File No. S370-37
Order No. GC20-1807-4

Systems

# IBM Virtual Machine Facility/370: System Programmer's Guide

**Release 3 PLC 1**

This publication is intended for VM/370 system programmers. The first three parts comprise:

- Detailed descriptions of procedures, commands, and service programs useful in debugging as well as guidelines for reading dumps.
- A description of CP and how it works and details of how to better utilize CP.
- A description of CMS and how it works, as well as details of some special features of CMS.

The last two parts describe VM/370 teleprocessing support:

- The 3704/3705 Communications Controllers
- The Remote Spooling Communications Subsystem (RSCS).

IBM

| Fifth Edition (February 1976)

| This is a major revision of GC20-1807-3 and makes that edition and
| Technical Newsletter GN30-2662, dated March 31, 1975, obsolete. This
| edition corresponds to Release 3 PLC 1 (Program Level Change) of IBM
Virtual Machine Facility/370 and to all subsequent releases until
otherwise indicated in new editions or technical newsletters.

Changes are periodically made to the specifications herein; before using this publication in connection with the operation of IBM systems, consult the latest IBM System/370 Bibliography, Order No. GC20-0001, for the editions that are applicable and current.

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

Requests for copies of IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form for readers' comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, VM/370 Publications, 24 New England Executive Park, Burlington, Massachusetts 01803. Comments become the property of IBM.

© Copyright International Business Machines Corporation 1972, 1973, 1974, 1975, 1976

# Preface

This publication describes how to debug VM/370 and how to modify, extend or implement Control Program (CP) and Conversational Monitor System (CMS) functions. This information is intended for system programmers, system analysts, and program personnel.

This publication consists of five parts and two appendixes.

"Part 1. Debugging with VM/370" discusses the CP and CMS debugging tools and procedures to follow when debugging. This part is logically divided into three topics. The first section "Introduction to Debugging" tells you how to identify a problem and lists guidelines to follow to find the cause. The second section "Debugging with CP" describes the CP debugging commands and utilities, debugging CP in a virtual machine, the internal trace table and restrictions. A detailed description of CP dump reading is also included. The third section "Debugging with CMS" describes the CMS debugging commands and utilities, load maps, and restrictions and tells you what fields to examine when reading a CMS dump.

"Part 2. Control Program (CP)" contains an introductory and functional description of CP as well as guidance in implementing some CP features.

"Part 3. Conversational Monitor System (CMS)" contains an introductory and functional description of CMS including how CMS handles interrupts and SVC calls, structures its nucleus and its storage, and manages free storage. Information on saving the CMS system and implementing the Batch Facility is also included.

"Part 4. IBM 3704 and 3705 Communications Controllers" describes the functions and uses of these programmable units. Information is included on loading, testing, and updating the control program.

"Part 5. Remote Spooling Communications Subsystem (RSCS)" describes the functions and uses of the component of VM/370 that handles the transmission of files between VM/370 users and remote programmable and nonprogrammable stations.

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

"Appendix B: MULTI-LEAVING" provides a detailed description of MULTI-LEAVING[1], a computer-to-computer communications technique developed for use by the HASP system and used by the RSCS component of VM/370.

In this publication, the following terminology is used:

- 2305 refers to the IBM 2305 Fixed Head Storage, Models 1 and 2.
- 3270 refers to both the IBM 3275 Display Station, Model 2 and the IBM 3277 Display Station, Model 2.
- 3330 refers to the IBM 3330 Disk Storