR Program

CP WAIT STATE CODES

A wait state is produced by one of the following modules:

DMKCCH	DMKMCH
DMKCKP	DMKPAG
DMKCPI	DMKSAV
DMKDMP	DMKWRM

When a wait state occurs, the Program Status Word (PSW) is displayed at the operator's console in the following format:

xxyyyyyyzzzzzwww

where:

xxyyyyyy is the left half of the program status word. This half may be either:

03yyyyyy Valid wait condition. The system is waiting for work.

00yyyyyy System wait caused by an error condition.

zzzzzwww is the right half of the program status word. The wait state code is found in the right half of the PSW when the CPU is in the wait state. The wait state code, www, indicates the error condition.

Wait Code	Explanation
001	The machine check handler found an unrecoverable failure. Probable hardware error.
002	The channel check handler found an unrecoverable failure. Probable hardware error.
003	A system failure occurred before a valid warm start was performed.
004	This wait state code is loaded by DMKDMP when a console, or an output device is not operational, or when a console or output device produces an inexplicable error status. Probable hardware error.
005	DMKCPI could not find an operational primary or alternate console. Probable hardware error.
006	This is a normal wait when a system shutdown is completed.
007	A program check, a machine check, or a permanent I/O error was found by the checkpoint program.
008	Checkpoint and system shutdown are complete. If the system is running under an alternate console, error messages DMKCKP910I, DMKCKP911W, DMKCKP960I, and DMKCKP961I are not displayed.
009	An error condition occurred that prevents a warm start. If the system is running under an alternate console, error messages DMKCKP910I and DMKCKP911W are not displayed.

00A A machine check occurred while DMKSAV was attempting to save or restore a page image copy of the nucleus on a SYSRES device. Probable hardware error.

00B A machine check occurred before initialization was complete.

00C An attempt was made to IPL from a disk that did not contain a system. Thus, the wait state code 00C entered on disk by the Format/Allocate program is encountered.

00D The machine size defined during system generation is greater than the real machine size, or a hardware error has occurred which inhibits VM/370 from using the required storage.

00F Hardware errors are being received on VM/370 paging device(s). The wait state that causes this code is preceded by message

DMKPAG415E CONTINUOUS PAGING ERRORS FROM DASDxxx

010 The SYSRES device, on which DMKSAV is attempting to write a page image copy of the nucleus, is not mounted or not ready.

011 An unrecoverable error, other than a machine check, occurred while DMKSAV attempted to write a page image copy of the nucleus on the SYSRES device.

012 The normal wait state code loaded by DMKSAV when it has completed loading the nucleus.

CP ABEND CODES

Figure 64 lists the CP ABEND, their cause and required action.

ABEND Code	Reason for ABEND	Action
BLD001	Register 8 should contain a pointer to the RDEVBLOK for the user's terminal. This routine (DMKBLDVM) attempts to create and partially initialize a VMBLOK for a user. DMKBLDVM abnormally terminates if general register 8 does not contain a pointer to the user.	Verify that general register 8 points to an RDEVBLOK for a terminal. If it does not, there is probably an error in the calling program. Identify the calling program by means of the return address and the base register in the save area pointed to by general register 13. Then, attempt to identify the source of the incorrect RDEVBLOK address.
BLD002	Pages are being released but the page invalid bit is not on in the page table entry.	Examine the dump and determine why the page was released without the page invalid bit turned on.
CFG010	DMKCFGCL was called to perform an unsupported function. The function request may be found in SAVEWORK1, byte 2. Supported values are: X'01' LOAD SYS X'02' FIND SYS X'04' PURGE SYS	Identify the caller by the return address and base register in the SAVEAREA pointed to by register 13 to identify the source of the unsupported function request.
CKS001	The map for dynamic checkpoint has not been allocated prior to a call to DMKCKSPL.	The map should be allocated via a call to entry points DMKCKSIN or DMKCKSWM from DMKWRM. Check that DMKWRM does, in fact, call one of these entry points and that they do allocate a map.
CKS002	The spool file identification in the map and on the checkpoint cylinder do not match.	In this case, (1) DMKCKSWM or DMKCKSIN did not set up the map properly, (2) a call to DMKCKSPL caused the mismatch, or (3) the SFBLOK was released but the map was not updated.
CKS003	No function was specified in the call to DMKCKSPL.	Check location SAVERTN in the save area pointed to by general register 13. This indicates which routine called DMKCKSPL with insufficient data.
CKS004	A spool file to be deleted cannot be found on the system printer, punch, or reader file chains.	The SFBLOK for the file should have been queued previously on either the printer, punch, or reader file chain by DKMCKSWM when performing a CKPT start. Check for an error in this logic.

Figure 64. CP ABEND Codes (Part 1 of 15)

ABEND Code	Reason for ABEND	Action
CPI001	The RDEVBLOK for the DASD on which the SYSRES volume is mounted cannot be located, or the IPL volume is not the SYSRES volume. The SYSRES volume is specified in the SYSRES macro in the DMKSYS module.	Verify that the volume serial number on the SYSRES volume from which the IPL was attempted, is the same as that specified in the field DMKSYSVL. If the volume serial number is not the same, it may have been altered by the CLIP utility. Or, the image of the same nucleus saved on the SYSRES may have been partially destroyed and the SYSRES specification altered. Load or restore the nucleus from a backup copy to the SYSRES volume and try to IPL again.
CPI002	A valid system directory file could not be located.	Display the volume labels for all owned volumes. If the volumes do not contain an active directory pointer, run DMKDIR (the stand-alone directory program) to recreate the system directory on an owned volume. If an active directory pointer is present in at least one volume label, verify that the device on which the volume is mounted is online and ready before trying to IPL the system.
CPI003	The system TOD clock is not operational.	Call IBM for hardware support to fix the clock.
CVT001	The system TOD clock is in error or is not operational.	
DRD001	The device code index in the compressed DASD address for the system dump file points to an RDEVBLOK for an invalid DASD. The valid DASDs are 2305 series, 3330 series, 3340 series, 3350 series or 2314/2319.	Verify that the contents and order of the owned list have not been altered since the dump was taken. If these fields have not been altered, the SFBLOK for the dump file may have been destroyed. The owned list is specified by the SYSOWN macro in the DMKSYS module.
DSP001	During I/O interruption, unstack and reflection, DMKSCNVU could not locate all of the virtual control blocks for the interrupting unit.	The integrity of the user's virtual I/O configuration has probably been violated. The unit addresses or indexes in the virtual control blocks are in error, or the virtual configuration has been altered by ATTACH/DETACH while I/O was in progress. Check for a device reset failure in DMKCFPRD.

Figure 64. CP ABEND Codes (Part 2 of 15)

ABEND Code	Reason for ABEND	Action
DSP002	The dispatcher (DMKDSP) is attempting to dispatch a virtual relocate user whose shadow segment tables or virtual extended control register 0 are invalid.	Most likely, a free storage violation has occurred. First look at the DMKPRV and DMKVAT modules. Examine the real, virtual, and shadow translation tables for consistency of entry size and format. Also compare page and segment size.
DSP003	The interval timer was not incremented properly. This is most likely a hardware error. The dispatcher tests for interval timer errors and abnormally terminates if such an error occurs. Results would be unpredictable if CP continued when the interval timer was in error.	Check the timer fields in real storage. The value of the real interval timer is at real storage location X'50'. The dispatcher loads the value of the real interval timer in real storage location X'54' when a user is dispatched. The value of the real interval timer is loaded into real storage location X'4C' when an interrupt occurs. If the value stored at X'4C' is not less than the value stored at X'54', the dispatcher abnormally terminates. Check the routines that control the value of the time fields at X'4C', X'50', and X'54'.
DSP004	While tracing SIOs or I/O interrupts, the virtual device was detached. Now, the VDEVBLOK cannot be found.	Examine the operator's console sheet and the user's terminal sheet to see who detached the device. Warn the person responsible that devices should not be detached during I/O tracing.
FRE001	The size of the block being returned (via GR 0) is less than or equal to 0.	Using FREER14 and FREER12 in the PSA, identify the CP module releasing the storage. Check for an error in calculating the size of the block or for a modification to the stored block size for variable-size blocks.
FRE002	The address of the free storage block being returned matches the address of a block already in the free storage chain.	Identify the program returning the storage by means of the return address and base registers (FREE14 and FREE12 in DMKFRE's save area in PSA). The most common cause of this type of failure is a module that returns a free storage block but fails to clear a pointer to the block that has been saved elsewhere. All modules that return blocks via a call to DMKFRET should first verify that the saved pointer is nonzero; after returning the block, any saved pointers should be set to zero.

Figure 64. CP ABEND Codes (Part 3 of 15)

ABEND Code	Reason for ABEND	Action
FRE003	The address of the free storage block being returned overlaps the next lower block on the free storage chain.	A free storage pointer may have been destroyed. Also, the module releasing the lower (overlapped) block may have returned too much storage. Examine the lower block and determine its use and former owner. Or, identify the program returning the storage by means of the return address and base registers stored (FREER14 and FREER12 in DMKFRE's save area in PSA). The most common cause of this type of failure is a module that returns a free storage block but fails to clear a pointer to the block that has been saved elsewhere. All modules that return blocks via a call to DMKFRET should first verify that the saved pointer is nonzero; after returning the block, any saved pointers should be set to zero.
FRE004	The address of the free storage block being returned overlaps the next higher block on free storage chain.	A free storage pointer may have been destroyed. Also, the module releasing the higher (overlapped) block may have returned too much storage, or the module may be attempting to release storage at the wrong address.
FRE005	A module is attempting to release storage in the resident VM/370 nucleus.	A module is probably attempting to release location 0. Check for the module picking up a pointer to the free storage block without first testing the pointer for 0. Use FREER14 and FREER12 in the PSA to identify the module.
FRE006	A module is requesting a block of storage whose size (contained in GR 0) is less than or equal to zero.	Using FREER14 and FREER12 in the PSA, identify the module. Check for an error in calculating the block size. Improper use of the halfword instructions ICM and STCM can cause truncation of high order bits that results in a calculation error.
FRE007	A module is attempting to release a block of storage whose address exceeds the size of real storage.	A free storage pointer may have been destroyed. Attempt to identify the owners of the free storage blocks adjacent to the one containing the pointer that was destroyed. Check for moves and translation where initial counts of zero have been decremented to minus 1, thus generating an executed length code of X'FF', or an effective length of 256 bytes.

Figure 64. CP ABEND Codes (Part 4 of 15)

ABEND Code	Reason for ABEND	Action
FRE008	The address of the free storage block being returned matches the address of the first block in the subpool for that size.	Identify the program returning the storage by means of the return address and stored base registers (FREER14 and FREER12 in DMKFRE's save area in the PSA). The common cause of this type of failure is a module that returns a free storage block but fails to clear a pointer to the block that has been saved elsewhere. All modules that return blocks via a call to DMKFRET should first verify that the saved pointer is nonzero; after returning the block, any saved pointers should be set to zero.
FRE009	The address of the free storage block being returned matches the second block in the subpool for that size.	
FRE010	A program is attempting to extend free storage while storage is being extended. This can be caused by I/O interruptions or channel programs involving channels other than channel 0.	If the storage requests that caused the ABEND are due to channel activity, place the device involved on channel 0, which is disabled during free storage extension.
FRE011	A CP module has attempted to return a block of storage that is in the user dynamic paging area.	Identify the program returning the storage by means of the return address and stored base registers (FREER14 and FREER12 in DMKFREE's save area in the PSA). The common cause of this type of failure is a module that returns a free storage block but fails to clear a pointer to the block that has been saved elsewhere. All modules that return blocks via a call to DMKFRET should first verify that the saved pointer is nonzero; after returning the block, any saved pointers should be set to zero.
HVD001	The user pointed to by GR 11 issued a DIAGNOSE instruction while attempting to format the I/O error, channel check, or machine check recording areas: the SYSRES device type is unrecognizable.	The RDEVBLOK for the SYSRES device was probably destroyed, or a volume with the same serial number as the SYSRES volume was mounted. If a volume with the same serial number was mounted, check the ATTACH processing in the DMKVDB routine.

Figure 64. CP ABEND Codes (Part 5 of 15)

ABEND Code	Reason for ABEND	Action
IOS001	The caller is trying to reset an active IOBLOK from the RCHBLOK queue, but that IOBLOK contains an invalid address.	The IOBLOK may have been returned (via DMKFRET) or destroyed. Verify that the IOBLOK was valid and use the IOBLOK and RDEVBLOK to determine the last operation.
IOS002	DMKIOS is attempting to restart an IOBLOK from the RCHBLOK queue, but that IOBLOK contains an invalid address.	
IOS003	DMKIOS is attempting to remove an IOBLOK from a queue, but that IOBLOK contains an invalid address.	Register 2 points to the RCHBLOK, RCUBLOK, or RDEVBLOK from whose queue the IOBLOK is being removed. Register 10 points to the IOBLOK. Use the CP internal trace table to determine which module called DMKIOS twice to start the same IOBLOK.
NLD001	During execution of a NETWORK DUMP command, or during an automatic dump of a 3704 or 3705, VM/370 detected that it had not allocated sufficient DASD spool space to contain the information from the 3704 or 3705. The MODEL operand of the RDEVICE macro describing the 3704 or 3705 was not specified correctly. VM/370 determines the storage size of a 3704 or 3705 by the model specified on the RDEVICE macro.	Correct the RDEVICE macro specifying the 3704 or 3705, reassemble the DMKRIO module, and regenerate the VM/370 CP nucleus with the corrected module.
PGS001	The user page count in the VMBLOK (VMPAGES) became negative.	A module has attempted to release more pages than it originally received. The module that last called DMKPGS is probably the module in error.

Figure 64. CP ABEND Codes (Part 6 of 15)

ABEND Code	Reason for ABEND	Action
PGT001	The number of cylinders in use stored in the allocation block (ALOCBLOK) is less than the maximum but the DMKPGT module was unable to find available cylinders.	Inspect the chains of paging and spooling allocation blocks anchored at RDEVPAGE and RDEVRECS on the RDEVBLOK for the device in question, and verify that a cylinder allocation block (RECBLOK) exists for each cylinder marked and allocated in the ALOCBLOK. If RECBLOKs for some cylinders are missing, it is possible that the bit map in the ALOCBLOK has been destroyed. If all cylinders are accounted for, the updating of the count field is in error.
PGT002	The count of pages in a page allocation block (RECBLOK) is less than the maximum but the DMKPGT module was unable to find available pages.	If the RECBLOK in question is in use for paging, then locate a SWPTABLE entry for each page allocated on the cylinder. However, if the cylinder is in use for spooling, it is possible that the RECBLOK itself has been destroyed or that the updating of the use count is faulty.
PGT003	The DASD page slot being released is not marked allocated.	Identify the module attempting to release the page by means of the caller's return address and base register stored in BALR14 and BALR12 in the BALRSAVE save area in PSA. Locate the source (control block or SWPTABLE entry) of the DASD address being released to verify that they have not been destroyed. If the DASD page is in a spool file, it is possible that the file or the RECBLOK chain has been incorrectly checkpointed and warmstarted after a system shutdown or a system crash.
PGT004	The dummy RECBLOK indicating the spooling DASD pages on the cylinder that are to be released contains a page count greater than the number of pages allocated on the cylinder.	The spool file pointers may have been destroyed while the file was being processed, or the allocation chain may be in error. A cold start may be necessary. If feasible, use the DASD dump restore program to print the DASD areas containing the affected file, and try to locate the incorrect pointers.

Figure 64. CP ABEND Codes (Part 7 of 15)

ABEND Code	Reason for ABEND	Action
PGT005	A module is trying to release a DASD page slot on a cylinder for which no page allocation block (RECBLOK) exists.	Use BALR14 and BALR12 in the BALRSAVE area of the PSA to identify the module attempting to release the page. Verify that the DASD cylinder address is valid for the device in question. If it is and the rest of the DASD address is valid, verify that the cylinder is in the dynamically allocatable area. If these restrictions are met, the DASD page must have been used by more than one user.
PGT006	The last DASD page slot in a RECBLOK has been deallocated but the bit representing the cylinder in the cylinder allocation block (ALOCBLOK) is not currently set to one, indicating that the cylinder was not allocated.	The ALOCBLOK has probably been destroyed, or the chain pointer in the RDEVBLOK is in error.
PGT007	A module is trying to release a page of virtual storage in use by the VM/370 control program that has not been marked allocated.	Use BALR14 and BALR12 in the BALRSAVE area of the PSA to identify the module attempting to release the page. Locate the control block containing the virtual page address that is being released. It is possible that the address has been destroyed, or a pointer to a virtual page has been retained after the page was destroyed.
PGT008	The system's virtual storage buffers have been exhausted because of an excessive number of open spool files.	Request users to close all spool files that are no longer active.
PRG001	Program check (operation) in the control program.	Examine the ABEND dump. In particular, examine the old PSW and identify the module that had the program check.
PRG002	Program check (privileged operation) in the control program.	
PRG003	Program check (execute) in the control program.	
PRG004	Program check (protection) in the control program.	

Figure 64. CP ABEND Codes (Part 8 of 15)

ABEND Code	Reason for ABEND	Action
PRG005	Program check (addressing) in the control program.	Examine the ABEND dump. In particular, examine the old PSW and identify the module that had the program check.
PRG006	Program check (specification) in the control program.	
PRG007	Program check (data) in the control program.	
PRG008	Program check (fixed-point overflow) in the control program.	
PRG009	Program check (fixed-point divide) in the control program.	
PRG010	Program check (decimal overflow) in the control program.	
PRG011	Program check (decimal divide) in the control program.	
PRG012	Program check (exponential overflow) in the control program.	
PRG013	Program check (exponential underflow) in the control program.	
PRG014	Program check (significance) in the control program.	
PRG015	Program check (floating-point divide in the control program.	
PRG016	Program check (segment) in the control program.	
PRG017	Program check (paging) in the control program.	
PRG018	Program check (translation) in the control program.	
PRG019	Program check (special operation) in the control program.	

Figure 64. CP ABEND Codes (Part 9 of 15)

ABEND Code	Reason for ABEND	Action
PRG254	A translation specification exception has been received for a virtual machine that is not in extended control mode.	If the set of translation tables pointed to by RUNCR1 is correct, a hardware failure has occurred, possibly with dynamic address translation. Otherwise, call IBM for software support.
PRG255	A PER (program event recording) has been received for a virtual machine that is running with PER disabled in its virtual PSW.	Retry the program causing the error; if the problem persists, call IBM for software support.
PSA001	No free storage is available for save areas.	Try to identify the extreme load condition that caused the problem. Verify that a routine has not requested an inordinate amount of storage. If the storage requests are valid and the problem occurs regularly, alter the DMKCPI module to allocate more than six pages of free storage per 256K bytes of storage.
PSA002	The 'PSW Restart' console key was pressed and caused this ABEND. The operator normally takes this action when an unusual system condition occurs, such as a system loop or slow machine operation.	Examine the resulting ABEND dump for a dynamic picture of the system's status.
PSA003	An unrecoverable DASD I/O error occurred on a paging device.	Check the unit address in the I/O old PSW to find the paging device in error. This is a hardware error. Call IBM for hardware support.
PTR001	A segment exception or translation specification has occurred while executing a LOAD REAL ADDRESS (LRA) instruction in the DMKPTR module.	Inspect the contents of control registers 0 and 1, and the format of the segment table pointed to by CR 1. One or more of these tables and registers may contain invalid data. If CR 1 is invalid, check the contents of the VMBLOK pointed to by GR 11, especially the address in the VMSEG field.
PTR002	A program is attempting to unlock a page frame whose address exceeds the size of real storage.	Use BALR14 and BALR12 in the BALRSAVE area of the PSA to identify the module attempting to unlock the page frame. Check for the source of the invalid address.

Figure 64. CP ABEND Codes (Part 10 of 15)

ABEND Code	Reason for ABEND	Action
PTR003	A program is attempting to unlock a real storage page frame whose CORTABLE entry is not flagged as locked.	Use BALR14 and BALR12 in the BALRSAVE area of the PSA to identify the module attempting to unlock the page frame. Check for the source of the invalid address.
PTR004	The lock count in the CORTABLE entry for the page frame being unlocked has been decremented to a value that is less than 0.	Check the routines that update the lock count field and CORTABLE entry.
PTR005	The user page count in the VMBLOK (VMPAGES) is negative.	A module attempted to release more pages than it originally received. The last module that called DMKPTR is probably the module that caused the error.
PTR007	DMKFRE requested a page for fixed free storage but DMKPTR determined that there were no pages left in the dynamic paging area.	Examine the dump for one of the following conditions: 1. Excessive amounts of free storage have been allocated by CP and not released via DMKFRET. Look for blocks of identical data and determine which modules built that data. 2. A block of storage greater than 4096 bytes was requested. Requests for large blocks of free storage require contiguous pages from DMKPTR and as a result have a higher probability of failure than requests for one page or less. If possible, change the application to reduce the size of storage requests. Otherwise, schedule the application when storage is less fragmented.
PTR008	A CORTABLE entry on the free list points to a valid PTE (page table entry), but the page is allocated.	Pages on the free list should not contain valid PTEs. Examine the dump to determine which module called DMKPTRFR. The module that called DMKPTRFR probably contains an error.
PTR009	The count of the number of resident shared pages was incorrectly decremented making the count now less than zero.	The field DMKPTRSC contains the number of resident shared pages and the field DMKDSPNP contains the number of pageable pages. DMKDSPNP must always be greater than DMKPTRSC. Check the routines that update these two count fields.

Figure 64. CP ABEND Codes (Part 11 of 15)

ABEND Code	Reason for ABEND	Action
PTR010	The count of the number of resident reserved pages was incorrectly decremented so that the count is now less than zero.	The field DMKPTRRC contains the number of reserved pages. DMKPTRRC must always be less than DMKDSPNP. Check the routines that update these two count fields (DMKDSPNP and DMKPTRRC).
PTR011	A CORTABLE entry to be placed on the free list points to a valid PTE (page table entry), but the page is allocated. An abend occurs trying to honor a deferred request.	Pages to be put on the free list should not contain valid PTEs. Examine the dump to determine why the page was not marked invalid before the call to DMKPTRFT.
PTR012	A CORTABLE entry to be placed on the free list points to a valid PTE (page table entry), but the page is allocated.	Pages to be put on the free list should not contain valid PTEs. Examine the dump to determine why the page was not marked invalid before the call to DMKPTRFT.
PTR013	DMKFRE requested a page for fixed free storage but there were no DASD page slots left to write out the selected page.	Examine the dump to determine what was using all the TEMP space. Excessive space may be consumed by large spool files or not enough TEMP space exists for paging.
RGA001	The reflected device status in the CSW is not supported for certain 3270 remote device and line protocol I/O operations. Specifically, the returned CSW contains a device status other than CE, DE, and UE; and, the ending CCW contains an embeded teleprocessing code of 02, 03, or 06.	IPL to restart the system. If the problem persists, call IBM for system support.
RGA002	The status flag BSCFLAG in the BSCBLOK indicates a condition that is not valid for a 3270 line reset function (Teleprocessing code 09).	
RNH001	An unrecoverable I/O error occurred during read or write for the 3704 or 3705. Status indicates program failure.	Retry. If the problem persists, ensure that the 3704/3705 and channel hardware are functioning correctly.

Figure 64. CP ABEND Codes (Part 12 of 15)

ABEND Code	Reason for ABEND	Action
RNH002	A response that should not occur was received from the 3704/3705 control program.	Verify that the 3704/3705 NCP is operating correctly. Use the NETWORK TRACE command to determine the exact cause of the response.
RPA001	The virtual address supplied to DMKRPAGT is outside of the virtual storage being referenced.	The virtual storage belongs either to the user whose VMBLOK is pointed to by GR 11 or, if GR 2 in the SAVEAREA indicates a PARM of SYSTEM, to the system VMBLOK. Identify the calling program by means of the return address and base register saved in the SAVEAREA pointed to by GR 13. If the virtual address was obtained from the system's virtual storage, examine the virtual page allocation routine, DMKPTRVG. If the virtual page refers to a user's storage, attempt to identify the routine that has generated the incorrect address. Verify that the VMSIZE in the relevant VMBLOK reflects the correct storage size for the system or user being referenced.
RPA002	The virtual address supplied to DMKRPAPT is outside of the virtual storage being referenced.	
RPA003	The user page count in the VMBLOK became negative.	A module has attempted to release more pages than it originally received. The module that last called DMKRPA is probably the module in error.
SCH001	The total number of interactive users plus batch users in the scheduler's queue is less than zero. A counter was probably decremented incorrectly.	The field SCHN1 is the count of the number of interactive users and the field SCHN2 is the count of the number of batch users. Check the routines that update these two count fields (SCHN1 and SCHN2) to determine why their sum was negative.
SCN001	The VDEVLINK chain is invalid. A VDEVBLOK has a link field that points to another VDEVBLOK associated with the same real device. The first VDEVBLOK is not pointed to by any other link field in the chain.	IPL to restart. If the problem persists, examine the VDEVBLOKs in the link chain as well as the one whose link field points into the chain but is not in the chain. Determine what the owner of the VDEVBLOK was doing at the time.

Figure 64. CP ABEND Codes (Part 13 of 15)

ABEND Code	Reason for ABEND	Action
TDK001	A program is attempting to deallocate a cylinder of T-disk space for which no cylinder allocation block (ALOCBLOK) exists.	Verify that GR 8 points to a RDEVBLOK for a CP-owned volume. If it does not, the error may originate in the calling program. Identify the caller by the return address and base register in the SAVEAREA pointed to by GR 13, and try to identify the source of the incorrect RDEVBLOK address. If the RDEVBLOK is valid, it may be that the cylinder number passed is incorrect. The VDEVBLOK for the device for which the T-disk was defined may have been destroyed. If the cylinder number appears valid, examine the allocation record on the real volume by running DMKFMT (VM/370 Format program), invoking the ALLOCATE option without allocating any new space. If the output shows that deallocated cylinder falls within an area defined for T-disk allocation, the ALOCBLOK chained to the RDEVBLOK may be destroyed.
TDK002	A program is attempting to deallocate cylinder(s) of T-disk space that are not marked allocated.	
UDR001	The user directory module is looping trying to read all of the UDIRBLOK page buffers from the directory device. Or, a directory containing over 10,816 users was loaded.	Use the DASD Dump Restore program to print the UDIRBLOK page buffers from the directory device. Determine if the chain pointers are valid.
VDB002	The system-owned list has an invalid format.	IPL to restart. If the problem persists, check the SYSOWN macro in DMKSYS for validity. If the macro is good, print the dump and examine it.
VDR003	The DASD link chain is invalid. In the case of minidisks, attaching a minidisk that points to an RDEVBLOK whose count of users is already zero causes this ABEND.	IPL to restart. If the problem persists, examine the RDEVSYS flag. If the RDEVSYS flag is off, the problem is especially serious; print and examine the dump. Examine the VDEVBLOK and RDEVBLOK checking the link chain.
VIO002	DMKSCNVU was unable to locate all of the virtual I/O control blocks for the virtual unit address associated with the interrupt just stacked.	Verify that the unit address in the field IOBVADD in the IOBLOK pointed to by GR 10 is valid for the user who initiated the I/O. The field IOBUSER contains the address of the user's VMBLOK. If the address is valid, the integrity of the user's virtual I/O

Figure 64. CP ABEND Codes (Part 14 of 15)

ABEND Code	Reason for ABEND	Action
VI0002 (cont.)		configuration has probably been been destroyed. If the address is not valid, the IOBLOK has been altered, or was built incorrectly in the first place.
VI0003	DMKICS has returned an IOBLOK indicating a condition code of 2 was received from the START I/O for the operation.	Condition code 2 should never be returned to the virtual I/O interrupt handler. Its presence indicates either a failure in the I/O supervisor (DMKIOS), or that the status field in the IOBLOK (IOBSTAT) has been destroyed.
VMA001	DMKVMASH was called to check if any shared pages were altered. A VMABLOK associated with a shared named system could not be found.	Examine BALR14 for the address of the module that issues the call. The probable cause of error is that the VMBLOK has been overlaid. Examine the CP trace table entries and determine when the VMBLOK was overlaid.
VMA002	DMKVMA was called to make a shared named system unshared. However, the SHRTABLE associated with the shared page that was changed could not be located.	The SHRTABLE may have been overlaid or the shared page that was changed was altered by another virtual machine. If the SHRTABLE was not overlaid find out which virtual machine altered the shared page and why it was not detected.
VMA003	A shared page was changed and a named system could not be found for the virtual machine.	A shared page was alterd by another virtual machine and went by undetected. Investigate system routines that could allow the undetected alteration of a shared page.
VMA004	A shared page was changed and the corresponding VMABLOK could not be found.	A shared page was altered by another virtual machine without being detected. Investigate the system routines that could allow an undetected alteration of a shared page.
VSP001	The virtual spooling manager could not locate all virtual control blocks for an interrupting unit.	Verify that the unit address (IOBVADD) in the IOBLOK is valid. If the address is valid, the integrity of the virtual I/O configuration has probably been destroyed. If the address is not valid, the IOBLOK has been altered or was built incorrectly.

Figure 64. CP ABEND Codes (Part 15 of 15)

CMS RETURN CODES

If a condition arises during execution of a command which types out a Warning, Error, Severe or Terminal error message, the command passes a nonzero return code in register 15. These return codes have the following values:

Code	Meaning
RC = 4:	The user did not specify all the conditions to execute the command as intended but these conditions did not prevent the command from completing execution. However, the results are unpredictable.
RC = 8:	Device errors for which a warning message is issued, or errors have been introduced into the output file.
RC = 12	Errors in input file.
RC = 20:	Invalid character in fileid. The valid characters are 0-9, a-z, A-Z, at-sign, pound sign, dollar sign.
RC = 24:	The user did not specify the command line correctly.
RC = 28:	Error occurred while trying to access, or manipulate a user's files. For example: file not found.
RC = 32:	The user's file(s) is not in the expected format, or the user's file(s) does not contain the expected information.
RC = 36:	Error occurred involving the user's devices for which he is responsible. For example: disk is read-only.
RC = 40:	Functional error occurred executing the command for which the user is responsible, or the user failed to supply all the necessary conditions for executing the command; or, end of file or end of tape occurred (where applicable).
RC = 88:	A CMS system restriction prevented command execution, or the requested function is an unsupported feature, or the device requested is unsupported.
RC = 100:	I/O errors, or serious device errors occurred.
RC = 104:	A functional error occurred during command execution for which the system is responsible.
RC = 256:	All unexpected errors for which the system is responsible, that is, Terminal messages.

If command execution generates no Warning, Error, Severe or Terminal error messages, the return code passed in register 15 is zero.

Commands which invoke program products pass to the user the return code passed by the program in register 15. OS Simulation routines indicate return codes within the text of the messages. Commands or functions of commands passed to CP pass the return code passed by CP in register 15.

CMS DMSFREX ERROR CODES

ERROR CODES FROM DMSFREE, DMSFRES, AND DMSFRET

A nonzero return code upon return from DMSFRES, DMSFREE or DMSFRET indicates that the request could not be satisfied. Register 15 contains this return code, indicating which error has occurred. The codes below apply to the DMSFRES, DMSFREE and DMSFRET macros, described on the following pages.

Code	Error
1	(DMSFREE) Insufficient storage space is available to satisfy the request for free storage. In the case of a variable request, the minimum request could not be satisfied.
2	(DMSFREE or DMSFRET) User storage pointers destroyed.
3	(DMSFREE or DMSFRET) Nucleus storage pointers destroyed.
4	(DMSFREE) An invalid size was requested. This error exit is taken if the requested size is not greater than zero. In the case of variable requests, this error exit is taken if the minimum request is greater than the maximum request. However, the error is not detected if DMSFREE is able to satisfy the maximum request.
5	(DMSFRET) An invalid size was passed to the DMSFRET macro. This error exit is taken if the specified length is not positive.
6	(DMSFRET) The block of storage that is being released was never allocated by DMSFREE. This error occurs if one of the following errors is found: a. The block is not entirely inside either the low-core free storage area or the user program area between FREELOWE and FREEUPPR. b. The block crosses a page boundary that separates a page allocated for USER storage from a page allocated for NUCLEUS storage. c. The block overlaps another block already on the free storage chain.
7	(DMSFRET) The address given for the block being released is not a doubleword boundary address.
8	(DMSFRES) An illegal request code was passed to the DMSFRES routine. Because the DMSFRES macro generates all codes, this error code should never appear.
9	(DMSFRE, DMSFRET, or DMSFRES) Unexpected internal error.

CMS ABEND CODES

ABEND RECOVERY

Modules Used: DMSABN

Operation of the Abend Routine, DMSABN

When the abend recovery routine is entered, it types out the abend message, followed by the line 'CMS', to indicate to the user that he may type in his next command.

At this point, there are two options available to the user.

First, he may type the DEBUG command. In this case, DMSABN passes control to DMSDBG, to make the facilities of DEBUG available to him. DEBUG's PSW and registers are as they were at the time that the abend recovery routine was invoked. From DEBUG, the user may alter the PSW or registers, as he wishes, and type GO to continue processing, or type RETURN to return to DMSABN, so that abend recovery can continue.

The second option available is to type in any other command. If this is done, DMSABN performs its abend recovery function and passes control to DMSINT to execute the command that has been typed in.

The abend recovery function consists of the following steps:

1. The SVC handler, DMSITS, is reinitialized, and all stacked save areas are released.
2. "FINIS * * *" is invoked by means of SVC 202, to close all files, and to update the user file directory.
3. If the EXEC interpreter (EXECTOR module) is in storage, it is released.
4. All link blocks allocated by the OS macros simulation routine DMSSLN are freed.
5. If VSAM or Access Method Services are still active, call DMSVSR for cleanup.
6. All FCB and DOSCB pointers are zeroed out.
7. All user storage is released.
8. The amount of system free storage that should be allocated is computed. This figure is compared against the amount of free storage that is actually allocated. If the two are equal, then storage recovery can be considered successful. If they are unequal, then a message is sent to the user.

Unrecoverable Termination -- The HALT Option of DMSERR

There are certain times, such as when the SVC handler's pointers are modified, that the system can neither continue processing nor try to recover. In these cases, DMSERR with the option HALT=YES is specified to cause a message to be typed out, after which a disabled wait state PSW is loaded.

In CP mode, the programmer can examine the PSW, whose address field contains the address of the instruction following the call to the DMSERR macro. He can also examine all the registers, which are as they were when the DMSERR macro was invoked.

Figure 65 lists the CMS ABEND codes and describes the cause of the ABEND and the action required.

ABEND Code	Module Name	Cause of ABEND	Action
001	DMSSCT	The problem program encountered an input/output error processing an OS macro. Either the associated DCB did not have a SYNAD routine specified or the I/O error was encountered processing an OS CLOSE macro.	Message DMSSCT120S indicates the possible cause of the error. Examine the error message and take the action indicated.
034	DMSVIP	The problem program encountered an I/O error while processing a VSAM action macro under DOS/VS for which there is no OS equivalent. An internal error occurred in a DOS VSAM routine.	Refer to the <u>DOS/VS Messages Reference</u>, Order No. GC33-5379, to determine the cause of the VSAM error.
0Cx	DMSITP	The specified hardware exception occurred at a specified location. "x" is the type of exception: <u>x</u> <u>Type</u> 0 IMPRECISE 1 OPERATION 2 PRIVILEGED OPERATION 3 EXECUTE 4 PROTECTION 5 ADDRESSING 6 SPECIFICATION 7 DECIMAL DATA 8 FIXED-POINT OVERFLOW 9 FIXED-POINT DIVIDE A DECIMAL OVERFLOW B DECIMAL DIVIDE C EXPONENT OVERFLOW D EXPONENT UNDERFLOW E SIGNIFICANCE F FLOATING-POINT DIVIDE	Type DEBUG to examine the PSW and registers at the time of the exception.
0F0	DMSITS	Insufficient free storage is available to allocate a save area for an SVC call.	If the ABEND was caused by an error in the application program, correct it; if not, use the CP DEFINE command to increase the size of virtual storage and then restart CMS.
0F1	DMSITS	An invalid halfword code is associated with SVC 203.	Enter DEBUG and type GO. Execution continues.

Figure 65. CMS ABEND Codes (Part 1 of 3)

ABEND Code	Module	Cause of ABEND	Action
0F2	DMSITS	The CMS nesting level of 20 has been exceeded.	None. ABEND recovery takes place when the next command is entered.
0F3	DMSITS	CMS SVC (202 or 203) instruction was executed and provision was made for an error return from the routine processing the SVC call.	Enter DEBUG and type GO. Control returns to the point to which a normal return would have been made.
0F4	DMSITS	The DMSKEY key stack overflowed.	Enter DEBUG and type GO. Execution continues and the DMSKEY macro is ignored.
0F5	DMSITS	The DMSKEY key stack underflowed.	
0F6	DMSITS	The DMSKEY key stack was not empty when control returned from a command or function.	Enter DEBUG and type GO. Control returns from the command or function as if the key stack had been empty.
0F7	DMSFRE	Occurs when TYPCALL=SVC (the default) is specified in the DMSFREE or DMSFRET macro.	When a system ABEND occurs, use DEBUG to attempt recovery.
0F8	DMSFRE	Occurs when TYPCALL=BALR is specified in the DMSFREE or DMSFRET Macro devices.	When a system ABEND occurs, use DEBUG to attempt recovery.
101	DMSSVN	The wait count specified in an OS WAIT macro was larger than the number of ECBs specified.	Examine the program for excessive wait count specification.
104	DMSVIB	The OS interface to DOS/VS VSAM is unable to continue execution of the problem program.	See the additional error message accompanying the ABEND message, correct the error, and reexecute the program.
155	DMSSLN	Error during LOADMOD after an OS LINK, LOAD, XCTL, or ATTACH. The compiler switch is on.	See the last LOADMOD (DMSMOD) error message for error description. In the case of an I/O error, recreate the module. If the module is missing, create it.
15A	DMSSLN	Severe error during load (phase not found) after an OS LINK, LOAD, XCTL, or ATTACH. The compiler switch is on.	See last LOAD error message (DMSLIO) for the error description. In the case of an I/O error, recreate the

Figure 65. CMS ABEND Codes (Part 2 of 3)

ABEND Code	Module Name	Cause of ABEND	Action
15A cont.			text deck or TXTLIB. If either is missing, create it.
174	DMSVIB	The OS interace to DOS/VS VSAM is unable to continue execution of the problem program.	See the additional error message accompanying the ABEND message, correct the error, and reexecute the program.
177	DMSVIB DMSVIP	The OS interface to DOS/VS VSAM is unable to continue execution of the problem program.	See the additional error message accompanying the ABEND message, correct the error, and reexecute the program.
240	DMSSVT	No work area was provided in the parameter list for an OS RDJFCB macro.	Check RDJFCB specification.
400	DMSSVT	An invalid or unsupported form of the OS XDAP macro has been issued by the problem program.	Examine program for unsupported XDAP macro or for SVC 0.
704	DMSSMN	An OS GETMAIN macro (SVC 4) was issued specifying the LC or LU operand. These operands are not supported by CMS.	Change the program so that it specifies allocation of only one area at a time.
705	DMSSMN	An OS FREEMAIN macro (SVC 5) was issued specifying the L operand. This operand is not supported by CMS.	Change the program so that it specifies the release of only one area at a time.
804 80A	DMSSMN	An OS GETMAIN macro (804 - SVC 4, 80A - SVC 10) was issued that requested either zero bytes of storage, or more storage than was available.	Check the program for a valid GETMAIN request. If more storage was requested than was available, increase the size of the virtual machine and retry.
905 90A	DMSSMN	An OS FREEMAIN macro (905 - SVC 5, 90A - SVC 10) was issued specifying an area to be released whose address was not on a doubleword boundary.	Check the program for a valid FREEMAIN request; the address may have been incorrectly specified or modified.
A05 A0A	DMSSMN	An OS FREEMAIN macro (A05 - SVC 5, A0A - SVC 10) was issued specifying an area to be released which overlaps an existing free area.	Check the program for a valid FREEMAIN request; the address and/or length may have been incorrectly specified or modified.

Figure 65. CMS ABEND Codes (Part 3 of 3)

RSCS MESSAGE-TO-LABEL CROSS REFERENCE

Message Code	Generated at Label	Message Text
DMTAXS101I	TAGPEND	FILE 'spoolid' ENQUEUED ON LINK 'linkid'
DMTAXS102I	ACCEPEND	FILE 'spoolid' PENDING FOR LINK 'linkid'
DMTAXS103E	ACCEPURG	FILE 'spoolid' REJECTED -- INVALID DESTINATION ADDRESS
DMTAXS104I	CLOOSCAN	FILE SPOOLED TO 'userid2' -- ORG 'locid1' ('userid1') mm/dd/yy hh:mm:ss
DMTAXS105I	CLOIPURG	FILE 'spoolid' PURGED
DMTAXS106I	FILSTRY	FILE 'spoolid'MISSING -- DEQUEUED FROM LINK 'linkid'
	OPENPOOF	
DMTAXS107I	UNPECHEK	nn PENDING FILES FOR LINK 'linkid' MISSING
DMTAXS108E	OPENRDER	SYSTEM ERROR READING SPOOL FILE 'spoolid'
DMTAXS520I	CHANGE	File 'spoolid' CHANGED
DMTAXS521I	CHANHO	FILE 'spoolid' HELD FOR LINK 'linkid'
DMTAXS522I	CHANNOH	FILE 'spoolid' RELEASED FOR LINK 'linkid'
DMTAXS523I	CHANSCAN	LINK 'linkid' QUEUE REORDERED
	ORDENEXT	
DMTAXS524E	CHANGE	FILE 'spoolid' ACTIVE -- NO ACTION TAKEN
	ORDECHEK	
	PURGCHEK	
DMTAXS525E	CHANGE	FILE 'spoolid' IS FOR LINK 'linkid' -- NO ACTION TAKEN
	ORDECHEK	
	PURGCHEK	
DMTAXS526E	CHANGE	FILE 'spoolid' NOT FOUND -- NO ACTION TAKEN
	ORDECHEK	
	PURGCHEK	
DMTAXS640I	PURGDONE	nn FILE(S) PURGED ON LINK 'linkid'
DMTCMX001I	CMXFINXT	FREE STORAGE = nn PAGES
DMTCMX003I	CMXM003	LINK 'linkid' EXECUTING: (command line)
DMTCMX200I	CMXLGOT	RSCS
DMTCMX201E	CMXHIT	INVALID COMMAND 'command'
	CMXLGOT	
	CMXMISS	
DMTCMX202E	DEFNOLNK	INVALID LINK 'linkid'
	MSGNOLNK	
DMTCMX203E	A1FLKGOT	INVALID SPOOL FILE ID 'spoolid'
	A1FSTOW	
	CHALKGOT	
	L2FLKGOT	
	QYOFILE	
	QYOFNULL	
DMTCMX204E	CHALKGOT	INVALID KEYWORD 'keyword'
	CHANTERM	
	CHASCAN	
	FLUMORE	
	LOTERM	
	L1TERM	
	QYTOOMCH	
	QYOFILE	
	QYOLINK	
	QYOSYSTM	
	ROSCAN	
DMTCMX205E	CHACLASS	CONFLICTING KEYWORD 'keyword'
	CHACOPY	

Message Code	Generated at Label	Message Text
DMTCMX205E (cont.)	CHAHOLD CHANOHOL CHAPRIOR FLUKEYWD LCTKEYWD ROCLASS ROKEEP ROLINE ROTASK ROTYPE	
DMTCMX206E	CHACLASS CHACOPY CHADIST CHANAME CHAPRIOR LOHOLD LOTRACE L1FLKGOT QUERY ROCLASS ROCLMULT ROKEEP ROLINE ROTASK ROTYPE	INVALID OPTION 'keyword' 'option'
DMTCMX208E	DISCONN MSGNOLNK MSGNOUSR	INVALID USER ID 'userid'
DMTCMX300I	CMXALRDY	ACCEPTED BY TASK 'task'
DMTCMX301E	CMXALRDY	REJECTED BY TASK 'task' -- PREVIOUS COMMAND ACTIVE
DMTCMX302E	MSGNOLNK	LINK 'linkid' IS NOT DEFINED
DMTCMX303E	CMD L0FLKGOT L1FLKGOT L2FLKGOT MSG	LINK 'linkid' IS NOT ACTIVE
DMTCMX304E	CMXALRDY	REJECTED BY TASK 'task' NOT RECEIVING
DMTCMX540I	DEFLKNEW	NEW LINK 'linkid' DEFINED
DMTCMX541I	DEFLKNEW	LINK 'linkid' REDEFINED
DMTCMX542E	DEFINE	LINK 'linkid' ACTIVE -- NOT REDEFINED
DMTCMX543E	DEFNEXT DEFNOLNK	LINK 'linkid' NOT DEFINED -- LINK LIMIT REACHED
DMTCMX544E	DEFLKNEW	LINK 'linkid' NOT DEFINED -- LIMIT REACHED
DMTCMX550I	DELDELET	LINK 'linkid' NOW DELETED
DMTCMX551E	DELETE	LINK 'linkid' ACTIVE -- NOT DELETED
DMTCMX552E	DELETE	LINK 'linkid' HAS A FILE QUEUE -- NOT DELETED
DMTCMX560I	DISCHARG	RSCS DISCONNECTING
DMTCMX561E	DISCONN	USERID 'userid' NOT RECEIVING
DMTCMX651I	QY1STAT	LINK 'linkid' INACTIVE
DMTCMX652I	QY1SNOD	LINK 'linkid' ACTIVE 'type' 'vaddr' c (HO|NOH) (DR|NOD) (TRA|TRE|NOT) Q=m P=n
DMTCMX653I	QY1DEF	LINK 'linkid' DEFAULT 'task' 'type' 'vaddr' c R=m
DMTCMX654I	QY1QUEUE	LINK 'linkid' Q=m P=n
DMTCMX655I	QY1INACT	FILE 'spoolid' 'locid' 'userid' CL a PR mm REC nnnnnn
DMTCMX660I	QY2STAT	FILE 'spoolid' INACTIVE ON LINK 'linkid'
DMTCMX661I	QY2STAT	FILE 'spoolid' ACTIVE ON LINK 'linkid'
DMTCMX662I	QY2RSS	FILE 'spoolid' ORG 'locid' 'userid' mm/dd/yy hh:mm:ss zzz TO 'locid' 'userid' VIA 'linkid'
DMTCMX663I	QY2VNOH	FILE 'spoolid' PR mm CL a CO nn (HO|NOH) DI 'distcode', NA ('fn ft'|'dsname')

Message Code	Generated at Label	Message Text
DMTCMX664E	QY2RSS QY2STAT QY2VM QY2VNOH	FILE 'spoolid' NOT FOUND
DMTCMX670I	QYSYACT	LINK 'linkid' ACTIVE -- LINE 'vaddr' (HO|NOH)
DMTCMX671I	QYM671	LINK 'linkid' INACTIVE
DMTCMX672I	QYSYNEXT	NO LINK ACTIVE
DMTCMX673I	QYM673	NO LINK DEFINED
DMTCMX700I	STALNGOT	ACTIVATING LINK 'linkid' 'task' 'type' 'vaddr'
DMTCMX701E	STACREAT	NO SWITCHED LINE AVAILABLE -- LINK 'linkid' NOT ACTIVATED
DMTCMX702E	STACREAT	LINE 'vaddr' IS IN USE BY LINK 'linkid1' -- LINK 'linkid2' NOT ACTIVATED
DMTCMX703E	STACREAT	DEV 'cuu' IS NOT A LINE PORT -- LINK 'linkid' NOT ACTIVATED
DMTCMX704E	STACREAT	LINE 'vaddr' CC=3 NOT OPERATIONAL -- LINK 'linkid' NOT ACTIVATED
DMTCMX705E	STACRERR	DRIVER 'type' NOT FOUND ON DISK 'vaddr' -- LINK 'linkid' NOT ACTIVATED
DMTCMX706E	STACRERR	FATAL ERROR LOADING FROM 'vaddr' -- LINK 'linkid' NOT ACTIVATED
DMTCMX707E	STACRERR	DRIVER 'type' FILE FORMAT INVALID -- LINK 'linkid' NOT ACTIVATED
DMTCMX708E	STACRERR	VIRTUAL STORAGE CAPACITY EXCEEDED -- LINK 'linkid' NOT ACTIVATED
DMTCMX709E	STACRERR	TASK NAME 'task' ALREADY IN USE -- LINK 'linkid' NOT ACTIVATED
DMTCMX710E	STAMAXER	MAX (nn) ACTIVE -- LINK 'linkid' NOT ACTIVATED
DMTCMX750E	STANOTCL	LINK 'linkid' ALREADY ACTIVE -- NO ACTION TAKEN
DMTCMX751I	CMXALRDY	LINK 'linkid' ALREADY ACTIVE -- NEW CLASS(ES) SET AS REQUESTED
DMTINI402T	INIEXIT	IPL DEVICE READ I/O ERROR
DMTINI407R	ASKQUEST	REWRITE THE NUCLEUS? Y OR N
DMTINI408R	IPLDISK	IPL DEVICE ADDRESS = ccu
DMTINI409R	NUCCYLN	NUCLEUS CYL ADDRESS = nnn
DMTINI410R	IPLZERO	ALSO IPL CYLINDER 0? Y OR N
DMTINI431S	WRERROR	IPL DEVICE WRITE I/O ERROR
DMTINI479E	BINERR1	INVALID DEVICE ADDRESS - REENTER
DMTINI480E	DECERR1 RDORWRT	INVALID CYLINDER NUMBER - REENTER
DMTINI481E	IPLZERO	INVALID REPLY - ANSWER YES OR NO
DMTINI482E	BADIPLD	IPL DEVICE ERROR - REENTER
DMTINI483E	NUCCYLN	NUCLEUS OVERLAYS CMS FILES - RECOMPUTE
DMTNPT070E	IOERRPRT	ERROR cuu SIOCC cc CSW csw SENSE sense CCW ccw
DMTNPT108E	VMSGET	SYSTEM ERROR READING SPOOL FILE 'spoolid'
DMTNPT141I	NPTEINIT	LINE 'vaddr' READY FOR CONNECTION TO LINK 'linkid'
DMTNPT142I	NPTEINIT	LINK 'linkid' LINE 'vaddr' CONNECTED
DMTNPT143I	LINEDIS2 LINEDROP	LINK 'linkid' LINE 'vaddr' DISCONNECTED
DMTNPT144I	PUTOPEN	RECEIVING: FILE FROM 'locid1' ('name1') FOR 'locid2' ('name2')
DMTNPT145I	PUTCLS1	RECEIVED: FILE FROM 'locid1' ('name1') FOR 'locid2' ('name1')
DMTNPT146I	GETGOT2	SENDING: FILE 'spoolid' ON LINK 'linkid', REC nnnnnn
DMTNPT147I	GETPURGE	SENT: FILE 'spoolid' ON LINK 'linkid'
DMTNPT149I	TRPRT	LINK 'linkid' LINE ACTIVITY: TOT= mmm; ERRS= nnn; TMOUTS= ppp
DMTNPT190E	VMSP1	INVALID SPOOL BLOCK FORMAT ON FILE 'spoolid'
DMTNPT510I	GTBKMSG	FILE 'spoolid' BACKSPACED
DMTNPT511E	SBKFWDN	NO FILE ACTIVE ON LINK 'linkid'

Message Code	Generated at Label	Message Text
DMTNPT570I	SETDRAIN	LINK 'linkid' NOW SET TO DEACTIVATE
DMTNPT571E	SETDRER1	LINK 'linkid' ALREADY SET TO DEACTIVATE
DMTNPT580I	GETFLUSH	FILE 'spoolid' PROCESSING TERMINATED
DMTNPT581E	SETFLUSH GETFLSHE	FILE 'spoolid' NOT ACTIVE
DMTNPT590I	SETFREE	LINK 'linkid' RESUMING FILE TRANSFER
DMTNPT591E	SETFRER1	LINK 'linkid' NOT IN HOLD STATUS
DMTNPT600I	GDGODNE	FILE 'spoolid' FORWARD SPACED
DMTNPT610I	SETHOLD GETFILE	LINK 'linkid' TO SUSPEND FILE TRANSMISSION
DMTNPT611I	SETHLDIM GETFILE	LINK 'linkid' FILE TRANSMISSION SUSPENDED
DMTNPT612E	SETHLDE1	LINK 'linkid' ALREADY IN HOLD STATUS
DMTNPT750E	SETSTRT1	LINK 'linkid' ALREADY ACTIVE -- NO ACTION TAKEN
DMTNPT752I	SETSTART	LINK 'linkid' STILL ACTIVE -- DRAIN STATUS RESET
DMTNPT801I	SETTR1	LINK 'linkid' ERROR TRACE STARTED
DMTNPT802I	SETTR2	LINK 'linkid' TRACE STARTED
DMTNPT803I	SETTRACE	LINK 'linkid' TRACE ENDED
DMTNPT810E	SETTRE1	LINK 'linkid' TRACE ALREADY ACTIVE
DMTNPT811E	SETTRE2	LINK 'linkid' TRACE NOT ACTIVE
DMTNPT902E	CONFCK1	NON-SIGNON CARD READ ON LINK (linkid)
DMTNPT903E	SPASS	PASSWORD (password) on LINK (linkid) IS INVALID
DMTNPT904E	SGNERR	SIGNON KEYWORD (keyword) INVALID
DMTNPT905I	NPTGETX	SIGNON OF LINK 'linkid' COMPLETE
DMTNPT934E	PUTCLOSE	ID MISSING ON LINK 'linkid' -- INPUT FILE PURGED
DMTNPT936E	GETGOT1	NO REMOTE PUNCH AVAILABLE ON LINK 'linkid' -- FILE 'spoolid' PURGED
DMTREX000I	REXICGOT	RSCS (VER v, LEV l, mm/dd/yy) READY
DMTREX002I	TERLHIT	LINK 'linkid' DEACTIVATED
DMTREX080E	TERLHIT	PROGRAM CHECK -- 'linkid' DEACTIVATED
DMTREX090T	REXPTERM	PROGRAM CHECK IN SUPERVISOR -- RSCS SHUTDOWN
DMTREX091T	REXITERM	INITIALIZATION FAILURE - RSCS SHUTDOWN
DMTSML070E	IOERRPRT	I/O ERROR -- SIOCC -- CSW -- SENSE -- CCW --
DMTSML108E	VMSPGET	SYSTEM ERROR READING SPOOL FILE 'spoolid'
DMTSML141I	ISIO	LINE 'vaddr' READY FOR CONNECTION TO LINK 'linkid'
DMTSML142I	SIGNOK	LINK 'linkid' LINE 'vaddr' CONNECTED
DMTSML143I	EOJ	LINK 'linkid' LINE 'vaddr' DISCONNECTED
DMTSML144I	JOUTPUT PCONT UOUTPUT	RECEIVING: FILE FROM 'locid1' ('name1') FOR 'locid2' ('name2')
DMTSML145I	JCLOSE1 PCLOSE UCLOSE	RECEIVED: FILE FROM 'locid1' ('name1') FOR 'locid2' ('name2')
DMTSML146I	RLOC1	SENDING: FILE 'spoolid' ON LINK 'linkid', REC nnnnnn
DMTSML147I	RDEOF	SENT: FILE 'spoolid' ON LINK 'linkid'
DMTSML149I	TRPRT	LINK 'linkid' LINE ACTIVITY: TOT= mmm; ERRS= nnn; TMOUTS= ppp
DMTSML170I	WGET2	FROM 'linkid': (MSG message text)
DMTSML190E	VMSP1	INVALID SPOOL BLOCK FORMAT ON FILE 'spoolid'
DMTSML510I	RDBKMSG	FILE 'spoolid' BACKSPACED
DMTSML511E	SBKFWDN	NO FILE ACTIVE ON LINK 'linkid'
DMTSML530I	SETCMD	COMMAND FORWARDED ON LINK 'linkid'
DMTSML570I	SETDRAIN $USRNPUN	LINK 'linkid' NOW SET TO DEACTIVATE
DMTSML571E	SETDRER1	LINK 'linkid' ALREADY SET TO DEACTIVATE
DMTSML580I	RDFLUSH	FILE 'spoolid' PROCESSING TERMINATED
DMTSML581E	SETFLUSH RDFLSHER	FILE 'spoolid' NOT ACTIVE
DMTSML590I	SETFREE	LINK 'linkid' RESUMING FILE TRANSFER
DMTSML591E	SETFRER1	LINK 'linkid' NOT IN HOLD STATUS

Message Code	Generated at Label	Message Text
DMTSML600I	RDGODNE	FILE 'spoolid' FORWARD SPACED
DMTSML610I	SETHOLD	LINK 'linkid' TO SUSPEND FILE TRANSMISSION
DMTSML611I	ALLHLD SETHLDIM	LINK 'linkid' FILE TRANSMISSION SUSPENDED
DMTSML612E	SETHLDE1	LINK 'linkid' ALREADY IN HOLD STATUS
DMTSML750E	SETSTRT1	LINK 'linkid' ALREADY ACTIVE -- NO ACTION TAKEN
DMTSML752I	SETSTART	LINK 'linkid' STILL ACTIVE -- DRAIN STATUS RESET
DMTSML801I	SETTR1	LINK 'linkid' ERROR TRACE STARTED
DMTSML802I	SETTR2	LINK 'linkid' TRACE STARTED
DMTSML803I	SETTRACE	LINK 'linkid' TRACE ENDED
DMTSML810E	SETTRE1	LINK 'linkid' TRACE ALREADY ACTIVE
DMTSML811E	SETTRE2	LINK 'linkid' TRACE NOT ACTIVE
DMTSML901E	SMLIERR1	INVALID SML MODE SPECIFIED -- LINK 'linkid' NOT ACTIVATED
DMTSML902E	MC7ERR	NON-SIGNON CARD READ ON LINK (linkid)
DMTSML903E	MC7A	PASSWORD (password) ON LINK (linkid) IS INVALID
DMTSML905I	MC7B	SIGNON OF LINK 'linkid' COMPLETE
DMTSML906E	SMLIERR2	INVALID SML BUFFER PARAMETER -- LINK 'linkid' NOT ACTIVATED
DMTSML934E	JCLOSE	ID CARD MISSING ON LINK 'linkid' -- INPUT FILE PURGED
DMTSML935E	RDNOHLD	LINK 'linkid' IN RJE MODE -- PRINT FILE 'spoolid' PURGED

CMS COMMANDS FOR DEBUGGING

DEBUGGING WITH CMS

This section describes the debug tools that CMS provides. These tools can be used to help you debug CMS or a problem program. In addition, a CMS user can use the CP commands to debug. Information that is often useful in debugging is also included. The following topics are discussed in this section:

- CMS debugging commands
- DASD dump restore program

CMS DEBUGGING COMMANDS

CMS provides two commands that are useful in debugging: DEBUG and SVCTRACE. Both commands execute from the terminal.

The debug environment is entered whenever:

- The DEBUG command is issued
- A breakpoint is reached
- An external or program interruption occurs

CMS does not accept other commands while in the debug environment. However, while in the debug environment, the options of the DEBUG command can:

- Set breakpoints (address stops) that stop program execution at specific locations.
- Display the contents of the CAW (channel address word), CSW (channel status word), old PSW (program status word), or general registers at the terminal.
- Change the contents of the control words (CAW, CSW and PSW) and general registers.
- Dump all or part of virtual storage at the printer.
- Display the contents of up to 56 bytes of virtual storage at the terminal.
- Store data in virtual storage locations.
- Allow an origin or base address to be specified for the program.
- Assign symbolic names to specific storage locations.
- Close all open files and I/O devices and update the master file directory.
- Exit from the debug environment.

The SVCTRACE command records information for all SVC calls. When the trace is terminated, the information recorded up to that point is printed at the system printer.

In addition, several CMS commands produce or print load maps. These load maps can locate storage areas while debugging programs.

DEBUG

The DEBUG command provides support for debugging programs at a terminal. The virtual machine operator can stop the program at a specified location and examine and alter virtual storage, registers, and various control words. Once CMS is in its debug environment, the virtual machine operator can request the various DEBUG options. However, in the debug environment, all of the other CMS commands are considered invalid.

Any DEBUG subcommand may be entered if CMS is in the debug environment and if the keyboard is unlocked. The following rules apply to DEBUG subcommands:

1. No operand should be longer than eight characters. All operands longer than eight characters are left justified and truncated on the right after the eighth character.
2. You must use the DEFINE subcommand to create all entries in the DEBUG symbol table.
3. The DEBUG subcommands can be truncated. The following is a list of all valid DEBUG subcommands and their minimum truncation.

Subcommand	Minimum Truncation
BREAK	BR
CAW	CAW
CSW	CSW
DEFINE	DEF
DUMP	DU
GO	GO
GPR	GPR
HX	HX
ORIGIN	OR
PSW	PSW
RETURN	RET
SET	SET
STORE	ST
X	X

One way to enter the debug environment is to issue the DEBUG command. The message

```
DMSDBG728I DEBUG ENTERED
```

appears at the terminal. Any of the DEBUG subcommands may be entered. To continue normal processing, issue the RETURN subcommand.

Whenever a program check occurs, the DMSABN routine gains control. Issue the DEBUG command at this time if you want CMS to enter its debug environment.

Whenever a breakpoint is encountered, a program check occurs. The message

```
DMSDBG728I DEBUG ENTERED
BREAKPOINT YY AT XXXXX
```

appears on the terminal. Follow the same procedure to enter subcommands and resume processing as with a regular program check.

An external interrupt, which occurs when the CP EXTERNAL command is issued, causes CMS to enter its debug environment. The message

```
DMSDBG728I DEBUG ENTERED
EXTERNAL INTERRUPT
```

appears on the console. Any of the DEBUG subcommands may be issued. To exit from the debug environment after an external interruption, use GO.

While CMS is in its debug environment, the control words and low storage locations contain the debug program values. The debug program saves the control words and low storage contents (X'00' - X'100') of the interrupted routine at location X'C0'.

The following is a detailed discussion of the possible DEBUG subcommands.

BREAK

Use the BREAK subcommand to set breakpoints which stop execution of a program or module at specific instruction locations, called breakpoints. Issuing the BREAK subcommand causes a single breakpoint to be set. A separate BREAK subcommand must be issued for each breakpoint desired. A maximum of 16 breakpoints (with identification numbers 0 through 15) may be in effect at one time; any attempt to set more than 16 breakpoints is rejected. The format of the BREAK subcommand is:

BReak	id {symbol | hexloc}

where:

id is a decimal number, from 0 to 15, which identifies the breakpoint.

symbol
: is a name assigned to the storage location where the breakpoint is set. The symbolic name must be previously assigned to the storage address using the DEF subcommand of the DEBUG command.

hexloc
: is the hexadecimal storage location (relative to the current origin) where the breakpoint is set.

Setting Breakpoints

Breakpoints should be set after a program is loaded, but before it executes. When a breakpoint is encountered during program execution, execution stops and the debug environment is entered. You can then use the other DEBUG subcommands to analyze the program at that particular point. Registers, storage, and control words can be examined and altered. After you finish analyzing the program at this point in its execution, issue the GO subcommand to resume program execution.

Breakpoints are set before the program executes. They are set on instruction (halfword) boundaries at locations that contain operation codes. After setting all the desired breakpoints, issue the RETURN subcommand to exit from the debug environment. Then issue the CMS START command to begin program execution.

The first operand of the BREAK subcommand (id) assigns an identification number (0-15) to the breakpoint. If the identification number specified is the same as a currently set breakpoint, the previous breakpoint is cleared and the new one is set.

The second operand of the BREAK subcommand (symbol or hexloc) indicates the storage location of the breakpoint. If the operand contains any nonhexadecimal characters, the DEBUG symbol table is searched for a matching symbol entry. If a match is found, the breakpoint is set at the storage address corresponding to that symbol, provided that the storage address is on an even (halfword) boundary. If no match is found in the DEBUG symbol table (and the operand is a valid hexadecimal number), the second operand is treated as the hexadecimal representation of the storage address. When the second operand is a valid hexadecimal number, this number is added to the program origin. If the resulting storage address is on a halfword boundary and is not greater than the user's virtual machine's storage size, the breakpoint is set.

How Breakpointing Works

When the debug program sets a breakpoint, it saves the contents of the halfword at the location specified by the second operand of the BREAK subcommand. This halfword is replaced by B2Ex, where x is the hexadecimal equivalent of the identification number, specified in the first operand of the BREAK subcommand. The storage location specified for a breakpoint must contain an operation code. It is your responsibility to see that breakpoints are set only at locations containing operation codes. After breakpoints are set and during program execution, the value B2E0 through B2EF is encountered at a location where an operation code should appear. A program check occurs because all values B2E0 through B2EF are invalid operation codes and control is transferred to the debug environment. DEBUG recognizes the invalid operation code as a breakpoint. The original operation code replaces the invalid operation code, and a message

    DMSDBG728I DEBUG ENTERED
    BREAKPOINT yy AT xxxxxx

appears at the terminal. "yy" is the breakpoint identification number and xxxxxx is the storage address of the breakpoint. After the message is displayed, the keyboard is unlocked to accept any DEBUG subcommands except RETURN. A breakpoint is cleared when it is encountered during program execution.

It is your responsibility to ensure that breakpoints are set only at operation code locations. Otherwise, the breakpoint is not recognized; data or some part of the instruction other than the operation code is overlaid. Thus, errors may be generated if breakpoints are set at locations that do not contain operation codes.

Error Messages

The following error messages may appear while entering the BREAK subcommand.

INVALID OPERAND

This message indicates that the breakpoint identification number specified in the first operand is not a decimal number between 0 and 15 inclusive, or the second operand cannot be located in the DEBUG symbol table and is not a valid hexadecimal number. If the second operand is intended to be a symbol, a DEF subcommand must have been previously issued for that symbol; if not, the operand must be a valid hexadecimal storage location.

INVALID STORAGE REFERENCE

The location indicated by the second operand is uneven (not on a halfword boundary) or the sum of the second operand and the current origin value is greater than the virtual machine's virtual storage size. If the current origin value is unknown, it may be reset to the desired value by issuing the ORIGIN subcommand.

MISSING OPERAND

The minimum number of operands has not been supplied.

TOO MANY OPERANDS

You entered more than two operands.

CAW

Use the CAW subcommand anytime the virtual machine is in the debug environment. Issue the CAW subcommand to check that the command address field contains a valid CCW address, or to find the address of the current CCW so that you can examine it. Issuing the CAW subcommand causes the contents of the CAW (channel address word), as it existed at the time the debug environment was entered, to appear at the terminal. The CAW located at storage location X'48' is saved at the time the debug environment is entered and displayed on the terminal whenever the CAW subcommand is issued. If the subcommand is issued correctly, the contents of the CAW are displayed in hexadecimal representation at the terminal.

The format of the CAW subcommand is:

CAW	

The CAW subcommand has no operands.

The format of the CAW is:

KEY	0000	Command Address
0 3	4 7	8 31

where:

Bits	Contents
0-3	The protection key for all commands associated with START I/O. The protection key in the CAW is compared with a key in storage whenever a reference is made to storage.
4-7	This field is not used and must contain binary zeros.
8-31	The command address field contains the storage address (in hexadecimal representation) of the first CCW (channel command word) associated with the next or most recent START I/O.

The three low-order bits of the command address field must be zeros for the CCW to be on a doubleword boundary. If the CCW is not on a doubleword boundary or if the command address specifies a location protected from fetching or outside the storage of a particular virtual machine, START I/O causes the status portion of the CSW to be stored with the program check or protection check bit on. In this event, the I/O operation is not initiated.

Error Mesages

The following error message may appear while entering the CAW subcommand.

TOO MANY OPERANDS

An operand was entered on the command line; the CAW subcommand has no operands.

CSW

Use the CSW subcommand any time the virtual machine is in the debug environment. Issue the CSW subcommand whenever an I/O operation abnormally terminates. The status and residual count information in the CSW is very useful in debugging. Also, use the CSW to calculate the address of the last executed CCW (subtract 8 bytes from the command address to find the address of the last CCW executed). Issuing the CSW subcommand causes the contents of the CSW (channel status word), as it existed at the time the debug environment was entered, to appear at the terminal. The CSW indicates the status of the channel or an input/output device, or the conditions under which an I/O operation terminated. The CSW is formed in the channel and stored in storage location X'40' when an I/O interrupt occurs. If I/O interruptions are suppressed, the CSW is stored when the next Start I/O, Test I/O, or Halt I/O instruction is executed. The CSW is saved when DEBUG is entered.

If the subcommand is issued correctly, the contents of the CSW are displayed at the terminal in hexadecimal representation.

The format of the CSW subcommand is:

CSW	

The CSW subcommand has no operands.

The format of the CSW is:

KEY	0000	Command Address	Status	Byte Count
0 3	4 7	8 31	32 47	48 63

where:

Bits	Contents
0-3	The protection key is moved to the CSW from the CAW. It indicates the protection key at the time the I/O started. The contents of this field are not affected by programming errors detected by the channel or by the condition causing termination of the operation.
4-7	This field is not used and must contain binary zeros.
8-31	The command address contains a storage address (in hexadecimal representation) eight bytes greater than the address of the last CCW executed.
32-47	The status bits indicate the conditions in the device or channel that caused the CSW to be stored.
48-63	The residual count is the difference between the number of bytes specified in the last executed CCW and the number of bytes that were actually transferred. When an input operation is terminated, the difference between the original count in the CCW and the residual count in the CSW is equal to the number of bytes transferred to storage; on an output operation, the difference is equal to the I/O device.

Error Messages

The following error message may appear when you enter the CSW subcommand.

TOO MANY OPERANDS

An operand was entered on the command line; the CSW subcommand has no operands.

DEFINE

Use the DEFINE subcommand to assign symbolic names to a specific storage address. Once a symbolic name is assigned to a storage address, that symbolic name can refer to that address in any of the other DEBUG subcommands. However, the symbol is valid only in the debug environment.

Issuing the DEFINE subcommand creates an entry in the DEBUG symbol table. The entry consists of the symbol name, the storage address, and the length of the field. A maximum of 16 symbols can be defined in the DEBUG symbol table at a given time.

When a DEFINE subcommand specifies a symbol that already exists in the DEBUG symbol table, the storage address derived from the current request replaces the previous storage address. Several symbols may be assigned to the same storage address, but each of these symbols constitutes one entry in the DEBUG symbol table. The symbols remain defined until a new DEF is issued for them or until an IPL request loads a new copy of CMS.

The format of the DEFINE subcommand is:

```
┌─────────────────────────────────────────────────────┐
│         │                      ┌         ┐          │
│ DEFine  │ symbol   hexloc      │bytecount│          │
│         │                      │    4    │          │
│         │                      └         ┘          │
└─────────────────────────────────────────────────────┘
```

where:

symbol
: is the name to be assigned to the storage address derived from the second operand, hexloc. Symbol may be from 1 to 8 characters long. It must contain at least one nonhexadecimal character. Any symbolic name longer than eight characters is left-justified and truncated on the right after the eighth character.

hexloc
: is the hexadecimal storage location, in relation to the current origin, to which the name specified in the first operand (symbol), is assigned. Hexloc, a hexadecimal number, is added to the current origin established by the ORIGIN subcommand. The sum of the second operand (hexloc) and the origin is the storage address to which the symbolic name is assigned. To assign the symbolic name to the correct location be sure to know the current origin. The existing DEBUG symbol table entries remain unchanged when the ORIGIN subcommand is issued.

bytecount
: is a decimal number, from 1 through 56, which specifies the length in bytes of the field whose name is specifed by the first operand (symbol) and whose starting location is specified by the second operand (hexloc). When the bytecount operand is not specified, a default bytecount of 4 is assumed.

Error Messages

The following error messages may appear when the DEFINE subcommand is issued:

INVALID OPERAND

This message indicates that the name specified in the first operand contains all numeric characters, the second operand is not a valid hexadecimal number, or the third operand is not a decimal number between 1 and 56 inclusive.

INVALID STORAGE ADDRESS

The sum of the second operand and the current origin is greater than the virtual machine's storage size. If the current origin size is unknown, reset it to the desired value by issuing the ORIGIN subcommand and then reissue the DEF subcommand.

16 SYMBOLS ALREADY DEFINED

The DEBUG symbol table is full and no new symbols may be defined until the current definitions are cleared by obtaining a new copy of CMS. However, an existing symbol may be assigned to a new storage location by issuing another DEF subcommand for that symbol.

MISSING OPERAND

The DEFINE subcommand requires at least two operands and less than two were entered.

TOO MANY OPERANDS

Three is the maximum number of operands for the DEFINE subcommand and more than three were entered.

DUMP

Use the DUMP subcommand to print part or all of a virtual machine's storage on the printer.

The format of the DUMP subcommand is:

```
+------------------------------------------------------------+
|        |  [ symbol1 ]  [ symbol2 ]                          |
|  DUmp  |  [ hexloc1 ]  [ hexloc2 ]  [ident]                 |
|        |  [    0    ]  [    *    ]                          |
|        |               [   32    ]                          |
+------------------------------------------------------------+
```

where:

symbol1
is the name assigned (via the DEFINE subcommand) to the storage address that begins the dump.

hexloc1
is the hexadecimal storage location, in relation to the current origin, that begins the dump.

symbol2
is the name assigned (via the DEFINE subcommand) to the storage address that ends the dump.

hexloc2
is the hexadecimal storage location, in relation to the current origin, that ends the dump.

* indicates that the dump ends at the user's last virtual storage address.

ident
is the name (up to eight characters) that identifies this particular printout.

The requested information is printed offline as soon as the printer is available. First, a heading:

ident FROM starting location TO ending location

is printed. Next, the general registers 0 through 7 and 8 through 15, and the floating-point registers 0 through 6 are printed. Then the specified portion of virtual storage is printed with the storage address of the first byte in the line printed at the left, followed by the alphameric interpretation of 32 bytes of storage.

The first and second operands specify the starting and ending addresses, respectively, of the area of storage to be dumped. If you issue DUMP without the first and second operands, 32 bytes of storage are dumped starting at the current origin. If you issue DUMP without the second operand, 32 bytes of storage are dumped starting at the location indicated by the first operand.

If you specify the first and second operands, they must be valid symbols or hexadecimal numbers. When you specify a symbol, the DEBUG symbol table is searched. If a match is found, the storage location corresponding to that symbol is the starting or ending address for the dump. When a hexadecimal number is specified, it is added to the current origin to calculate the starting or ending storage address for the dump. The first and second operands must designate storage addresses that are not greater than the virtual machine's storage size. Also, the storage address derived from the second operand must be greater than the storage address derived from the first operand. An asterisk may be specified for the second operand. In this case, all of storage from the starting address (first operand) to the end of storage is printed on the printer.

Error Messages

The following error messages may appear when you issue the DUMP subcommand.

INVALID OPERAND

This message is issued if the address specified by the second operand is less than that specified by the first operand, or if the first or second operands cannot be located in the DEBUG symbol table and are not valid hexadecimal numbers. If either operand is intended to be a symbol, a DEFINE subcommand must previously have been issued for that symbol; if not, the operand must specify a valid hexadecimal location.

INVALID STORAGE ADDRESS

The hexadecimal number specified in the first or second operand, when added to the current origin, is greater than the user's virtual storage size. If the current origin value is unknown, reset it to the desired value by issuing the ORIGIN subcommand and then reissue the DUMP subcommand.

TOO MANY OPERANDS

Three is the maximum number of operands for the DUMP subcommand; more than three operands were entered.

GO

Use the GO subcommand to exit from the debug environment and begin execution in the CMS environment. The old PSW for the interruption that caused the debug environment to be entered is saved and later loaded to resume processing. Issuing the GO subcommand loads the old PSW.

The format of the GO subcommand is:

```
┌─────────┬───────────────────────────────────────┐
│         │  ┌        ┐                           │
│  GO     │  │ symbol │                           │
│         │  │ hexloc │                           │
│         │  └        ┘                           │
└─────────┴───────────────────────────────────────┘
```

where:

symbol
is the name, already assigned by the DEFINE subcommand, to a storage location where execution begins.

hexloc
is the hexadecimal location, in relation to the current origin, where execution begins.

When the GO subcommand is issued, the general registers, CAW (channel address word), and CSW (channel status word) are restored either to their contents upon entering the debug environment, or, if they have been modified while in the debug environment, to their modified contents. Then the old PSW is loaded and becomes the current PSW. Execution begins at the instruction address contained in bits 40-63 of the PSW.

By specifying an operand with the GO subcommand, you can alter the address where execution is to begin. This operand must be specified whenever the GO subcommand is issued if the debug environment is entered by issuing the DEBUG command.

The operand may be a symbol or a hexadecimal location. When a symbol is specified, the DEBUG symbol table is searched. If a match is found, the storage address corresponding to the symbol replaces the instruction address in the old PSW. When a hexadecimal number is specified, it is added to the current origin to calculate the storage address that replaces the instruction address in the old PSW. In either case, the derived storage address must not be greater than the virtual machine's storage size. Further, it is your responsibility to make sure that the address referred to by the operand of the GO subcommand contains an operation code.

If the debug environment was entered due to a breakpoint, external interruption, or program interruption, then the GO subcommand does not need an operand specifying the starting address.

Error Messages

The following error messages may appear while entering the GO subcommand.

INVALID OPERAND

An operand specified in the GO subcommand cannot be located in the DEBUG symbol table and is not a valid hexadecimal number. If the operand is intended to be a symbol, a DEFINE subcommand must have been previously issued for that symbol; if not, the operand must specify a valid hexadecimal storage location.

INVALID STORAGE ADDRESS

The address at which execution is to begin is not on a halfword boundary (indicating that an operation code is not located at that address) or the sum of the GO operand and the current origin value is greater than the virtual machine's storage size. If the current value is unknown, it may be reset to the desired value by issuing the ORIGIN subcommand.

INCORRECT DEBUG EXIT

The GO subcommand without an operand has been issued when DEBUG had not been entered due to a breakpoint or external interruption. The RETURN subcommand must be issued if DEBUG had been entered via the DEBUG command.

TOO MANY OPERANDS

The GO subcommand has a maximum of one operand; more than one operand was entered.

GPR

Use the GPR subcommand to print the contents of one or more general registers at the terminal.

The format of the GPR subcommand is:

GPR	reg1 [reg2]

where:

reg1
: is a decimal number (from 0 through 15) indicating the first or only general register whose contents are to be typed.

reg2
: is a decimal number (from 0 through 15) indicating the last general register whose contents are to be typed. This operand is optional and is only specified when more than one register's contents are to be printed.

When only one operand is specified, only the contents of that general register are typed at the terminal. When two registers are specified, the contents of all general registers from the register indicated by the first operand through the register indicated by the second operand are typed at the terminal. Both operands must be decimal numbers from 0 through 15 inclusive, and the second operand must be greater than the first.

Error Messages

The following error messages may appear on the terminal when the GPR subcommand is entered.

INVALID OPERAND

The operand(s) specified are not decimal numbers from 0 through 15, or the second operand is less than the first.

MISSING OPERAND

The GPR subommand requires at least one operand, and none was entered.

TOO MANY OPERANDS

The GPR subcommand has a maximum of two operands, and more than two operands were entered.

HX

Use the HX subcommand to close all open files and I/O devices, and to update the master file directory. This subcommand may be issued whenever the keyboard is unlocked in the debug environment, regardless of the reason the debug environment was entered.

The format of the HX subcommand is:

HX	

The HX subcommand has no operands.

Error Messages

The following error message may appear on the terminal while entering the HX subcommand.

TOO MANY OPERANDS

The HX subcommand has no operands, and one or more operands were entered.

ORIGIN

Use the ORIGIN subcommand to set the origin equal to the program load point. The ORIGIN subcommand sets an origin or base address to be used in the debug environment. In all debug subcommands, you can specify instruction addresses in relation to the program load point, rather than to 0. The hexadecimal location specified in DEBUG subcommands then represents a specific location within a program, the origin represents the storage location of the beginning of the program; and the two values added together represent the actual storage location of that specific point in the program.

The format of the ORIGIN subcommand is:

```
| ORigin | {symbol}  |
|        | {hexloc}  |
```

where:

symbol
is a name that was previously assigned (via the DEFINE subcommand) to a storage address.

hexloc
is a hexadecimal location within the limits of the virtual machine's storage.

When the ORIGIN subcommand specifies a symbol, the DEBUG symbol table is searched. If a match is found, the value corresponding to the symbol becomes the new origin. When a hexadecimal location is specified, that value becomes the origin. In either case, the operand cannot specify an address greater than the virtual machine's storage size.

Any origin set by an remains in effect until another ORIGIN subcommand ORIGIN subcommand is issued, or until you obtain a new copy of CMS. Whenever a new ORIGIN subcommand is issued, the value specified in that subcommand overlays the previous origin setting. If you obtain a new copy of CMS (via IPL), the origin is set to 0 until a new ORIGIN subcommand is issued.

Error Messages

The following error messages may appear while you enter the ORIGIN subcommand.

INVALID OPERAND

The operand specified in the ORIGIN subcommand cannot be located in the DEBUG symbol table and is not a valid hexadecimal number. If the operand is intended to be a symbol, a DEFINE subcommand must have been previously issued for that symbol; if not, the operand must specify a valid hexadecimal location.

INVALID STORAGE ADDRESS

The address specified by the ORIGIN operand is greater than the user's virtual storage size.

MISSING OPERAND

The ORIGIN subcommand requires one operand, and none was entered.

TOO MANY OPERANDS

The ORIGIN subcommand requires only one operand, and more than one was entered.

PSW

Use the PSW subcommand to type the contents of the old PSW (program status word) for the interruption that caused DEBUG to be entered.

The format of the PSW subcommand is:

```
┌──────────────────────────────────────────────────┐
| PSW  |                                           |
└──────────────────────────────────────────────────┘
```

The PSW subcommand has no operands.

If DEBUG was entered due to an external interrupt, the PSW subcommand causes the contents of the external old PSW to be typed at the terminal. If a program interrupt caused DEBUG to be entered, the contents of the program old PSW are typed. If DEBUG was entered for any other reason, the following is typed in response to the PSW subcommand:

01000000 xxxxxxxx

where the 1 in the first byte means that external interruptions are allowed and xxxxxxxx is the hexadecimal storage address of the DEBUG program.

The PSW contains some information not contained in storage or registers but required for proper program execution. In general, the PSW controls instruction sequencing and holds and indicates the status of the system in relation to the program currently executing.

Error Messages

The following error message may appear while entering the PSW subcommand.

TOO MANY OPERANDS

The PSW subcommand has no operands and one or more was entered.

RETURN

Use the RETURN subcommand to exit from the debug environment to the CMS command environment. RETURN should be used only when DEBUG is entered by issuing the DEBUG command.

The format of the RETURN subcommand is:

RETurn	

The RETURN subcommand has no operands.

When RETURN is issued, the information contained in the general registers at the time DEBUG was entered is restored or, if this information was changed while in the debug environment, the changed information is restored. In either case, register 15, the error code register, is set to zero. A branch is then made to the address contained in register 14, the normal CMS return register. If DEBUG is entered by issuing the DEBUG command, register 14 contains the address of a central CMS service routine and control transfers directly to the CMS command environment. The ready message followed by a carriage return and an unlocked keyboard indicates that the RETURN subcommand has successfully executed and that control has transferred from the DEBUG environment to the CMS command environment.

Error Messages

The following error messages may appear while entering the RETURN subcommand.

TOO MANY OPERANDS

The RETURN subcommand has no operands, and one or more were specified.

INCORRECT DEBUG EXIT

If DEBUG is entered due to a program or external interruption, a breakpoint or an unrecoverable error, this message is displayed in response to the RETURN subcommand. To exit from the DEBUG environment under the above circumstances, issue GO.

SET

Use the SET subcommand to change the contents of the control words and general registers that are saved when the debug environment is entered. The contents of these registers are restored when control transfers from DEBUG to another environment. If register contents were modified in DEBUG, the changed contents are stored.

The format of the SET subcommand is:

```
| SET | CAW  hexinfo                          |
|     | CSW  hexinfo  [hexinfo]               |
|     | PSW  hexinfo  [hexinfo]               |
|     | GPR  reg       hexinfo   [hexinfo]    |
```

where:

CAW hexinfo
: indicates that the specified information (hexinfo) is stored in the CAW (channel address word) that existed at the time DEBUG was entered.

CSW hexinfo [hexinfo]
: indicates that the specified information (hexinfo [hexinfo]) is stored in the CSW (channel status word) that existed at the time DEBUG was entered.

PSW hexinfo [hexinfo]
: indicates that the specified information (hexinfo [hexinfo]) is stored in old PSW (program status word) for the interruption that caused DEBUG to be entered.

GPR reg hexinfo [hexinfo]
: indicates that the specified information (hexinfo [hexinfo]) is stored in the specified general register (reg).

Each hexinfo operand should be from one to four bytes long. If an operand is less than four bytes and contains an uneven number of hexadecimal digits (representing half-byte information), the information is right-justified and the left half of the uneven byte is set to zero. If more than eight hexadecimal digits are specified in a single operand, the information is left-justified and truncated on the right after the eighth digit.

The SET subcommand can only change the contents of one control word at a time. For example, the SET subcommand must be issued three times:

```
SET  CAW  hexinfo
SET  CSW  hexinfo [hexinfo]
SET  PSW  hexinfo [hexinfo]
```

to change the contents of the three control words.

The SET subcommand can change the contents of one or two general registers each time it is issued. When four or less bytes of information are specified, only the contents of the specified register are changed. When more than four bytes of information is specified, the contents of the specified register and the next sequential register are changed. For example, the SET subcommand:

```
SET  GPR  2 xxxxxxxx
```

changes only the contents of general register 2. But, the SET subcommand:

```
SET  GPR  2 xxxxxxxx xxxxxxxx
```

changes the contents of general registers 2 and 3.

The number of bytes that can be stored using the SET subcommand varies depending on the form of the subcommand. With the CAW form, up to four bytes of information may be stored. With the CSW, GPR, and PSW forms, up to eight bytes of information may be stored, but these bytes must be represented in two operands of four bytes each. When two operands of information are specified, the information is stored in consecutive locations (or registers), even if one or both operands contain less than four bytes of information.

The contents of registers changed using the SET subcommand are not displayed after the subcommand is issued. To inspect the contents of control words and registers, the CAW, CSW, PSW, or GPR subcommands must be issued.

Error Messages

The following error messages may appear while entering the SET subcommand.

INVALID OPERAND

The first operand is not CAW, CSW, PSW, or GPR, or the first operand is GPR and the second operand is not a decimal number between 0 and 15 inclusive, or one or more of the hexinfo operands does not contain hexadecimal information.

MISSING OPERAND

The minimum number of operands has not been entered.

TOO MANY OPERANDS

More than the required number of operands were specified.

STORE

Use the STORE subcommand to store up to 12 bytes of hexadecimal information in any valid virtual storage address. The information is stored starting in the location derived from the first operand (symbol or hexloc).

The format of the STORE subcommand is:

```
┌─────────────────────────────────────────────────┐
│ STore │ {symbol} hexinfo [hexinfo [hexinfo]] │
│       │ {hexloc}                              │
└─────────────────────────────────────────────────┘
```

where:

symbol
: is the name assigned (via the DEFINE subcommand) to the storage address where the first byte of specified information is stored.

hexloc
: is the hexadecimal location, relative to the current origin, where the first byte of information is stored.

hexinfo
: is any hexadecimal information, four bytes or less in length, to be stored.

If the first operand contains any nonhexadecimal characters, the DEBUG symbol table is searched for a matching symbol entry. If a match is found in the DEBUG symbol table, or if the first operand contains only hexadecimal characters, the current origin is added to the specified operand and the resulting storage address is used, provided it is not greater than the virtual machine's storage size.

The information to be stored is specified in hexadecimal format in the second through the fourth operands. Each of these operands is from one to four bytes (that is, two to eight hexadeciaml digits) long. If an operand is less than four bytes long and contains an uneven number of hexadecimal digits (representing half-byte information), the information is right-justified and the left half of the uneven byte is set to zero. If more than eight hexadecimal digits are specified in a single operand, the information is left-justified and truncated on the right after the eighth digit.

The STORE subcommand can store a maximum of 12 bytes at one time. By specifying all three information operands, each containing four bytes of information, the maximum 12 bytes can be stored. If less than four bytes are specified in any or all of the operands, the information given is arranged into a string of consecutive bytes, and that string is stored starting at the location derived from the first operand. Stored information is not typed at the terminal. To inspect the changed contents of storage after a STORE subcommand, issue an X subcommand.

Error Messages

The following errcr messages may appear on the terminal while entering the STORE subcommand.

INVALID OPERAND

The first operand cannot be located in the DEBUG symbol table and is not a valid hexadecimal number, or the information specified in the second, third, or fourth operands is not in hexadecimal format. If the first operand is intended to be a symbol, a DEFINE subcommand must have been previously issued for that symbol; if not, the operand must specify a valid hexadecimal storage location.

INVALID STORAGE ADDRESS

The current origin value, when added to the hexadecimal number specified as the first operand, gives an address greater than the user's virtual storage size. If the origin value is unknown, reset it to the desired value using the ORIGIN subcommand and reissue the STORE subcommand.

MISSING OPERAND

Less than two operands were specified.

TOO MANY OPERANDS

More than four operands were specified.

X

Use the X subcommand to examine and display the contents of specific locations in virtual storage. The information is displayed at the terminal in hexadecimal format.

The format of the X (examine) subcommand is:

```
┌──────────────────────────────────────────────┐
│   │ / symbol  ┌        ┐ \                    │
│ X │ |         │ n      │ |                    │
│   │ |         │ length │ |                    │
│   │ <         └        ┘ >                    │
│   │ | hexloc  ┌        ┐ |                    │
│   │ |         │ n      │ |                    │
│   │ \         │ 4      │ /                    │
│   │           └        ┘                      │
└──────────────────────────────────────────────┘
```

where:

symbol
: is the name assigned (via the DEFINE subcommand) to the storage address of the first byte to be examined.

hexloc
: is the hexadecimal location, in relation to the current origin, of the first byte to be examined.

n
: is a decimal number from 1 through 56 that specifies the number of bytes to be examined. If a symbol is specified without a second operand, the length attribute associated with that symbol in the DEBUG symbol table specifies the number of bytes to be examined. If a hexadecimal location is specified without a second operand, four bytes are examined.

The first operand of the subcommand specifies the beginning address of the portion of storage to be examined. If the operand contains any nonhexadecimal characters, the DEBUG symbol table is searched for a matching symbol entry. If a match is found, the storage address to which that symbol refers is the location of the first byte to be examined. If no match is found, or if the first operand contains only hexadecimal characters, the current origin as established by the ORIGIN subcommand is added to the specified operand and the resulting storage address is the location of the first byte to be examined. The derived address must not be greater than the virtual machine's storage size.

The second operand of the X subcommand is optional. If specified, it indicates the number of bytes (up to a maximum of 56) whose contents are to be displayed. If the second operand is omitted and the first operand is a hexadecimal location, a default value of four bytes is assumed. If the second operand is omitted and the first operand is a symbol, the length attribute associated with that symbol in the DEBUG symbol table is used as the number of bytes to be displayed.

Error Messages

The following error messages may appear on the terminal when the X subcommand is entered.

INVALID OPERAND

The first operand cannot be located in the DEBUG symbol table and is not a valid hexadecimal number, or the second operand is not a decimal number from 1 through 56. If the first operand is intended to be a symbol, it must have been defined in a previous DEFINE subcommand; otherwise, the operand must specify a valid hexadecimal number.

INVALID STORAGE ADDRESS

The hexadecimal number specified in the first operand, when added to the current origin, is greater than the storage size of the machine being used. If the current origin value is unknown, reset it to the desired value by issuing the ORIGIN subcommand and reissue the X subcommand.

MISSING OPERAND

No operands were entered; at least one is required.

TOO MANY OPERANDS

More than the maximum of two operands were entered.

SVCTRACE

Use the SVCTRACE command to trace internal transfers of information resulting from SVC (supervisor call) instructions. Issuing the SVCTRACE command causes switches to be set. These switches, in turn, cause information to be recorded at appropriate times. When the trace is terminated, the recorded information is printed at the system printer.

The information recorded for a normal SVC call is:

- Storage address of the SVC calling instruction
- Name of the program being called
- Contents of the SVC old PSW
- Storage address of the return from the called program
- The general registers and floating-point registers
- The parameter list at the time the SVC is issued.

The format of the SVCTRACE command is:

```
+----------------------------------------------------+
|SVCTrace  |  {ON }                                  |
|          |  {OFF}                                  |
+----------------------------------------------------+
```

where:

ON indicates tracing for all SVC calls.

OFF discontinues all SVC tracing.

The trace information is:

- The general registers both before the SVC-called program is given control and after a return from that program.
- The floating-point registers both before the SVC-called program is given control and after a return from that program.
- The parameter list, as it existed when the SVC was issued.

To terminate tracing set by the SVCTRACE command, issue the HO or SVCTRACE OFF command. Both SVCTRACE OFF and HO cause all trace information recorded up to the point they are issued to be printed at the system printer. SVCTRACE OFF can be issued only when the keyboard is unlocked to accept input to the CMS command environment. To terminate tracing at any other point in system processing, HO must be issued. If a HX subcommand to the DEBUG environment or a logout from the control program is issued before terminating SVCTRACE, the switches are cleared automatically and all recorded trace information is printed at the system printer.

Interpreting the Output

A variety of information is printed whenever the

    SVCTRACE ON

command is issued.

The first line of trace output starts with a minus sign (-), a plus sign (+), or an asterisk (*). The format of the first line of trace output is:

```
{ - }
{ + } N/D = xxx/dd name FROM loc OLDPSW = psw1
{ * } GOPSW = psw2 [RC = rc]
```

where:

- indicates information recorded before processing the SVC.

+ indicates information recorded after processing the SVC, unless * applies.

* indicates information recorded after processing a CMS SVC which had an error return.

N/D is an abbreviation for SVC Number and Depth (or level).

xxx is the number of the SVC call (they are numbered sequentially).

dd is the nesting level of the SVC call.

name is the macro or routine being called.

loc is the program location from which the SVC was issued.

psw1 is the PSW at the time the SVC was called.

psw2 the PSW with which the routine (for example, RDBUF) being called is invoked, if the first character of this line is a minus sign (-). If the first character of this line is a plus sign (+) or asterisk (*), PSW2 represents the PSW which returns control to the user.

rc is the return code passed from the SVC handling routine in general register 15. This field is omitted if the first character of this line is a minus sign (-), or if this is an OS SVC call. For a CMS SVC, this field is zero if the line begins with a plus sign (+), and nonzero for an asterisk (*). Also, this field equals the contents of Register 15 in the "GPRS AFTER" line.

The next two lines of output are the contents of the general registers when control is passed to the SVC handling routine. This output is identified at the left by "•GPRSB". The format of the output is:

```
•GPRSB = h h h h h h h h *dddddddd*
       = h h h h h h h h *dddddddd*
```

where:

h represents the contents of a general register in hexadecimal format.

d represents the EBCDIC translation of the contents of a general register.

The contents of general registers 0-7 are printed on the first line, with the contents of 8-F on the second line. The hexadecimal contents of the registers registers are printed first, following by the EBCDIC. The EBCDIC translation translation is preceded and followed by an asterisk (*).

The next line of output is the contents of general registers 0, 1 and 15 when control is returned to the user's program. The output is identified at the left by "•GPRS AFTER :". The format of the output is:

```
•GPRS AFTER : R0-R1 = h h *dd* R15 = h *d*
```

where:

h represents the hexadecimal contents of a general register.

d is the EBCDIC translation of the contents of a general register.

The only general registers that CMS routines alter are registers 0, 1, and 15 so only those registers are printed when control returns to the user program. The EBCDIC translation is preceded and followed by an asterisk (*).

The next two lines of output are the contents of the general registers when the SVC handling routine is finished processing. This output is identified at the left by "•GPRSS". The format of the output is:

```
•GPRSS = h h h h h h h h *dddddddd*
       = h h h h h h h h *dddddddd*
```

where:

h represents the hexadecimal contents of a general register.

d represents the EBCDIC translation of the contents of a general register.

General registers 0-7 are printed on the first line with registers 8-F on the second line. The EBCDIC translation is preceded and followed by an asterisk (*).

The next line of output is the contents of the caller's floating-point registers. The output is identified at the left by "•FPRS." The format of the output is:

```
•FPRS = f f f f *gggg*
```

where:

f represents the hexadecimal contents of a floating-point register.

g is the EBCDIC translation of a floating-point register.

Each floating-point register is a doubleword: each f and g represents a doubleword of data. The EBCDIC translation is preceded and followed by an asterisk (*).

The next line of output is the contents of floating-point registers when the SVC-handling routine is finished processing. The output is identified by "•FPRSS" at the left. The format of the output is:

```
•FPRSS = f f f f *gggg*
```

where:

f represents the hexadecimal contents of a floating-point register.

g is the EBDCIC translation.

Each floating-point register is a doubleword and each f and g represents a doubleword of data. The EBCDIC translation is preceded and followed by an asterisk (*).

The last two lines of output are only printed if the address in Register 1 is a valid address for the virtual machine. If printed, the output is the parameter list passed to the SVC. The output is identified by "•PARM" at the left. The output format is:

```
•PARM = h h h h h h h h *dddddddd*
      = h h h h h h h h *dddddddd*
```

where:

h represents a word of hexadecimal data

d is the EBCDIC translation.

The parameter list is found at the address contained in register 1 before control is passed to the SVC-handling program. The EBCDIC translation is preceded and followed by an asterisk (*).

Figure 66 summarizes the types of SVC trace output.

Identification	Comments
{ + - * } N/D	The SVC and the routine which issued the SVC.
•GPRSB	Contents of general registers when control passed to the SVC handling routine.
•GPRS AFTER	Contents of general registers 0, 1, and 15 when control is returned to the user program.
•GPRSS	Contents of the general registers when the SVC handling routine is finished processing.
•FPRS	Contents of floating-point register before the SVC-called program is given control and after returning from that program.
•FPRSS	Contents of the floating-point registers when the SVC handling routine is finished processing.
•PARM	The parameter list, when one is passed to the SVC.

Figure 66. Summary of SVC Trace Output Lines

DASD DUMP RESTORE SERVICE PROGRAM AND HOW TO USE IT

Use the DASD Dump Restore (DDR) service program to dump, restore, copy, display, or print VM/370 user minidisks. The DDR program may run as a standalone program, or under CMS via the DDR command.

INVOKING DDR UNDER CMS

The format of the DDR command is:

DDR	[filename [filetype {filemode | *}]]

where:

filename filetype [filemode]
is the identification of the file containing the control statements for the DDR program. If no file identification is provided, the DDR program attempts to obtain control statements from the console. The filemode defaults to * if a value is not provided.

INVOKING DDR AS A STANDALONE PROGRAM

To use DDR as a standalone program, load it from a real or virtual IPL device as you would any other standalone program. Then indicate where the DDR program is to obtain its control statements by responding to prompting messages at the console.

See the "DDR Control Statements" discussion in the "CP Commands for Debugging" section. The control statements for running standalone and under CMS are identical, except that CMS ignores the SYSPRINT control statement.

SECTION 5. APPENDIXES

Section 5 has nine appendixes:

- "Appendix A: VM/370 Coding Conventions"
- "Appendix B: CP and RSCS Equate Symbols"
- "Appendix C: CMS Equate Symbols"
- "Appendix D: DASD Record Formats"
- "Appendix E: VM/370 Restrictions"
- "Appendix F: Virtual Devices Used in CMS"
- "Appendix G: Function Codes for DIAGNOSE Instructions"
- "Appendix H: CMS ZAP Service Program"
- "Appendix I: Applying PTFs"

CP CODING CONVENTIONS

The following are coding conventions used by CP modules. This information should prove helpful if you debug, modify, or update CP.

- FORMAT

Column	Contents
1	Labels
10	Operation Code
16	Operands
31, 36, 41, etc.	Comments

- COMMENT

  Approximately 75 percent of the source code contains comments. Sections of code performing distinctly separate functions are separated from each other by a comment section.

- CONSTANTS

  Constants follow the executable code and precede the copy files and/or macros that contain DSECTs or system equates. Constants are defined in a section followed by a section containing initialized working storage, followed by working storage. Each of these sections is identified by a comment. Wherever possible for a module that is greater than a page, constants and working storage are within the same page in which they are referenced.

- No program modifies its own instructions during execution.

- No program uses its own unlabeled instructions as data.

- REGISTER USAGE

  - For CP:

Register	Use
6	RCHBLOK, VCHBLOK
7	RCUBLOK, VCUBLOK
8	RDEVBLOK, VDEVBLOK
10	IOBLOK
11	VMBLOK
12	Base register for modules called via SVC
13	SAVEAREA for modules called via SVC
14	Return linkage for modules called via BALR
15	Base address for modules called via BALR

  - For Virtual-to-Real address translation:

Register	Use
1	Virtual address
2	Real address

- When describing an area of storage in mainline code, a copy file, or a macro, DSECT is issued containing DS instructions.

- Meaningful names are used instead of self-defining terms, for example 5, X'02', or C'I' represent a quantity (absolute address, offset, length, register, etc.). All labels, displacements, and values are symbolic. All bits should be symbolic and defined by EQU. For example:

```
VMSTATUS  EQU  X'02'
```

  - To set a bit, use:

```
OI   BYTE,BIT
```

  where BYTE = name of field, BIT is an EQU symbol.

  - To reset a bit, use:

```
NI   BYTE,255-BIT
```

  - To set multiple bits, use:

```
OI   BYTE,BIT1+BIT2
```

  - All registers are referred to as:

```
R0, R1, ..., R15
```

  - All lengths of fields or blocks are symbolic, that is, length of VMBLOK is:

```
VMBLOKSZ  EQU  *-VMBLOK
```

- Avoid absolute relative addressing in branches and data references, (that is, location counter value (*) or symbolic label plus or minus a self-defining term used to form a displacement).

- When using a single operation to reference multiple values, specify each value referenced, for example:

```
LM  R2,R4,CONT  SET R2=CON1
                SET R3=CON2
                SET R4=CON3
.
.
.
CON1  DC  F'1'
CON2  DC  F'2'
CON3  DC  F'3'
```

- Do not use PRINT NOGEN or PRINT OFF in source code.

- MODULE NAMES

  Control section names and external references are as follows:

  - The first three letters of the name are the assigned component code.

    Example: DMK

  - The next three letters of the module name identify the module and must be unique.

    Example: DSP

  - The preceding three-letter, unique module identifier is the label of the TITLE card.

    Each entry point or external reference must be prefixed by the six-letter unique identifier of the module.

    Example: DMKDSPCH

- TITLE Card Example:

  DSP TITLE 'DMKDSP VM/370 DISPATCHER VERSION v LEVEL 1'

- PTF Card Example:

  CP/CMS: PUNCH 'xxxxxxxx APPLIED'

  where xxxxxxxx = APAR number response

- ERROR MESSAGES

  There should not be any insertions into the message at execution time and the length of the message should be resolved by the assembler. If insertions must be made, the message must be assembled as different DC statements, and the insert positions are to be individually labeled.

- For all RX instructions use ',' to specify the base register when indexing is not being used, that is:

```
L  R2,AB(,R4)
```

- To determine if your program is executing in a virtual machine or a real machine, issue the Store CPU ID (STIDP) instruction. If STIDP is issued from a virtual machine, the version number (the first byte of the CPUID field) returned will be X'FF'.

CP LOADLIST REQUIREMENTS

The CP loadlist EXEC contains a list of CP modules used by the VMFLOAD procedures when punching the text decks that make up the CP system. All modules following DMKCPE in the list are pageable CP modules. Each 4K page in this area may contain one or more modules. The module grouping governs the order in which they appear in the loadlist. An SPB[1] (Set Page Boundary) card is a loader control card which forces the loader to start this module at the next higher 4K boundary. An SPB card is required only for the first module following DMKCPE. If more than one module is to be contained in a 4K page, only the first can be assembled with an SPB card. The second and subsequent modules for a multiple module 4K page must not contain SPB cards.

If changes are made to the loadlist, care must be taken to ensure that any modules loaded together in the pageable area do not exceed the 4K limit. Page boundary crossover is not allowed in the pageable CP modules.

The position of two modules in the loadlist is critical. All modules following DMKCPE must be reenterable and must not contain any address constants referring to anything in the pageable CP area. DMKCKP must be the last module in the loadlist.

----

[1]A 12-2-9 multipunch must be in column 1 of an SPB card.

APPENDIX B: CP AND RSCS EQUATE SYMBOLS

This appendix contains assembler language equate symbols that reference CP and RSCS data for:

- VM/370 Device Classes, Types, Models and Features
- VM/370 Machine Usage
- VM/370 Extended Control Registers
- VM/370 CP Usage
- VM/370 Registers

VM/370 DEVICE CLASSES, TYPES, MODELS AND FEATURES

```
CLASTERM EQU   X'80'           Teriminal Device Class
TYP2700  EQU   X'40'           2700 Bisync Line
TYP2955  EQU   TYP2700         2955 Communications Line
TYPTELE2 EQU   X'20'           Telegraph Terminal Control Type II
TYPTTY   EQU   X'20'           TELETYPE Terminal
TYPIBM1  EQU   X'10'           IBM Terminal Control Type I
TYP2741  EQU   X'18'           2741 Communications Terminal
TYP1050  EQU   X'14'           1050 Communications Terminal
TYPUNDEF EQU   X'1C'           Terminal device type is undefined
TYPBSC   EQU   X'80'           Bisync Line for 3270 Remote Stations
TYP3210  EQU   X'00'           3210 Console
TYP3215  EQU   TYP3210         3215 Console
TYP2150  EQU   TYP3210         2150 Console
TYP1052  EQU   TYP3210         1052 Console
CLASGRAF EQU   X'40'           Graphics Device Class
TYP2250  EQU   X'80'           2250 Display Unit
TYP2260  EQU   X'40'           2260 Display Station
TYP2265  EQU   X'20'           2265 Display Station
TYP3066  EQU   X'10'           3066 Console
TYP1053  EQU   X'08'           1053 Printer
TYP3277  EQU   X'04'           3277 Display Station
TYP3284  EQU   X'02'           3284 Printer
TYP3286  EQU   TYP3284         3286 Printer
TYP3158  EQU   TYP3277         3158 Console
FTROPRDR EQU   X'80'           Operator ID Card Reader
CLASURI  EQU   X'20'           Unit Record Input Device Class
TYPRDR   EQU   X'80'           Card Reader Device
TYP2501  EQU   X'81'           2501 Card Reader
TYP2540R EQU   X'82'           2540 Card Reader
TYP3505  EQU   X'84'           3505 Card Reader
TYP1442R EQU   X'88'           1442 Card Reader/Punch
TYP2520R EQU   X'90'           2520 Card Reader/Punch
TYPTIMER EQU   X'40'           Timer Device
TYPTR    EQU   X'20'           Tape Reader Device
TYP2495  EQU   X'21'           2495 Magnetic Tape Cartridge Reader
TYP2671  EQU   X'22'           2671 Paper Tape Reader
TYP1017  EQU   X'24'           1017 Paper Tape Reader
CLASURO  EQU   X'10'           Unit Record Output Device Class
TYPPUN   EQU   X'80'           Card Punch Device
TYP2540P EQU   X'82'           2540 Card Punch
TYP3525  EQU   X'84'           3525 Card Punch
TYP1442P EQU   X'88'           1442 Card Punch
TYP2520P EQU   X'90'           2520 Card Punch
TYPPRT   EQU   X'40'           Printer Type Device
TYP1403  EQU   X'41'           1403 Printer
TYP3211  EQU   X'42'           3211 Printer
TYP1443  EQU   X'44'           1443 Printer
TYPTP    EQU   X'20'           Tape Punch Device
TYP1018  EQU   X'24'           1018 Paper Tape Punch
FTRUCS   EQU   X'01'           UCS Feature
CLASTAPE EQU   X'08'           Magnetic Tape Device Class
TYP2401  EQU   X'80'           2401 Tape Drive
TYP2415  EQU   X'40'           2415 Tape Drive
TYP2420  EQU   X'20'           2420 Tape Drive
TYP3420  EQU   X'10'           3420 Tape Drive
TYP3410  EQU   X'08'           3410 Tape Drive
TYP3411  EQU   TYP3410         3411 Tape Drive
FTR7TRK  EQU   X'80'           7-track Feature
FTRDLDNS EQU   X'40'           Dual Density Feature
FTRTRANS EQU   X'20'           Translate Feature
FTRDCONV EQU   X'10'           Data Conversion Feature
CLASDASD EQU   X'04'           Direct Access Storage Device Class
TYP2311  EQU   X'80'           2311 Disk Storage Drive
TYP2314  EQU   X'40'           2314 Disk Storage Facility
TYP2319  EQU   TYP2314         2319 Disk Storage Facility
```

VM/370 Device Classes, Types, Models and Features (continued)

```
TYP2321  EQU   TYP2311          2321 Data Cell Drive
TYP3330  EQU   X'10'            3330 Disk Storage Facility
TYP3333  EQU   TYP3330          3333 Disk Storage Facility
TYP3350  EQU   X'08'            3350 Disk Storage Facility
TYP2301  EQU   TYP2311          3201 Parallel Drum
TYP2303  EQU   TYP2311          2303 Serial Drum
TYP2305  EQU   X'02'            2305 Fixed Head Storage Device
TYP3340  EQU   X'01'            3340 Disk Storage Facility
FTRRPS   EQU   X'80'            Rotational Positional Sensing (RPS)
                                  Installed (3340)
FTREXTSN EQU   X'40'            Extended Sense Bytes (24 bytes)
FTR2311T EQU   X'20'  (= VDEV231T)  Top half of 2314 used as 2311
FTR2311B EQU   X'10'  (= VDEV231B)  Bottom half of 2314 used as 2311
FTR35MB  EQU   X'08'            35 MB Data Module mounted (3340)
FTR70MB  EQU   X'04'            70 MB Data Module mounted (3340)
FTRRSRL  EQU   X'02'            RESERVE/RELEASE are valid CCW op codes
CLASSPEC EQU   X'02'            Special device class
TYPCTCA  EQU   X'80'            Channel-to-channel adapter
TYP3704  EQU   X'40'            3704 Programmable Communication Control Unit
TYP3705  EQU   TYP3704          3705 Programmable Communications Control Unit
TYPRSV1  EQU   X'02'            Reserved by IBM
TYPUNSUP EQU   X'01'            Device unsupported by VM/370
FTRTYP1  EQU   X'10'            Type 1 Channel Adapter (3704/3705)
FTRTYP2  EQU   X'20'            Type 2 Channel Adapter (3704/3705)
```

Bits Defined in Standard/Extended PSW

```
EXTMODE  EQU   X'08'          Bit 12 - Extended mode
MCHEK    EQU   X'04'          Bit 13 - Machine check enabled
WAIT     EQU   X'02'          Bit 14 - Wait state
PROBMODE EQU   X'01'          Bit 15 - Problem state
```

Bits Defined in Extended PSW

```
PERMODE  EQU   X'40'          Bit 01 - PER enabled
TRANMODE EQU   X'04'          Bit 05 - Translate mode
IOMASK   EQU   X'02'          Bit 06 - Summary I/O mask
EXTMASK  EQU   X'01'          Bit 07 - Summary external mask
```

Bits Defined in Channel Status Word - CSW

```
ATTN     EQU   X'80'          Bit 32 - Attention
SM       EQU   X'40'          Bit 33 - Status modifier
CUE      EQU   X'20'          Bit 34 - Control unit end
BUSY     EQU   X'10'          Bit 35 - Busy
CE       EQU   X'08'          Bit 36 - Channel end
DE       EQU   X'04'          Bit 37 - Device end
UC       EQU   X'02'          Bit 38 - Unit check
UE       EQU   X'01'          Bit 39 - Unit exception
PCI      EQU   X'80'          Bit 40 - Program-control interruption
IL       EQU   X'40'          Bit 41 - Incorrect length
PRGC     EQU   X'20'          Bit 42 - Program check
PRTC     EQU   X'10'          Bit 43 - Protection check
CDC      EQU   X'08'          Bit 44 - Channel data check
CCC      EQU   X'04'          Bit 45 - Channel control check
IFCC     EQU   X'02'          Bit 46 - Interface control check
CHC      EQU   X'01'          Bit 47 - Chaining check
```

Bits Defined in Channel Command Word - CCW

```
CD       EQU   X'80'          Bit 32 - Chain data
CC       EQU   X'40'          Bit 33 - Command chain
SILI     EQU   X'20'          Bit 34 - Suppress incorrect length indicator
SKIP     EQU   X'10'          Bit 35 - Suppress Data Transfer
PCIF     EQU   X'08'          Bit 36 - Program-control interruption fetch
IDA      EQU   X'04'          Bit 37 - Indirect data address
```

Bits Defined in Sense Byte 0 -- Common to Most Devices

```
CMDREJ   EQU   X'80'          Bit 0 - Command reject
INTREQ   EQU   X'40'          Bit 1 - Intervention required
BUSOUT   EQU   X'20'          Bit 2 - Bus out
EQCHK    EQU   X'10'          Bit 3 - Equipment check
DATACHK  EQU   X'08'          Bit 4 - Data check
```

VM/370 EXTENDED CONTROL REGISTERS

Bits Defined in CREG 0

```
                 BYTE 0
BLKMPX    EQU    X'80'          Bit 00 - Enable block multiplexing
SSMSUPP   EQU    X'40'          Bit 01 - Enable SSM suppression

                 BYTE 1
PAGE4K    EQU    X'80'          Bit 08 - Use 4K pages
PAGE2K    EQU    X'40'          Bit 09 - Use 2K pages
SEG1M     EQU    X'10'          Bit 11 - Use 1M segments

                 BYTE 2
CKCMASK   EQU    X'08'          Bit 20 - Mask on clock comparator interruption
CPTMASK   EQU    X'04'          Bit 21 - Mask on CPU timer interruption

                 BYTE 3
INTMASK   EQU    X'80'          Bit 24 - Mask on interval timer interruption
KEYMASK   EQU    X'40'          Bit 25 - Mask on operator key interruption
SIGMASK   EQU    X'20'          Bit 26 - Mask on external signals 2-7
```

Bits Defined in CREG 9

```
                 BYTE 0
PERSUBR   EQU    X'80'          Bit 00 - Monitor successful branches
PERIFET   EQU    X'40'          Bit 01 - Monitor instruction fetches
PERSALT   EQU    X'20'          Bit 02 - Monitor storage alteration
PERGPRS   EQU    X'10'          Bit 03 - Monitor register alteration
```

Bits Defined in CREG14

```
                 BYTE 0
HARDSTOP EQU     X'80'          Bit 00 - Check stop control
SYNCLOG   EQU    X'40'          Bit 01 - Synchronous logout control
IOLOG     EQU    X'20'          Bit 02 - I/O logout control
RECOVRPT EQU     X'08'          Bit 04 - Recovery report mask
CONFGRPT EQU     X'04'          Bit 05 - Configuration report mask
DAMAGRPT EQU     X'02'          Bit 06 - External damage report mask
WARNGRPT EQU     X'01'          Bit 07 - Warning condition report mask

                 BYTE 1
ASYNELOG EQU     X'80'          Bit 08 - Asynchronous extended logout control
ASYNFLOG EQU     X'40'          Bit 09 - Asynchronous fixed logcut control
```

Bits Defined for TRANS Macro

```
BRING    EQU   X'80'          Bring requested page
DEFER    EQU   X'40'          Defer execution until page in storage
LOCK     EQU   X'20'          Lock page for I/O operation
IOERETN  EQU   X'10'          Return I/O errors to caller
SYSTEM   EQU   X'08'          Call to DMKPTRAN for system virtual machine space
```

Equates for PARM Field for Calls to DMKBLDRT/DMKBLDRL

```
DELSEGS  EQU   X'80'          Release the segment tables
DELPAGES EQU   X'40'          Release the page/swap tables
NEWPAGES EQU   X'08'          Build new page/swap table
NEWSEGS  EQU   X'04'          Build new segment table
KEEPSEGS EQU   X'02'          Retain information in old segment table
OLDVMSEG EQU   X'01'          VMSEG pointer in VMBLOK valid
```

Bits Defined for Terminal I/O via DMKQCN

```
ERRMSG   EQU   X'0800'        Output - Control program error message
NORET    EQU   X'0400'        Output - Return immediately after call
DFRET    EQU   X'0200'        Output - Free buffer after queueing
OPERATOR EQU   X'0100'        Output - Message for system operator
LOGDROP  EQU   X'80'          Output - Logoff and drop line after message
LOGHOLD  EQU   X'40'          Output - Logoff and hold line after message
PRIORITY EQU   X'20'          Output - Write this message immediately
VMGENIO  EQU   X'10'          I/O request generated by virtual machine
NOAUTO   EQU   X'04'          Output - Suppress auto carriage return
ALARM    EQU   X'02'          Output - Sound the audible alarm
NOTIME   EQU   X'01'          Output - Suppress time stamp on message
INHIBIT  EQU   X'08'          Input - Prevent display of this data
EDIT     EQU   X'04'          Input - Edit input data for corrections
UCASE    EQU   X'02'          Input - Translate data to upper case
```

Equates for Spool File Recovery Routine - DMKCKS

```
RDRCHN   EQU   X'01'          SFBLOK goes on reader chain
PCHCHN   EQU   X'02'          SFBLOK goes on punch chain
PRTCHN   EQU   X'04'          SFBLOK goes on print chain
ADDSFB   EQU   X'08'          Add new SFBLOK to recovery cylinder
CHGSFB   EQU   X'10'          Change existing SFBLOK
DELSFB   EQU   X'20'          Delete SFBLOK from checkpoint
OPNSFB   EQU   X'40'          It is an open print-punch file
ACTSFB   EQU   X'80'          File being printed or punched
CHGRDV   EQU   X'0100'        Change attributes of real device
CHGSHQ   EQU   X'0200'        Checkpoint a SHQBLOK
```

MONITOR Class and Code Definitions

```
MNCLPERF EQU   X'00'          MONITOR PERFORM class
MNCOSYS  EQU   X'0000'        PERFORM class; system performance
MNCOTH   EQU   X'0061'        MONITOR tape header record
MNCOTT   EQU   X'0062'        MONITOR tape trailer record
MNCOSUS  EQU   X'0063'        MONITOR collection suspension record
MNCLRESP EQU   X'01'          MONITOR RESPONSE class
MNCOBRD  EQU   X'0000'        RESPONSE class; begin read code
MNCOWRIT EQU   X'0001'        RESPONSE class; write code
MNCOERD  EQU   X'0002'        RESPONSE class; end read code
MNCLSCH  EQU   X'02'          MONITOR SCHEDULE class
MNCODQ   EQU   X'0002'        SCHEDULE class; drop queue code
MNCOAQ   EQU   X'0003'        SCHEDULE class; add to queue code
MNCOAEL  EQU   X'0004'        Schedule class; add to eligable list code
MNCLUSER EQU   X'04'          MONITOR USER class
MNCOUSER EQU   X'0000'        USER class; user data
```

MONITOR Class and Code Definitions (continued)

```
MNCLINST EQU    X'05'           MONITOR instruction simulation class
MNCOSIM  EQU    X'0000'         INST class; instruction simulation code
MNCLDAST EQU    X'06'           MONITOR DASD/TAPE class
MNCODASH EQU    X'0000'         DASTAP class; first record
MNCODAS  EQU    X'0001'         DASTAP class; data records
MNCLSEEK EQU    X'07'           MONITOR DASD class
MNCOCYL  EQU    X'0000'         DASD class; SEEKs code
MNCLSYS  EQU    X'08'           MONITOR SYSTEM PROFILE class
MNCODA   EQU    X'0002'         SYS class; DASD data
```

VM/370 REGISTERS

Symbolic Register Equates

```
R0       EQU   0        ---
R1       EQU   1           |
R2       EQU   2           |
R3       EQU   3           |
R4       EQU   4           |
R5       EQU   5           |
R6       EQU   6           |
R7       EQU   7           General
R8       EQU   8           Register
R9       EQU   9           Definitions
R10      EQU   10          |
R11      EQU   11          |
R12      EQU   12          |
R13      EQU   13          |
R14      EQU   14          |
R15      EQU   15       ---|

Y0       EQU   0           Floating
Y2       EQU   2           Point
Y4       EQU   4           Register
Y6       EQU   6           Definitions

                        ---
C0       EQU   0           |
C1       EQU   1           |
C2       EQU   2           |
C3       EQU   3           |
C4       EQU   4           |
C5       EQU   5           |
C6       EQU   6           |
C7       EQU   7           Control
C8       EQU   8           Register
C9       EQU   9           Definitions
C10      EQU   10          |
C11      EQU   11          |
C12      EQU   12          |
C13      EQU   13          |
C14      EQU   14          |
C15      EQU   15       ---|
```

This appendix contains Assembler language equate symbols used in CMS to reference data for:

- CMS Usage
- CMS Registers

CMS USAGE EQUATES

```
Field
Name                            Field Description
-----                           -----------------
```

Bits Defined in the Program Status Word (PSW)

```
CHAN0    EQU   X'80'       Bit 00 - Channel 0 mask
CHAN1    EQU   X'40'       Bit 01 - Channel 1 mask
CHAN2    EQU   X'20'       Bit 02 - Channel 2 mask
CHAN3    EQU   X'10'       Bit 03 - Channel 3 mask
CHAN4    EQU   X'08'       Bit 04 - Channel 4 mask
CHAN5    EQU   X'04'       Bit 05 - Channel 5 mask
CHANM    EQU   X'02'       Bit 06 - Input/output mask
EXTM     EQU   X'01'       Bit 07 - External mask

ECMM     EQU   X'08'       Bit 12 - Extended control mode mask
MCKM     EQU   X'04'       Bit 13 - Machine check mask
WAIT     EQU   X'02'       Bit 14 - Wait state mask
PROB     EQU   X'01'       Bit 15 - Problem state mask

FOFM     EQU   X'08'       Bit 36 - Fixed-point overflow mask
DOFM     EQU   X'04'       Bit 37 - Decimal overflow mask
EUFM     EQU   X'02'       Bit 38 - Exponent underflow mask
SIGM     EQU   X'01'       Bit 39 - significance mask
```

Bits Defined in the Channel Status Word (CSW)

```
ATTN     EQU   X'80'       Bit 32 - Attention
SM       EQU   X'40'       Bit 33 - Status modifier
CUE      EQU   X'20'       Bit 34 - Control unit end
BUSY     EQU   X'10'       Bit 35 - Busy
CE       EQU   X'08'       Bit 36 - Channel end
DE       EQU   X'04'       Bit 37 - Device end
UC       EQU   X'02'       Bit 38 - Unit check
UE       EQU   X'01'       Bit 39 - Unit exception

PCI      EQU   X'80'       Bit 40 - Program-controlled interruption
ICL      EQU   X'40'       Bit 41 - Incorrect length
PGC      EQU   X'20'       Bit 42 - Program check
PTC      EQU   X'10'       Bit 43 - Protection check
CDC      EQU   X'08'       Bit 44 - Channel data check
CCC      EQU   X'04'       Bit 45 - Channel control check
ICC      EQU   X'02'       Bit 46 - Interface control check
CHC      EQU   X'01'       Bit 47 - Chaining check
```

Field Name			Field Description
Common Channel Command Codes			
WRITE	EQU	X'01'	Write
READ	EQU	X'02'	Read
NOP	EQU	X'03'	No operation
SENSE	EQU	X'04'	Sense
WRDATA	EQU	X'05'	Write data
RDDATA	EQU	X'06'	Read data
SEEK	EQU	X'07'	Seek
TIC	EQU	X'08'	Transfer in channel
WRITE1	EQU	X'09'	Write and space 1
RDCONS	EQU	X'0A'	Read from console
SETSEC	EQU	X'23'	Set sector
SEARCH	EQU	X'31'	Search ID equal
Bits Defined in a Channel Command Word (CCW)			
CD	EQU	X'80'	Bit 32 - Chain data
CC	EQU	X'40'	Bit 33 - Command chain
SILI	EQU	X'20'	Bit 34 - Suppress incorrect length
SKIP	EQU	X'10'	Bit 35 - Suppress data transfer
PCIF	EQU	X'08'	Bit 36 - Cause program control interruption
IDA	EQU	X'04'	Bit 37 - Indirect data address

CMS REGISTER EQUATES

Field
Name

General Purpose Registers

```
R0      EQU   0
R1      EQU   1
R2      EQU   2
R3      EQU   3
R4      EQU   4
R5      EQU   5
R6      EQU   6
R7      EQU   7
R8      EQU   8
R9      EQU   9
R10     EQU   10
R11     EQU   11
R12     EQU   12
R13     EQU   13
R14     EQU   14
R15     EQU   15
```

Floating-Point Registers

```
F0      EQU   0
F2      EQU   2
F4      EQU   4
F6      EQU   6
```

Extended Control Registers

```
C0      EQU   0
C1      EQU   1
C2      EQU   2
C3      EQU   3
C4      EQU   4
C5      EQU   5
C6      EQU   6
C7      EQU   7
C8      EQU   8
C9      EQU   9
C10     EQU   10
C11     EQU   11
C12     EQU   12
C13     EQU   13
C14     EQU   14
C15     EQU   15
```

RECORD 0 TRACK 0 CYLINDER 0 ONLY

Record 0 (8 bytes long) of all tracks other than track 0 is initialized to X'00'.

```
      32 Pages/cylinder 2314,2319
     ┌──┬──┬──┬──┬──┬──┬──┬──┐
    *|E0 00 00 00 00 00 00 00|
     └──┴──┴──┴──┴──┴──┴──┴──┘
      |
    11100000

      57 pages/cylinder 3330
     ┌──┬──┬──┬──┬──┬──┬──┬──┐
     |E0 00 00 00 00 00 00 00|
     └──┴──┴──┴──┴──┴──┴──┴──┘

      24 pages/cylinder 2305
     ┌──┬──┬──┬──┬──┬──┬──┬──┐
     |E0 00 00 00 00 00 00 00|
     └──┴──┴──┴──┴──┴──┴──┴──┘

      120 pages/cylinder 3350 (native mode)
     ┌──┬──┬──┬──┬──┬──┬──┬──┐
     |F0 00 00 00 00 00 00 00|
     └──┴──┴──┴──┴──┴──┴──┴──┘
```

All Page Records, 4096 Bytes Each

2314 and 2319	32 pages/cylinder
3330 series	57 pages/cylinder
2305 and 3340	24 pages/cylinder
3350	120 pages/cylinder

Cylinder 0 contains less pages because this area is used by CP.

---

*The first three pages of cylinder 0 are always flagged in use. On all other cylinders, the first byte is a hexadecimal '00' unless the disk area is flagged as bad. Record 0 of all tracks other than track 0 is initialized to hexadecimal '00'.

RECORD 1 (24 BYTES)

IPL record -- Puts system into wait state if storage device is initial program loaded.

00020000	0000000C	03000000	20000000	00000000	00000000

RECORD 2, 4096 BYTES

Checkpoint record -- This is the checkpoint program load at CP IPL time to retrieve and save control information for a warm start.

RECORD 3

4 byte key of VOL1
80 byte data record

Key

VOL1

Record

```
Bytes ┌─────────┬─────────┬─────────┬─────────┬─────────┐
 1-20 │E5D6D3F1 xx────────>xxF000 00000005 00000000│
      │                                            │
21-40 │0040────────────────────────────────────>40│
      │                                            │
41-60 │4000───────>00C3D7 F3F7F040 40404040 40───>40│
      │                                            │
61-80 │40──────────────────────────────────────>40│
      └─────────┴─────────┴─────────┴─────────┴─────────┘
```

where:

xx->xx is a 6-byte label

Bytes 13-16 contain a pointer to the VTOC

Bytes 46-50 identify the system

Bytes 52-55 contain a pointer to the active directory

RECORD 4

1024 bytes Track 0 Cylinder 0

Allocation byte map - used to identify cylinder 1 usage. Each byte identifies one cylinder.

```
 0                             ┌─> all 0<─┐
┌─────────┬──────────────┬─┼─────────┼┐
│00000100 040200───>FF  0000────>0000│
└─────────┴──────────┼─┴──────────────┘
                          *
```

* FF defines the last cylinder + 1 that can be allocated. This varies depending on the device.

00 = temporary
01 = permanent
02 = T-disk
04 = directory

RECORD 5

44 bytes key Track 0, Cylinder 0
96 bytes data area

Format 4 OS DSCB type label - used to be compatible with OS.

```
┌───────────┐   ┌────────────────────┐
│ 04────>04│   │   FORMAT 4 LABEL   │
└───────────┘   └────────────────────┘
   44 Key          96 Byte Data
```

RECORD 6

44 bytes key Track 0, Cylinder 0
96 bytes data area

Format 5 OS DSCB type label for compatibility with OS.

```
┌───┬──┬──┬──┬───┐   ┌──────────────────┐
│ 05│05│05│05│00 │   │ OS FORMAT 5 LABEL │
└───┴──┴──┴──┴───┘   └──────────────────┘
  44 Byte Key          96 Byte Data Area
```

RECORD F3

4096 bytes - 1 page, track 0 or track 1

F3 Record is reserved for CP system use. Referred to as filler record.

RECORD F4

1624 bytes, Track 1 (2314, 2319 only)

F4 used only on 2314 and 2319 devices to align Record 4 in proper position on track.

RECORD 4

824 bytes track 1, cylinder 0 (2314, 2319 only)

First segment of Record 4 to be used for paging.

2314 RECORD LAYOUT

CYLINDER 0, TRACK 0

```
  R0   R1  R2   Key  R3   R4   Key    R5   Key    R6
┌────┬──┬──────┬─┬─────┬──────┬──┬───────┬──┬───────┬────┐
|Page|I |Check |V|VOL1 |Alloc |  |Format |  |Format |    |
|Bit |P |Point |O|Label|Byte  |  | 4     |  | 5     |    |
|Map |L |      |L|     |Map   |  |       |  |       |    |
|    |  |      |1|     |      |  |       |  |       |    |
|  8 |24| 4096 |4|  80 | 1024 |44|    96 |44|    96 |    |
└────┴──┴──────┴─┴─────┴──────┴──┴───────┴──┴───────┴────┘
```

Cylinder 0, Track 1

```
   R0          RF3        RF4         R4
┌───────┐   ┌───────┐   ┌───────┐   ┌───────┐
|       |   |1 PAGE |   |FILLER |   |       |
|       |───|       |───|       |───|       |
|   8   |   | 4096  |   | 1624  |   |824    |
└───────┘   └───────┘   └───────┘   └───────┘
```

ALL CYLINDERS EXCEPT 0, TRACK 0

```
  R0            R1            R2
┌────────┐   ┌────────┐   ┌────────┐
|Page    |   |        |   |        |
|Bit Map |───|        |───|        |
|   8    |   | 4096   |   | 2472   |
└────────┘   └────────┘   └────────┘
```

These records appear as above formats if cylinder is 0.

Track 1

R0	R2	R3	R4
8	1624	4096	824

Track 2

R0	R4	R5
8	3272	3296

Track 3

R0	R5	R6	R7
8	800	4096	1648

Track 4

R0	R7	R8
8	2448	4096

Note: Tracks 0 to 4 are repeated for tracks 5 to 9 (R9-R16), 10 to 14, (R17-R24), and 15 to 19 (R25-R32). The last record is R32.

3330 SERIES RECORD LAYOUT

CYLINDER 0, TRACK 0

R0	R1	R2	Key	R3	R4	Key	R5	Key	R6	RF3	
Page Bit Map	IPL	Check Point	VOL1	VOL1 Label	Byte Map		Format 4		Format 5	1 Page	
8	24	4096	4	80	1024	44	96	44	96	4096	

ANY CYLINDER EXCEPT 0

Track 0

```
    R0        R1          R2          R3
 ┌────────┐ ┌────────┐ ┌────────┐ ┌────────┐
 |Page    | |        | |        | |        |
 |Bit Map|—|        |—|        |—|        |
 |        | |        | |        | |        |
 |   8    | | 4096   | | 4096   | | 4096   |
 └────────┘ └────────┘ └────────┘ └────────┘
                          |
```

Track 18

```
                          |
                          |
    R0        R55         V   R56        R57
 ┌────────┐ ┌────────┐ ┌────────┐ ┌────────┐
 |        |—|        |—|        |—|        |
 |        | |        | |        | |        |
 |   8    | | 4096   | | 4096   | | 4096   |
 └────────┘ └────────┘ └────────┘ └────────┘
```

2305 MODEL 1 AND MODEL 2

CYLINDER 0, TRACK 0

```
  R0   R1   R2   Key  R3    R4   Key    R5   Key    R6    RF3
┌─────┬──┬───────┬─┬──────┬────┬──┬─────────┬──┬───────┬─────┬─┐
|Page |I |Check  |V|VOL1  |Byte|  |Format   |  |Format | 1   | |
|Bit  |P |Point  |O|Label |Map |  | 4       |  | 5     |Page | |
|Map  |L |       |L|      |    |  |         |  |       |     | |
|     |  |       |1|      |    |  |         |  |       |     | |
|  8  |24| 4096  |4|  80  |1024|44|   96    |44|   96  |4096 | |
└─────┴──┴───────┴─┴──────┴────┴──┴─────────┴──┴───────┴─────┴─┘
```

ANY CYLINDER EXCEPT 0

Track 0

```
    R0        R1          R2          R3
 ┌────────┐ ┌────────┐ ┌────────┐ ┌────────┐
 |Page    | |        | |        | |        |
 |Bit Map|—|        |—|        |—|        |
 |        | |        | |        | |        |
 |   8    | | 4096   | | 4096   | | 4096   |
 └────────┘ └────────┘ └────────┘ └────────┘
                          |
```

Track 7

```
                          |
                          |
    R0        R22         V   R23        R24
 ┌────────┐ ┌────────┐ ┌────────┐ ┌────────┐
 |        |—|        |—|        |—|        |
 |        | |        | |        | |        |
 |  8     | | 4096   | | 4096   | | 4096   |
 └────────┘ └────────┘ └────────┘ └────────┘
```

3340 SERIES RECORD LAYOUT

CYLINDER 0, TRACK 0

```
  R0   R1   R2   Key  R3    R4   Key    R5   Key   R6
┌────┬──┬─────┬─┬─────┬────┬───┬──────┬───┬──────┐
│Page│I │Check│V│VOL1 │Byte│   │Format│   │Format│
│Bit │P │Point│C│Label│Map │   │ 4    │   │ 5    │
│Map │L │     │L│     │    │   │      │   │      │
│    │  │     │1│     │    │   │      │   │      │
│  8 │24│ 4096│4│  80 │1024│44 │   96 │44 │   96 │
└────┴──┴─────┴─┴─────┴────┴───┴──────┴───┴──────┘
```

ANY CYLINDER EXCEPT 0

Track 0

```
    R0        R1           R2           R3
┌────────┐  ┌────────┐  ┌─────────┐  ┌────────┐
│Page    │  │        │  │         │  │        │
│Bit Map │──│        │──│         │──│        │
│        │  │        │  │         │  │        │
│   8    │  │  4096  │  │  4096   │  │  4096  │
└────────┘  └────────┘  └─────────┘  └────────┘
```

Track 11

```
                 │
                 │
                 │
                 V
    R0          R23         R24
┌────────┐  ┌────────┐  ┌────────┐
│        │──│        │──│        │
│        │  │        │  │        │
│   8    │  │  4096  │  │  4096  │
└────────┘  └────────┘  └────────┘
```

CP RESTRICTIONS

A virtual machine created by VM/370 is capable of running an IBM System/360 or System/370 operating system as long as certain VM/370 restrictions are not violated. If your virtual machine produces unexpected results, be sure that none of the following restrictions are violated.

DYNAMICALLY MODIFIED CHANNEL PROGRAMS

In general, virtual machines may not execute channel programs that are dynamically modified (that is, channel programs that are changed between the time the START I/O (SIO) is issued and the time the input/output ends, either by the channel program itself or by the CPU). However, some dynamically modified channel programs are given special consideration by CP: specifically, those generated by the Indexed Sequential Access Method (ISAM) running under OS/PCP, OS/MFT, and OS/MVT; those generated by ISAM running in an OS/VS virtual=real partition; and those generated by the OS/VS Telecommunications Access Method (TCAM) Level 5, with the VM/370 option.

The self-modifying channel programs that ISAM generates for some of its operations receive special handling if the virtual machine using ISAM has that option specified in its VM/370 directory entry. There is no such restriction for DOS ISAM, or for ISAM if it is running in an OS/VS virtual=virtual partition. If ISAM is to run in an OS/VS virtual=real partition, you must specify the ISAM option in the VM/370 directory entry for the OS/VS virtual machine.

Virtual machines using OS/VS TCAM (Level 5, generated or invoked with the VM/370 option) issue a DIAGNOSE instruction when the channel program is modified. This instruction causes CP to reflect the change in the virtual CCW string to the real CCW string being executed by the channel. CP is then able to execute the dynamically modified channel program properly.

The restriction against dynamically modified channel programs does not apply if the virtual machine has the virtual=real performance option and the NOTRANS option has been set on.

MINIDISK RESTRICTIONS

The following restrictions exist for minidisks:

1. In the case of read home address with the skip bit off, VM/370 modifies the home address data in user storage at the completion of the channel program because the addresses must be converted for minidisks; therefore, the data buffer area may not be dynamically modified during the input/output operation.

2. On a minidisk, if a CCW string uses multitrack search on input/output operations, subsequent operations to that disk must have preceding seeks or continue to use multitrack operations. There is no restriction for dedicated disks.

3. OS/PCP, MFT, and MVT ISAM or OS/VS ISAM running virtual=real may be used with a minidisk only if the minidisk is located at the beginning of the physical disk (that is, at cylinder zero). There is no such restriction for DOS ISAM or OS/VS ISAM running virtual=virtual.

4. VM/370 does not return an end-of-cylinder condition to a virtual machine that has a virtual 2311 mapped to the top half (that is, tracks 0 through 9) of 2314 or 2319 cylinders.

5. If the user's channel program for a minidisk does not perform a seek operation, then to prevent accidental accessing, VM/370 inserts a positioning seek operation into the user's channel program. Thus, certain channel programs may generate a condition code (CC) of zero on a SIO instead of an expected CC of one, which is reflected to the virtual machine. The final status is reflected to the virtual machine as an interrupt.

6. A DASD channel program directed to a 3330, 3340, or 3350 device may give results on dedicated drives which differ from results on minidisks having non-zero relocation factors if the channel program includes multiple-track operations and depends on a search ID high or a search ID equal or high to terminate the program. This is because the record 0 count fields on the 3330, 3340, and 3350 must contain the real cylinder number of the track on which they reside. Therefore, a search ID high, for example, based on a low virtual cylinder number may terminate prematurely if a real record 0 is encountered.

   Note: Minidisks with non-zero relocation factors on 3330, 3340, and 3350 devices are not usable under OS and OS/VS systems. This is because the locate catalog management function employs a search ID equal or high CCW to find the end of the VTOC.

7. The IBCDASDI program cannot assign alternate tracks for a 3330, 3340, or 3350 minidisk.

8. If the DASD channel programs directed to 3330/3340/3350 devices include a write record R(0), results differ depending on whether the 3330/3340/3350 is dedicated (this includes a minidisk defined as the entire device) or nondedicated. For a dedicated 3330/3340/3350, a write R(0) is allowed, but the user must be aware that the track descriptor record may not be valid from one 3330/3340/3350 to another. For a nondedicated 3330/3340/3350, a write record R(0) is replaced by a read record R(0) and the skip flag is set on. This could result in a command reject condition due to an invalid command sequence.

9. When performing DASD I/O, if the record field of a search ID argument is zero when a virtual Start I/O is issued, but the search ID argument is dynamically read by the channel program before the search ID CCW is executed, then the real search ID uses the relocated search argument instead of the argument that was read dynamically. To avoid this problem, the record field of a search ID argument should not be set to binary zero if the search argument is to be dynamically read or if a search ID on record 0 is not intended.

TIMING DEPENDENCIES

Timing dependencies in input/output devices or programming do not function consistently under VM/370:

1. The following telecommunication access methods (or the designated option) violate the restriction on timing dependency by using program-controlled interrupt techniques and/or the restriction on dynamically modified channel programs:

   - OS Basic Telecommunications Access Method (BTAM) with the dynamic buffering option.
   - OS Queued Telecommunications Access Method (QTAM).
   - DOS Queued Telecommunications Access Method (QTAM).
   - OS Telecommunications Access Method (TCAM).
   - OS/VS Telecommunications Access Method (TCAM) Level 4 or earlier, and Level 5 if TCAM is not generated or invoked with the VM/370 option.

   These access methods may run in a virtual=real machine with CCW translation suppressed by the SET NOTRANS ON command. Even if SET NOTRANS ON is issued, CCW translation will take place if one of the following conditions is in effect:

   - The channel program is directed at an a nondedicated device (such as a spooled unit record device, a virtual CTCA, a minidisk, or a console).
   - The channel program starts with a SENSE operation code.
   - The channel program is for a dialed terminal.
   - START I/O tracing is in effect.
   - The CAW is in page zero or beyond the end of the virtual=real area.

   (OS BTAM can be generated without dynamic buffering, in which case no virtual machine execution violations occur. However, the BTAM reset poll macro will not execute under VM/370 if issued from third level storage. For example, a reset poll macro has a NOP effect if executed from a virtual=virtual storage under VS1 which is running under VM/370.)

2. Programming that makes use of the PCI channel interrupt for channel program modification or processor signalling must be written so that processing can continue normally if the PCI is not recognized until I/O completion or if the modifications performed are not executed by the channel.

3. Devices that expect a response to an interrupt within a fixed period of time may not function correctly because of execution delays caused by normal VM/370 system processing. An example of such a device is the IBM 1419 Magnetic Character Reader.

4. The operation of a virtual block multiplexer channel is timing dependent. For this reason, the channel appears available to the virtual machine operating system, and channel available interrupts are not observed. However, operations on virtual block-multiplexing

devices should use the available features like Rotational Position Sensing to enhance utilization of the real channels.

CPU MODEL-DEPENDENT FUNCTIONS

On the System/370 Model 158 only, the Virtual Machine Assist feature cannot operate concurrently with the 7070/7074 compatibility feature (Feature #7117).

Programs written for CPU model-dependent functions may not execute properly in the virtual machine under VM/370. The following points should be noted:

1. Programs written to examine the machine logout area do not have meaningful data since VM/370 does not reflect the machine logout data to a virtual machine.
2. Programs written to obtain CPU identification (via the Store CPU ID instruction, STIDP) receive the real machine value. When the STIDP instruction is issued by a virtual machine, the version code contains the value 255 in hexadecimal ("FF") to represent a virtual machine.
3. Programs written to obtain channel identification (via the Store Channel ID instruction, STIDC) receive information from the virtual channel block. Only the virtual channel type is reflected; the other fields contain zeroes.
4. No simulation of other CPU models is attempted by VM/370.

VIRTUAL MACHINE CHARACTERISTICS

Other characteristics that exist for a virtual machine under VM/370 are as follows:

1. If the virtual=real option is selected for a virtual machine, input/output operations specifying data transfer into or out of the virtual machine's page zero, or into or out of storage locations whose addresses are greater than the storage allocated by the virtual=real option, must not occur. The storage-protect-key mechanism of the IBM System/370 CPU and channels operates in these situations but is unable to provide predictable protection to other virtual machines. In addition, violation of this restriction may compromise the integrity of the system. The results are unpredictable.
2. VM/370 has no multiple path support and, hence, does not take advantage of the two-channel switch. However, a two-channel switch can be used between the IBM System/370 running a virtual machine under VM/370 and another CPU.
3. The DIAGNOSE instruction cannot be issued by the virtual machine for its normal function. VM/370 uses this instruction to allow the virtual machine to communicate system services requests. The Diagnose interface requires the operand storage addresses passed to it to be real to the virtual machine issuing the DIAGNOSE instruction. For more information about the DIAGNOSE instruction in a virtual machine, see the _VM/370: System Programmer's Guide_.

4. A control unit normally never appears busy to a virtual machine. An exception exists when a forward space file or backward space file command is executed for a tape drive. Subsequent I/O operations to the same virtual control unit result in a control unit busy condition until the forward space file or backward space file command completes. If the real tape control unit is shared by more than one virtual machine, a control unit busy condition is reflected only to the virtual machine executing the forward space file or backward space file command. When a virtual machine attempts an I/O operation to a device for which its real control unit is busy, the virtual machine is placed in I/O wait (nondispatchable) until the real control unit is available. If the virtual machine executed a SIOF instruction (rather than SIO) and was enabled for block-multiplexing, it is not placed in I/O wait for the above condition.

5. The CP IPL command cannot simulate self-modifying IPL sequences off dedicated unit record devices or certain self-modifying IPL sequences off tape devices.

6. The VM/370 spooling facilities do not support punch-feed-read, stacker selection, or column binary operations. Detection of carriage control channels is supported for a virtual 3211 only.

7. VM/370 does not support count control on the virtual 1052 operator's console.

8. Programs that use the integrated emulators function only if the real computing system has the appropriate compatibility feature. VM/370 does not attempt simulation. The DOS emulator running under OS or OS/VS is not supported under VM/370.

9. The READ DIRECT and WRITE DIRECT instructions are not supported for a virtual machine.

10. The System/370 SET CLOCK instruction cannot be simulated and, hence, is ignored if issued by a virtual machine. The System/370 STORE CLOCK instruction is a nonprivileged instruction and cannot be trapped by VM/370; it provides the true TOD clock value from the real CPU.

11. The 1050/1052 Model 2 Data Communication System is supported only as a keyboard operator's console. Card reading, paper tape I/O, and other modes of operation are not recognized as unique, and hence may not work properly. This restriction applies only when the 1050 system is used as a virtual machine operator's console. It does not apply when the 1050 system is attached to a virtual machine via a virtual 2701, 2702, or 2703 line.

12. The pseudo-timer (usually device address 0FF, device type TIMER) does not return an interrupt from a Start I/O; therefore, do not use EXCP to read this device.

13. A virtual machine device IPL with the NOCLEAR option overlays one page of virtual machine storage. The IPL simulator uses one page of the virtual machine to initiate the IPL function. The starting address of the overlayed page is either the result of the following formula:

    $$\frac{\text{virtual machine size}}{2} = \text{starting address of the overlayed page}$$

    or the hexadecimal value 20,000, whichever is smaller.

14. To maintain system integrity, data transfer sequences to and from a virtual system console are limited to a maximum of 2032 bytes. Channel programs containing data transfer sequences that violate this restriction are terminated with an interrupt whose CSW status indicates incorrect length and a channel program check.

    Note: A data transfer sequence is defined as one or more read or write CCWs connected via chain data. The introduction of command chaining defines the start of a new data transfer sequence.

15. When an I/O error occurs on a device, the System/370 hardware maintains a contingent connection for that device until a SENSE channel command is executed and sense data is recorded. That is, no other I/O activity can occur on the device during this time. Under VM/370, the contingent connection is maintained until the SENSE command is executed, but I/O activity from other virtual machines can begin on the device while the sense data is being reflected to the virtual machine. Therefore, the user should be aware that on a shared disk, the access mechanism may have moved during this time.

16. The mode setting for 7-track tape devices is maintained by the control unit. Therefore, when a virtual machine issues the SET MODE channel command to a 7-track tape device, it changes the mode setting of all 7-track tape devices attached to that control unit.

    This has no effect on virtual machines (such as OS or DOS) that issue SET MODE each time a CCW string is to be executed. However, it can cause a problem if a virtual machine fails to issue a SET MODE with each CCW string executed. Another virtual machine may change the mode setting for another device on the same control unit, thereby changing the mode setting of all 7-track tape devices attached to that control unit.

17. OS/VS2 is supported in uniprocessor mode only.

18. A shared system or one that uses discontiquous saved segments cannot be loaded (via IPL) into a virtual machine running in the virtual=real area.

19. The DUMMY feature for VSAM data sets is not supported and should not be used at program execution time. Specifying this option on the DLBL command will cause an execution-time OPEN error. See *VM/370: System Messages* for additional information.

CMS RESTRICTIONS

The following restrictions apply to CMS, the conversational subsystem of VM/370:

1. CMS executes only on a virtual IBM System/370 provided by VM/370.

2. The maximum sizes in cylinders of CMS minidisks are as follows:

Disk	Maximum Cylinders	CMS/VSAM
2314/2319	203	200
3330 Series	246	404
3340 Model 35	349	348
3340 Model 70/3344	682	696
3350 Series	115	not supported in native mode

3. CMS employs the spooling facilities of VM/370 to perform unit record I/O. However, a program running under CMS can issue its own SIOs to attached dedicated unit record devices.

4. Only those OS and DOS facilities that are simulated by CMS can be used to execute OS and DOS programs produced by language processors under CMS.

5. Many types of object programs produced by CMS (and OS) languages can be executed under CMS using CMS's simulation of OS supervisory functions. Although supported in OS and DOS virtual machines under VM/370, the writing and updating of non-VSAM OS data sets and DOS files are not supported under CMS.

6. CMS can read sequential and partitioned OS data sets and sequential DOS files, by simulating certain OS macros.

   The following restrictions apply when CMS reads OS data sets that reside on OS disks:

   - Read-password-protected data sets are not read.
   - BDAM and ISAM data sets are not read.
   - Multivolume data sets are read as single-volume data sets. End-of-volume is treated as end-of-file and there is no end-of-volume switching.
   - Keys in data sets with keys are ignored and only the data is read.
   - User labels in user-labeled data sets are bypassed.

   The following restrictions apply when CMS reads DOS files that reside on DOS disks:

   - Only DOS sequential files can be read. CMS options and operands that do not apply to OS sequential data sets (such as the MEMBER and CONCAT options of FILEDEF and the PDS option of MOVEFILE) also do not apply to DOS sequential files.
   - The following types of DOS files cannot be read:

     --DOS DAM and ISAM files.

     --Files with the input security indicator on.

     --DOS files that contain more than 16 user label and/or data extents. (If the file has user labels, they occupy the first extent; therefore the file must contain no more than 15 data extents.)

   - Multivolume files are read as single-volume files. End-of-volume is treated as end-of-file. There is no end-of-volume switching.
   - User labels in user-labeled files are bypassed.
   - Since DOS files do not contain BLKSIZE, RECFM, or LRECL parameters, these parameters must be specified via FILEDEF or DCB parameters; otherwise, defaults of BLOCKSIZE=32760 and RECFM=U are assigned. LRECL is not used for RECFM=U files.

- CMS does not support the use of OS/VS DUMMY VSAM data sets at program execution time, since the CMS/DOS implementation of the DUMMY statement corresponds to the DOS/VS implementation. Specifying the DUMMY option with the DLBL command will cause an execution-time error.

7. Assembler program usage of VSAM and the ISAM Interface Program (IIP) is not supported.

MISCELLANEOUS RESTRICTIONS

1. If you intend to run VM/370 Release 1 and pre-PLC 9 Release 2 systems alternately, apply Release 1 PLC 14 or higher (APAR V1179) to your Release 1 system, to provide compatibility and to prevent loss of spool files in case of a warm start. Changes to the spool file format in PLC 9 of Release 2 require a cold start when switching between pre-Release 2 PLC 9 and post-Release 2 PLC 9 systems.

2. The number of pages used for input/output must not exceed the total number of user pages available in real storage. Violation of this restriction causes the real computing system to be put into an enabled wait state.

3. If you intend to define more than 73 virtual devices for a single virtual machine, be aware that any single request for free storage in excess of 512 doublewords (a full page) may cause the VM/370 system to abnormally terminate (ABEND code PTR007) if the extra storage is not available on a contiguous page. Therefore, two contiguous pages of free storage must be available in order to log on a virtual machine with more than 73 virtual devices (three contiguous pages for a virtual machine with more than 146 virtual devices, etc.). Contiguous pages of free storage are sure to be available only immediately after IPL, before other virtual machines have logged on. Therefore, a virtual machine with more than 73 devices should be the first to log on after IPL. The larger the real machine size, the lesser the possibility of this occurring.

4. For remote 3270s, VM/370 supports a maximum of 16 binary synchronous lines, minus the number of 3704/3705 Communications Controllers in NCP mode minus one (if there are any 3704/3705 Communications Controllers in emulation mode).

5. If an I/O device (such as a disk or tape drive) drops ready status while it is processing virtual I/O activity, any virtual machine users performing I/O on that device are unable to continue processing or to log off. Also, the LOGOFF and FORCE commands are not effective because they do not complete until all outstanding I/O is finished. The system operator should determine which I/O device is involved and make that device ready once more.

Figure 67 indicates those devices that are supported by a CMS machine.

Virtual IBM Device	Virtual Address[1]	Symbolic Name	Device Type
3210, 3215, 1052, 3066, 3270	ccu	CON1	System console
2314, 3330, 3340 3350	190	DSK0	System disk (read-only)
2314, 3330, 3340 3350	191[2]	DSK1	Primary disk (user files)
2314, 2319, 3330, 3340, 3350	ccu	DSK2	Disk (user files)
2314, 2319, 3330, 3340, 3350	ccu	DSK3	Disk (user files)
2314, 2319, 3330, 3340, 3350	192	DSK4	Disk (user files)
2314, 2319, 3330, 3340, 3350	ccu	DSK5	Disk (user files)
2314, 2319, 3330, 3340, 3350	ccu	DSK6	Disk (user files)
2314, 2319, 3330, 3340, 3350	ccu	DSK7	Disk (user files)
2314, 2319, 3330, 3340, 3350	19E	DSK8	Disk (user files)
2314, 2319, 3330, 3340, 3350	ccu	DSK9	Disk (user files)
1403, 3211, 1443	00E	PRN1	Line printer
2540, 2501, 3505	00C	RDR1	Card reader
2540, 3525	00D	PCH1	Card punch
2415, 2420, 3410, 3420	181-4	TAP1-TAP4	Tape drives

[1]The device addresses shown are those that are preassembled into the CMS resident device table. These need only be modified and a new device table made resident to change the addresses.
[2]The virtual device address (ccu) of a disk for user files can be any valid System/370 device address, and can be specified by the CMS user when he activates a disk. If the user does not activate a disk immediately after loading CMS, CMS automatically activates the primary disk at virtual address 191.

Figure 67. Devices Supported by a CMS Virtual Machine

Figure 68 indicates the DIAGNOSE codes used in VM/370 and gives a brief explanation of its use.

Function Code	Class	Function	DMKHVC Label	DMKHVD Label
000	G	Store extended identification code.		HVDSTIDX
004	C,E	Examine data from real storage.		READCPC
008	G	Execute VM/370 CP command.	HVCONFN	
00C	G	Pseudo-timer facility.	HVCHRON	
010	G	Release virtual storage pages.	HVCPGRL	
014	G	Manipulate input spool files.		HCDSPRD
018	G	Standard DASD I/O.	HVCDISK	
01C	F	Clear I/O and MC recording areas.		HVDLRER
020	G	General virtual I/O interruptions.	HVCFAKE	
024	G	Virtual device type inquiry.		HVDDTYP
028	G	Dynamic TIC modification.	HVCDCPM	
02C	C,E,F	Get DASD address of error recording areas.		HVDEREP1
030	C,E,F	Read a page of error recording data.		HVDEREP2

Figure 68. Function Codes for DIAGNOSE Instruction (Part 1 of 2)

Function Code	Class	Function	DMKHVC Module	DMKHVD Module
034	C,F	Reads the system dump spool file.		HVDRSDF
038	C,E	Reads the system symbol table.		HVDRDSYM
03C	A,B,C	Dynamically updates the VM/370 directory.		HVDDIRCT
040		Reserved for IBM use.	HVCEXIT	
044		Reserved for IBM use.	HVCEXIT	
048		Reserved for IBM use.	HVCEXIT	
04C	any	Generate accounting cards.		HVDACCT
050	A,B,C	Saves 3704/3705 control program image.		HVD3705
054		Enable or disable external interruptions.		HVDEXPA
058	G	Virtual console interface for 3270.	HVCGRAF	
05C		Edit message according to EMSG settings.	HVCEMSG	
060		Provide virtual machine storage size.	HVCSTOR	
064		Load, find, or purge a named system.	HVCSYS	
100		Start of functions specified by a user.	HVCUSER	

Figure 68. Function Codes for DIAGNOSE Instruction (Part 2 of 2)

ZAP is a CMS command that modifies or dumps MODULE, LOADLIB, or TXTLIB files. It may be used to modify either fixed or variable length MODULE files. It is for use by system support personnel only.

Input control records control ZAP processing. They can be submitted either from the terminal or from a disk file. Using the VER and REP control records, you can verify and replace data or instructions in a control section (CSECT). Using the DUMP control record, you can dump all or part of a CSECT, or an entire member of a LOADLIB or TXTLIB file, or an entire module of a MODULE file.

The format of the ZAP command is:

```
+-------------------------------------------------------------------------+
|          | ( MODULE  )                                                  |
|  ZAP     | < LOADLIB > [libname1 ... libname3][(option...[)]]           |
|          | ( TXTLIB  )                                                  |
|          |                     options:                                 |
|          |                     [TERM          ][PRINT  ]                |
|          |                     [INPUT filename][NOPRINT]                |
+-------------------------------------------------------------------------+
```

where:

MODULE
LOADLIB
TXTLIB
: indicates the type of file that is to be modified or dumped.

libname
: is the library name containing the member to be modified or dumped. You can specify one to three library names. The libname is valid only for LOADLIB and TXTLIB files.

Options:

TERM [PRINT | NOPRINT]

indicates that input to the ZAP service program is submitted through the terminal. If you specify TERM, the prompting message ENTER: is issued, and you can then enter input control records up to 80 characters long. If you specify PRINT with TERM, all output prints on the printer, but only error messages display at the terminal. If you specify NOPRINT with TERM, nothing prints on the printer. All output except control records displays at the terminal.

INPUT filename [PRINT | NOPRINT]

specifies that input is submitted from a disk file, filename. This file must have a filetype of ZAP, and must be a fixed 80-byte sequential file residing on any accessible device. If you specify PRINT with INPUT filename, all output produced by the ZAP service program prints on the printer. In addition, commands and control records in error and error messages display at the terminal. If you specify NOPRINT with INPUT filename, nothing prints on the printer. All output displays at the terminal.

The following table shows the resulting output of valid option combinations:

OPTIONS	PRINT	NOPRINT
INPUT	Commands and control records in error and error messages on the terminal. Everything to printer.	Everything on the terminal. Nothing on the printer.
TERM	Only error messages on the terminal. Everything on the Printer.	Everything except control records on the terminal. Nothing on the printer.

ZAP INPUT CONTROL RECORDS

Seven types of ZAP control records exist: NAME, DUMP, BASE, VER or VERIFY, REP, comment, and END.

ZAP control records are free form and need not start in position one of the record but the ZAP program can accept only 80 characters of data for each control record. Separate all information by one or more blanks. All address fields including disp (displacement) fields in VER and REP control records must contain an even number of hexadecimal digits, to a maximum of six digits (0D, 02C8, 014318). Data fields in VER and REP control records must also contain an even number of hexadecimal digits, but are not limited to six digits.

If you wish, you may separate the data anywhere by commas (for example, 83256482 or 8325,6482). The commas have no effect on the operation.

The program sets the NOGO switch on if a control record is found to be in error. A file cannot be modified once the NOGO switch is turned on. The next valid NAME record turns the NOGO switch off. This means that if the control record is the NAME record, all succeeding records are ignored until the next NAME, DUMP, or END record. For any other error, only REP control records that follow are ignored.

<u>DUMP Control Record</u>

The DUMP control record resets the NOGO switch off. The DUMP control record must not immediately precede a BASE, VER, or REP control record. A NAME control record must precede the BASE, VER, and REP control records (if any) that follow a DUMP control record.

The DUMP control record allows you to dump a portion or all of a specified control section, or the complete member or module. The format of the output of the dump is hexadecimal with an EBCDIC translation of the hexadecimal data.

The DUMP control record is optional. The format of the DUMP control record is:

```
DUMP {membername} [csectname [startaddress [endaddress]]]
     {modulename} [ALL                                   ]
```

where:

membername
: is the name of the member to be dumped, or the member that contains the CSECT(s) to be dumped. This member must be found in one of the libraries specified in the ZAP command line. However, if the library is a CMS TXTLIB, its directory does not contain member names. Therefore, the program ignores the member name (although you must specify it), and the program searches for the csectname (which you must specify).

modulename
: is the name of the module to be dumped, or the module that contains the CSECT(s) to be dumped. If you specify a module that has no loader table, the program dumps the entire module.

csectname is the name of the control section that is to be dumped. If you do not specify csectname, the program dumps only the first CSECT. The csectname is required for CMS TXTLIBs, optional for OS TXTLIBs, LOADLIBs, and MODULE files. (See the discussion of csectname under "Name Control Record.") You must not specify csectname for a module created with the NOMAP option.

ALL specifies to the program to dump all CSECTs within the specified member or module. You can specify ALL for MODULE files, LOADLIBs, and OS TEXTLIBs, but not for CMS TXTLIBS. If you wish to dump all the CSECTs in a member of a CMS TXTLIB, you must issue a separate DUMP control record for each CSECT.

startaddress
: is the location within the specified CSECT where the dump is to begin. This must be two, four, or six hexadecimal digits. The start address is the displacement from the beginning of the CSECT. For example, if you wish to start dumping at address 08 in a CSECT that begins at location 400, you specify start address or 08, not 0408.

endaddress
: is the last address to be dumped. This must be two, four, or six hexadecimal digits. If you specify no address, the program dumps the rest of the CSECT. Note that start and end addresses apply only when you specify a csectname. If the file to be dumped contains undefined areas (such as a DS in a TXTLIB member), the hexadecimal portion of the dump contains blanks to indicate that the corresponding positions are undefined.

NAME Control Record

The NAME control record specifies the member or module and CSECT that contain the data to be verified or replaced by the ZAP operation. The format of the NAME control record is:

```
NAME  { membername } [csectname]
      { modulename }
```

where:

{ membername }
{ modulename }
is the member or module that you want to be searched for the desired CSECT.

csectname
is the name of the desired control section. You must specify csectname if the CSECT you wish to modify is in a CMS TXTLIB (that is, TXTLIB created by the TXTLIB command from CMS TEXT decks that do not have a NAME card following the END card). The directory of a CMS TXTLIB contains only CSECT names and no member names. The CSECT name specified in the NAME record is compared with CSECT names in the directory. If a CSECT match is found and no member name match is found, the member selected is the one that contains the CSECT name. The csectname is optional if the CSECT you wish to modify is a LOADLIB or an OS TXTLIB (that is, a TXTLIB created by the TXTLIB command from CMS TEXT decks that have a NAME card after the END card). The dictionaries of the specified libraries are searched for the member name and the member is then searched for the CSECT name, if you specified one. If you do not specify csectname for a LOADLIB or an OS TXTLIB, the program uses the first control section. The csectname is optional for a MODULE file. The module named in the NAME control record is located and, if you specified csectname, the first record is read to determine the number of records in the module and the availability of a loader table, which the program can then search for the csectname. If you do not specify csectname, the program uses the beginning location of the module. You are not allowed to specify csectname if the module was created with the NOMAP option. The NAME control record must precede the BASE, VER, and REP control records. If it does not, the program sets the NOGO switch on.

BASE Control Record

The BASE control record adjusts displacement values for subsequent VER or REP control records for a CSECT whose starting address is not location zero in an assembly listing. The format of the BASE control record is:

```
BASE    address
```

where:

address — is the starting address of the CSECT. The address must be two, four, or six hexadecimal digits. For example, for a CSECT starting at location 400, you would specify the BASE 0400 in the BASE control record. If a subsequent VER card requests verification of location 0408, the BASE of 0400 is subtracted from 0408, and the program verifies location 08 in the CSECT. This example applies if you specify TXTLIB, LOADLIB, or MODULE and the module map is present. However, if no module map is present for a MODULE file (that is, the module was generated with the NOMAP option), then all operations are performed as if the BASE address is location 0. For example, if you specify a BASE of 400 and the address you wish to inspect or modify is 408, then you must specify 08 and not 408 in REP and VER control records. The address in this case is from the start of the module. If you do not specify csectname in the NAME control record, you cannot specify any BASE value other than 00. The BASE control record is optional. See the discussion under "VER or VERIFY Control Record." If specified, the BASE control record must follow the NAME record, but it need not follow the NAME record immediately. For example, you could have the following sequence of control records: NAME, VER, REP, BASE, VER, REP.

VER or VERIFY Control Record

The VER control record requests verification of instructions or data within a CSECT. If the verification fails, the program does not perform a subsequent REP operation until it encounters another NAME control record.

The VER control record is optional. More than one VER record can follow a single NAME record.

The format of the VER control record is:

```
{ VERIFY }    disp    data
{ VER    }
```

where:

disp — is the hexadecimal displacement of the data to be inspected from the start of the CSECT, if you did not submit a BASE control record for this CSECT. If you did submit a BASE control record, then disp is the actual location of the data. The disp must be two, four, or six hexadecimal digits. This displacement does not have to be aligned on a fullword boundary. If this displacement value is outside the limits of the CSECT specified by the preceding NAME control record, the VERIFY control record is rejected.

data — is the data against which the data in the CSECT is to be compared. This must be an even number of hexadecimal digits. For example, if the location you wish to verify is 3CC, and the CSECT begins at location 2B0, you can either issue:

```
BASE 02B0
VER 03CC data
```

or you can omit the BASE control record, subtract the CSECT start address from the address of the data, and issue:

```
VER 011C data
```

This also applies to the disp operand of the REP control record.

REP Control Record

The REP control record modifies instructions or data at the specified location within the CSECT that you specified in a preceding NAME control record. The data specified in the REP control record replaces the data at the CSECT location specified by the disp operand. This replacement is on a "one-for-one" basis; that is, one byte of data defined in the control record replaces one byte of data at the location that you specified. If the replacement fails, the program does not perform additional REP operations until it encounters another NAME control record.

The REP control record is optional. More than one REP record can follow a single NAME record.

The format of the REP control record is:

```
REP    disp    data
```

where:

disp — is the hexadecimal displacement of the data to be replaced from the start of the CSECT, if you did not submit a BASE control record for this CSECT. If you did submit a BASE control record, then disp is the actual location of the data. The disp must be two, four, or six hexadecimal digits. This displacement need not address a fullword boundary. If this displacement value is outside the limits of the CSECT being modified, the program does not perform the replacement operation.

data — is the data that is to replace the data in the CSECT. This must be an even number of hexadecimal digits.

Note: Although you do not have to verify a location before replacing data, you should do so to make sure that the data being changed is what you expect it to be.

Comment Control Record

The ZAP program ignores comment control records. If the PRINT option is in effect, the program prints the comments. The format of a comment record is:

```
*  comment
```

You must follow the asterisk with at least one blank.

END Control Record

The END control record ends ZAP processing. The END record is required and must be the last control record. The format of the END control record is:

```
END
```

SPECIAL CONSIDERATIONS FOR USING THE ZAP SERVICE PROGRAM

Before you use the ZAP command against MODULE files, you can use the MODMAP command to determine whether a module map exists and what it contains.

When a ZAP input file has more than one pair of VER and REP control records and a VER control record (other than the first) fails, you must remove the records prior to the failing record and correct the error before you issue the ZAP command again. Otherwise, the file being modified returns to its original status.

If you issue a REP control record against a file that contains an undefined area (for example, a Define Storage area) within the REP data field and do not issue a VER control record prior to the REP control record, the bytes prior to the undefined area, if any, are modified and all the bytes after the undefined area are not modified. The program prints warning message DMSZAP248W.

Appendix I tells you how to apply Program Temporary Fixes (PTFs) and updates to an installed VM/370 system. It contains information about the following:

- Supporting a VM/370 system
- Updating modules using the VMFASM EXEC procedure
- Using the VMFMAC EXEC procedure to update macro libraries
- Using VMFLOAD to generate a new nucleus
- The loader
- Using the GENERATE EXEC procedure to generate a new CP, CMS, or RSCS nucleus, or to load IPCS
- Using the VMFBLD EXEC procedure to build a new nucleus
- Using the CMSGEND EXEC procedure to generate a CMS module
- Using the ASMGEND EXEC procedure to generate the Assembler
- Recommended procedures for updating VM/370

SUPPORTING A VM/370 SYSTEM

The multiple virtual machine environment created by VM/370 permits support of both hardware and software to be done concurrently with other installation work.

Virtual machines can be used to:

- Generate and test new systems
- Apply and test PTFs
- Run hardware diagnostics
- Retrieve and examine VM/370 ABEND dumps and error recordings
- Examine portions of real VM/370 storage
- Trace the execution of a system in a virtual machine

Before installing VM/370, you should develop an account support plan with the IBM FE representative. Appropriately configured virtual machine entries should be included in the VM/370 directory for the service representative. Two virtual machines, with userids CE and MAINT, are defined for these representatives in the VM/370 directory distributed with the starter system.

VM/370 UPDATE PROCEDURES

Using the VM/370 update facility, you can update files with several levels of updates and/or any number of program temporary fixes (PTFs). Procedures are supplied for assembling the updated source code to produce a uniquely identifiable text file. The file has a unique filename and records that identify the origin of the updates, macro libraries, and source statements.

Procedures are provided for generating load files from various object modules, and for generating MACLIB files from various COPY and MACRO files.

The update procedure involves a file naming convention for update and text files, a set of programs to support the processing, and a set of EXEC procedures to process the files.

The update procedures and programs supplied with VM/370 are:

• VMFASM	Incorporates PTFs and/or updates and creates a new text file
• VMFLOAD	Generates a new CP, CMS, or RSCS nucleus
• CMSGEND	Generates a new CMS module
• GENERATE	Generates a new VM/370 system (CP, CMS, or RSCS)
• GENERATE IPLDECK	Generates a new standalone version of a service program on disk
• GENERATE SRVCPGM	Punches the service programs on cards
• GENERATE IPCS	Loads the IPCS modules onto the IPCS disk
• VMFMAC	Generates a new CP, CMS, or RSCS macro library
• CMS UPDATE Command	Updates modules

All modules prefaced by the letters DMK are CP modules. There are two kinds of CP modules: those that are part of the CP nucleus (these modules are contained in the CPLOAD EXEC file) and those that are not part of the CP nucleus (service programs that execute either standalone or under CMS). The programs that execute standalone are DMKDDR, DMKDIR, and DMKFMT. If you apply a PTF to these modules and create a new text deck, use the GENERATE EXEC to create a new standalone file.

The service programs that execute under CMS are:

- DASD Dump Restore Program (module DMKDDR)
- Directory program (module DMKDIR)
- VMFDUMP, the virtual dump program (module DMKEDM)
- NCPDUMP, the 3704/3705 dump program (module DMKRND).

If you apply updates to these modules, use the CMSGEND EXEC to create a new CMS module. The module name to specify is DDR, DIRECT, NCPDUMP, or VMFDUMP, respectively. CMS cannot execute the Format/Allocate program (module DMKFMT) and the IBCDASDI Virtual Disk Initialization program (module IBCDASDI). If you apply a PTF to any DMK module other than these six service programs, you must reload CP (using the VMFLOAD command).

UPDATING A MODULE

The following discussions assume that areas containing the source code for CP and CMS are added to the appropriate virtual machine configurations. Source code for the CMS system is included on the CMS2 tape; source code for CP is included on the CP2 tape. These tapes are distributed by the Program Information Department (PID).

VM/370 has update procedures to incorporate changes (additions, deletions, or corrections) into an existing module, macro, or copy file. For example, if you apply updates to the DMKVAT module, the VMFASM update procedure:

- Locates the DMKVAT source file. It is the unmodified Assembler language source code that is distributed with the VM/370 system.

- Locates the update control file for DMKVAT. The control file name is specified on the VMFASM command. The control file can have any filename, but must have a filetype of CNTRL. It contains records indicating how to apply the updates.

- Applies the updates to the specified source (in this case, DMKVAT) and gives the updated source a temporary name by concatenating a $ (dollar sign) to the first seven characters of the filename (in this case, the temporary filename is $DMKVAT).

- Puts macro library names specified in the control file into the proper Assembler library list. (The macro libraries required at assembly time are specified in a MACS record in the control file.)

- Assembles the updated source file ($DMKVAT) and creates an updated object deck. The object deck filetype is derived from information found in the control file. The filename of the updated object file is the same as that of the original source, DMKVAT.

As the VMFASM update procedure progresses from one step to the next, informational or error messages are displayed. To more fully understand how the update procedures operate, you need to know what a control file contains, and how it is handled. The following discussion provides this information.

CONTROL FILES

The CMS UPDATE command and the VMFASM EXEC procedure use control files. You may have one or more control files to specify various combinations of updates and macro libraries to be used at different times. A control file contains the following types of records:

- The MACS Record -- This record contains the names of macro libraries to be used at assembly time. A MACS record is required for a control file used by the CMS UPDATE command; it is optional for a control file used by the VMFLOAD EXEC procedure. A control file must not contain more than one MACS record. If a MACS record is included, it must precede any other records except comments. Up to eight libraries may be specified in the MACS record (if space permits). A MACS record has the following format:

  ```
  uplevel MACS lib1 lib2 lib3...
  ```

  The uplevel field is usually TEXT; it is not used to generate the filetype of the updated file.

- Update identification records--These records identify updates that are to be applied to a particular source file, if such a file exists. An update identification record has the following format:

  ```
  uplevel upid
  ```

  where uplevel is an update level identifier that generates a filetype for the new file (the new filetype uniquely identifies the updated source file). The field, upid, is the update identification; it can be from one to four characters long. This update identification is concatenated with the prefix UPDT to identify the filetype of the direct update to be applied. The filename must be the same as the name of the file to be updated. For example, if an update identification record for DMKVAT is:

  ```
  TEXT NEW1
  ```

  the update file is called DMKVAT UPDTNEW1.The update level identifier is TEXT.

- AUX file identification records--These records contain the names of auxiliary files (AUX files), which in turn contain a list of filetypes of update files to be applied to a particular source file. An auxiliary file identification record has the following format:

  ```
  uplevel AUXnnnn
  ```

  where uplevel is an update level identifier that generates the filetype of the updated source file. The string, nnnn, is an identification string that can be from one to four characters long. This identification string, with the prefix AUX, is the filetype of the auxiliary file that contains the list of updates to be applied. The filename of the auxiliary file is the same as the filename of the source file to be updated. For example, if an AUX file identification record that contains a list of updates for DMKVAT is:

  ```
  TEXT AUXnnnn
  ```

  then the auxiliary file called DMKVAT AUXnnnn contains the filetypes of the update files that are to be applied. These update files have the same filename as the source file to be updated.

- Comments--An asterisk (*) in the first column of the record identifies a comment record.

Note: Control file records in the format that was used under VM/370 Release 1 or 2 are still accepted under Release 3. For example, an AUX file identification record in the format

```
TEXT nnnn AUX
```

is still accepted by the update procedures.

A control file can have many update identification records, AUX file identification records, and comments, but can have only one MACS record. The control file can have any filename. Note, however, that VM/370 updates from IBM normally use the following special control files:

- A control file for CP source, copy, and macro updates is called DMKR30 CNTRL. The DMKR30 CNTRL file contains the following records:
  - TEXT MACS DMKMAC CMSLIB OSMACRO
  - TEXT AUXR30
- A control file for CMS source updates is called DMSR30 CNTRL. The DMSR30 CNTRL file contains the following records:
  - TEXT MACS CMSLIB OSMACRO
  - TEXT AUXR30
- A control file for CMS macro and copy updates is called DMSM30 CNTRL. The DMSM30 CNTRL file contains the following record:
  - COPY AUXM30
- A control file for assembling the NCPDUMP source is called NCPR30 CNTRL. The NCPR30 CNTRL file contains the following records:
  - TEXT MACS OSMACRO DMKMAC CMSLIB
  - TEXT AUXR30
- A control file for assembling RSCS source, copy, and macro updates is called DMTR30 CNTRL. The DMTR30 CNTRL file contains the following records:
  - TEXT MACS DMTLOC DMTMAC
  - TEXT AUXR30

APPLYING PTFS TO VM/370

PTF updates are distributed in card or magnetic tape form, or as APAR answers typed on coding forms. In any case, a CMS file (with the correct filename and filetype) must be created on a disk to contain the update. The disk must belong to the user (userid MAINT) who is responsible for updating VM/370. The disk may be the CP or CMS source disk, but it is usually a separate disk.

A suggested virtual machine configuration for updating a 2314 system is:

```
USER MAINT CPCMS 720K 16M BCEG
  ACCOUNT (installation defined)
    OPTION ECMODE REALTIMER
    CONSOLE   009   3215
    SPOOL     00C   2540   READER A
    SPOOL     00D   2540   PUNCH  A
    SPOOL     00E   1403   A
    MDISK  190 2314 035 110 CPV3L0 MR READ
    MDISK  191 2314 019 010 CPV3L0 WR READ
    MDISK  194 2314 145 058 CPV3L0 MR READ
    MDISK  199 2314 034 001 CPV3L0 WR READ
    MDISK  193 2314 001 050 USERD1 MR READ
    MDISK  294 2314 051 050 USERD1 MR READ
    MDISK  393 2314 001 110 USERD2 MR READ
    MDISK  394 2314 001 110 USERD3 MR READ
    MDISK  390 2314 101 003 USERD1 MW READ
    MDISK  cuu 2314 000 203 yyyyyy MW
```

where cuu and yyyyyy are the address and label of your system residence volume defined in your DMKSYS module.

A suggested virtual machine configuration for updating a 3330 system is:

```
USER MAINT CPCMS 720K 16M BCEG
  ACCOUNT (installation defined)
    OPTION ECMODE REALTIMER
    CONSOLE   009   3215
    SPOOL     00C   2540   READER A
    SPOOL     00D   2540   PUNCH  A
    SPOOL     00E   1403   A
    MDISK  190 3330 030 076 CPV3L0 MR READ
    MDISK  191 3330 016 007 CPV3L0 WR READ
    MDISK  194 3330 106 044 CPV3L0 MR READ
    MDISK  199 3330 029 001 CPV3L0 WR READ
    MDISK  193 3330 001 030 USERD1 MR READ
    MDISK  294 3330 031 030 USERD1 MR READ
    MDISK  393 3330 061 060 USERD1 MR READ
    MDISK  394 3330 121 060 USERD1 MR READ
    MDISK  390 3330 181 002 USERD1 MW READ
    MDISK  cuu 3330 000 404 yyyyyy MW
```

where cuu and yyyyyy are the address and label of your system residence volume defined in your DMKSYS module.

A suggested virtual machine configuration for updating a 3340 system is:

```
USER MAINT CPCMS 720K 16M BCEG
  ACCOUNT (installation defined)
    OPTION ECMODE REALTIMER
    CONSOLE   009   3215
    SPOOL     00C   2540   READER A
    SPOOL     00D   2540   PUNCH  A
    SPOOL     00E   1403   A
    MDISK 190 3340 048 203 CPV3L0 MR READ
    MDISK 191 3340 026 015 CPV3L0 WR READ
    MDISK 194 3340 251 098 CPV3L0 MR READ
    MDISK 199 3340 046 002 CPV3L0 WR READ
    MDISK 193 3340 001 090 USERD1 MR READ
    MDISK 294 3340 031 090 USERD1 MR READ
    MDISK 393 3340 061 180 USERD1 MR READ
    MDISK 394 3340 121 180 USERD1 MR READ
    MDISK 390 3340 181 006 USERD1 MW READ
    MDISK cuu 3340 000 348 yyyyyy MW
```

where cuu and yyyyyy are the address and label of your system residence volume defined in your DMKSYS module.

The entries in the preceding VM/370 directory, with the exception of the 193, 294, 393, 394, and 390 virtual disks, are in the 2314, 3330, and 3340 VM/370 directories supplied with the starter system, and should be included in your VM/370 directory, because IBM uses them for support.

The contents of the preceding virtual disks are:

Disk	Contents
190	Current CMS system disk
191	Work area
194	CP and RSCS text retention
199	The 191 minidisk (work area)
193	CMS PTFs, updates, and updated text decks (object modules)
294	CP and RSCS PTFs, updates, and updated text decks (object modules)
393	CMS source and macros
394	CP and RSCS source, macros, and copy files
390	CMS test nucleus area
cuu	CP system residence device, or a replica of it, for test purposes

These virtual disks are shown in Figure 69.

You should apply all distributed updates. Once you create the appropriate files, you should access the disks containing the CP, CMS, or RSCS source files and update procedures, and apply the updates.

To apply the IBM distributed updates to an existing source file, use the VMFASM EXEC procedure. To apply the IBM distributed updates to a copy or macro file, use the VMFMAC EXEC procedure.

If you update a copy or macro file, you should use the VMFASM EXEC procedure to reassemble the module(s) that contain that copy or macro file.

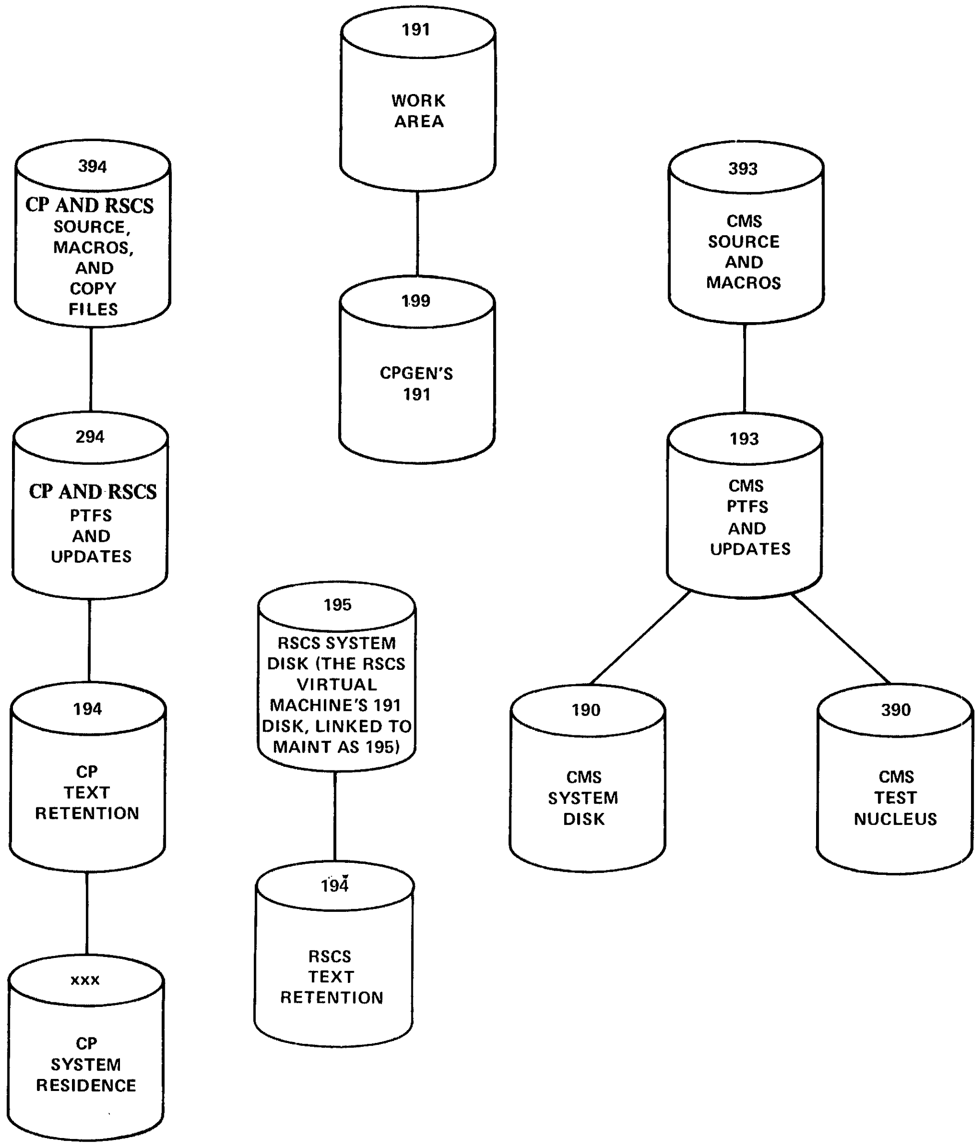


Figure 69. System Support Plan

UPDATING MODULES USING THE VMFASM EXEC PROCEDURE

Use the VMFASM EXEC procedure to update a specified source file according to entries in a control file, and to assemble the updated source file. VMFASM invokes the CMS UPDATE command. The format of the VMFASM command is:

```
┌──────────────────────────────────────────────────────────────────────┐
│ VMFASM │ fn1 fn2 [(options...[)]]                                    │
│        │                                                             │
│        │ Options:                                                    │
│        │ ┌      ┐ ┌      ┐ ┌      ┐                                  │
│        │ │DISK  │ │TERM  │ │LIST  │                                  │
│        │ │PRINT │ │NOTERM│ │NOLIST│                                  │
│        │ └      ┘ └      ┘ └      ┘                                  │
│        │ ┌      ┐ ┌      ┐                                           │
│        │ │DECK  │ │RENT  │ [EXP] [XREF]                              │
│        │ │NODECK│ │NORENT│                                           │
│        │ └      ┘ └      ┘                                           │
└──────────────────────────────────────────────────────────────────────┘
```

where:

fn1 is the filename of the source file to be updated.

fn2 is the filename of the control file. The control file must have a filetype of CNTRL.

Options:

DISK places the LISTING file on a virtual disk.

PRINT writes the LISTING file to the printer.

TERM writes the diagnostic information on the SYSTERM data set. The diagnostic information consists of the diagnosed statement followed by the error message issued.

NOTERM suppresses the TERM option.

LIST produces an Assembler listing.

NOLIST does not produce an Assembler listing.

DECK writes an object module on the device specified on the FILEDEF statement for PUNCH.

NODECK suppresses the DECK option.

RENT checks the program for a possible violation of program reenterability. Code that makes the program nonreenterable is identified by an error message.

NORENT suppresses the RENT option.

EXP expands printing of certain macros which check for the SUP parameter issued via the SYSPARM option of the assembler.

XREF causes the XREF(SHORT) option to be invoked when VMFASM invokes the assembler.

Note: VMFASM only accepts the non-defaulted options. All other options entered are ignored and the defaults are used.

USING VMFASM TO APPLY IBM-SUPPLIED UPDATES

The control file contains records that identify the updates to be applied and the macro libraries, if any, needed to assemble the source program. The updates are applied starting with the last update file in the control file and in sequence up to the first update file (or the update file immediately following the MACS record). Updates identified by auxiliary files are applied starting with the last update in the auxiliary file and proceeding in sequence up to the first.

For example, a control file named UPDATE1 CNTRL contains the following two records:

```
TEXT   MACS   DMKMAC   CMSLIB   OSMACRO
IBM1   AUX2000
```

An Assembler language source file is named DMKVAT ASSEMBLE.

An auxiliary file named DMKVAT AUX2000 contains a list of filetypes (NEW2 and NEW1) with NEW2 the first entry and NEW1 the last.

The two update files are named DMKVAT NEW1 and DMKVAT NEW2. These are the files identified by the auxiliary file, DMKVAT AUX2000. Assume these files contain IBM-supplied updates to DMKVAT, such as inserted, deleted, or replaced source statements and the appropriate control statements. The update control statements are described in the _VM/370: CMS Command and Macro Reference_ with the CMS UPDATE command.

To update DMKVAT, you enter the command:

```
VMFASM DMKVAT UPDATE1
```

VMFASM does the following:

- VMFASM locates the DMKVAT ASSEMBLE and UPDATE1 CNTRL files.
- The UPDATE1 CNTRL file is processed from the bottom up. The first entry found is IBM1 AUX2000.
- VMFASM tries to locate the file named DMKVAT AUX2000 by searching all accessed disks.
- When VMFASM locates the DMKVAT AUX2000 file, it processes it from the bottom up. The first entry found is NEW1.
- VMFASM tries to locate the the update file named DMKVAT NEW1. When it finds DMKVAT NEW1, VMFASM applies the updates that are in DMKVAT NEW1 to the DMKVAT ASSEMBLE file, and creates a new file called $DMKVAT ASSEMBLE.
- Next, VMFASM processes the NEW2 entry in the DMKVAT AUX2000 file. When VMFASM locates the update file DMKVAT NEW2, it applies the updates to the updated ASSEMBLE file ($DMKVAT).
- Because there are no more filetypes listed in the DMKVAT AUX2000 file, VMFASM reads the next control record in the UPDATE1 CNTRL file. In this case, it is the MACS record.

- After entering the macro library names that are on the MACS record into the appropriate Assembler library lists, VMFASM assembles the updated ASSEMBLE file ($DMKVAT).

- The UPDATE command then stacks in the console read buffer the uplevel (update level identifier) associated with the last update applied. If there were no updates, it stacks the uplevel associated with the MACS control record and the names of the macro libraries specified in the MACS record. VMFASM then reads the stacked lines and concatenates the uplevel (if it is not TEXT) to the characters TXT to form the filetype of the assembled updated source.

  An update level identifier of TEXT causes special handling in the VMFASM EXEC procedure, whether or not an update is used with it. A name of TEXT is used as the object module filetype without level identification concatenation. Thus, TEXT becomes the filetype.

  VMFASM places the macro library names that were specified cn the MACS record in the Assemble library list (via the CMS GLOBAL command) so that those libraries can be used when the updated source file is assembled.

  In this example, the last (and only) update applied was identified as IBM1 AUX2000. The file identification for the updated source is DMKVAT TXTIBM1. The updated source is assembled using the macro libraries DMKMAC, CMSLIB, and OSMACRO.

You may want, on occasion, to have entries in a control file that specify an update level identifier but no update. A record of the following format, for example, is allowed:

```
NAME5
```

because the control file is used for loading object modules (text decks) as well as for updating input files.

If updates are not found, a message is issued and processing continues, if possible.

USING VMFASM TO APPLY YOUR OWN UPDATES

If you wish to apply your own update to VM/370 (for example, if you wish to expand the accounting routines), you follow the same procedure described for applying IBM-supplied updates.

You create the update file. You can name the update file in either of two ways. If you are going to identify the update file directly in the control file, use the form:

```
DMKACO UPDTupid
```

where DMKACO is the filename of the accounting module you wish to expand, and upid is the identification for the filetype. For example, you might call your update file:

```
DMKACO UPDTFIX1
```

The second way to identify your update is via an auxiliary file. If you use an auxiliary file, you use the following form to name your update file:

```
DMKACO ft
```

where DMKACO is the filename of the accounting routine you wish to expand and ft is any filetype.

For example, you could have two update files called:

```
DMKACO NEW1
DMKACO NEW2
```

When you decide to use an auxiliary control file, it must have a name in the form

```
DMKACO AUXnnnn
```

For example, assume you have an auxiliary control file called:

```
DMKACO AUX1111
```

This AUX file has the following entries (the filetypes of your update files):

```
NEW2
NEW1
```

Next, you must create a control file to identify all IBM-supplied updates to the module you are changing and your own updates. You must apply the IBM-supplied updates first. Assume there are IBM-supplied updates in an auxiliary file called:

```
DMKACO AUXR30
```

and that your own updates are those used as examples in the preceding paragraphs. Then, you need your own control file, identified as:

```
fn CNTRL
```

It can have any filename, but its filetype must be CNTRL. For this example, the control file is called:

```
LCC CNTRL
```

and it has the following records:

```
TEXT MACS DMKMAC
LOCAL FIX1
SPEC AUX1111
IBM1 AUXR30
```

To apply the updates to DMKACO, issue the command:

```
VMFASM DMKVAT LOC
```

The VMFASM procedure handles the update as follows; it:

- Locates the source file, DMKACO.
- Locates the control file, LOC CNTRL.
- Reads the control file, last line first (IBM1 AUXR30).

- Locates the IBM-supplied auxiliary file, DMKACO AUXR30.
- Reads the DMKACO AUXR30 auxiliary file from bottom to top and applies the IBM-supplied updates to DMKACO, naming the updated source $DMKACO ASSEMBLE.
- Reads the next entry in the control file (SPEC AUX1111).
- Locates your own auxiliary file, DMKACO AUX1111.
- Reads the last entry (NEW1), locates the update file DMKACO NEW1, and applies the update to the updated source $DMKACO.
- Reads the next entry in your auxiliary file (NEW2), locates the corresponding update file DMKACO NEW2, and applies it to the updated source $DMKACO. Processing for your auxiliary file is now complete.
- Reads the next entry in the control file (LOCAL FIX1).
- Locates the directly-identified update file, DMKACO UPDTFIX1.
- Applies the DMKACO UPDTFIX1 updates to the updated source ($DMKACO).
- Reads the next control record, the MACS record.
- Issues the GLOBAL command for the macro libraries identified on the MACS record (DMKMAC MACLIB).
- Assembles the updated source file, $DMKACO ASSEMBLE.
- Names the object module DMKACO TXTLOCAL. The filetype is derived by concatenating the prefix (TXT) and the uplevel of the last update applied (LOCAL).

If you create a new object module for a VM/370 module, you must also reconstruct the CP, CMS, or RSCS nucleus using the VMFLOAD service program. Or, if the file is a part of a CMS command module, use the CMSGEND procedure to generate a new module utilizing the new object code. See Appendix D to determine whether the CMS nucleus or some other MODULE files must be generated again because of your update.

When you use CMSGEND to create a new module, you must change the filetype of the object deck to TEXT (if it is not already TEXT) before issuing the CMSGEND command. After the CMSGEND processing is complete, you can change the filetype of the object deck back to what it was before.

Note that CMSGEND renames the existing module to "fname MODOLD" before creating the updated module. This ensures that any users currently using the CMS system do not have their processing interrupted by the updating of modules, because the SSTAT (system status table) of the loaded system is still pointing to the area on the system disk occupied by the renamed module. When the system is reloaded, the SSTAT points to the updated module, and the old module can be erased.

OTHER FILES PRODUCED BY VMFASM

VMFASM invokes the UPDATE command, which produces two output files that indicate which updates were applied. The file

fn UPDATES

lists the names of update files that were applied to the source file (fn) and the file

fn UPDLOG

lists the actual updates that were applied to the source file (fn) and error messages. Both of these files are included in the LISTING file, and precede the program source statements. Also, the file (fn UPDLOG) precedes the object code in the text file.

Trim Along This Line

READER'S COMMENT FORM

Title: IBM Virtual Machine Facility/370: System Logic and Problem Determination Guide

Order No. SY20-0885-0

Please check or fill in the items; adding explanations/comments in the space provided.

Which of the following terms best describes your job?

☐ Customer Engineer	☐ Manager	☐ Programmer	☐ Systems Analyst
☐ Engineer	☐ Mathematician	☐ Sales Representative	☐ Systems Engineer
☐ Instructor	☐ Operator	☐ Student/Trainee	☐ Other (explain below)

How did you use this publication?

☐ Introductory text ☐ Reference manual ☐ Student/ ☐ Instructor text

☐ Other (explain) ______________________________

Did you find the material easy to read and understand? ☐ Yes ☐ No (explain below)

Did you find the material organized for convenient use? ☐ Yes ☐ No (explain below)

Specific criticisms (explain below)

Clarifications on pages ______________________________

Additions on pages ______________________________

Deletions on pages ______________________________

Errors on pages ______________________________

Explanations and other comments:

Thank you for your cooperation. No postage necessary if mailed in the U.S.A.

SY20-0885-0

Your views about this publication may help improve its usefulness; this form will be sent to the author's department for appropriate action. Using this form to request system assistance and/or additional publications or to suggest programming changes will delay response, however. *For more direct handling of such requests, please contact your IBM representative or the IBM Branch Office serving your locality.* Your comments will be carefully reviewed by the person or persons responsible for writing and publishing this material. All comments or suggestions become the property of IBM

FOLD FOLD

FIRST CLASS
PERMIT NO. 172
BURLINGTON, MASS.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

POSTAGE WILL BE PAID BY

IBM CORPORATION
VM/370 PUBLICATIONS
24 NEW ENGLAND EXECUTIVE PARK
BURLINGTON, MASS. 01803

FOLD FOLD

IBM

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)

Trim Along This Line

SY20-0885-0

IBM VM/370: Sys. Logic and Prob. Determination Guide Printed in U.S.A. SY20-0885-0

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)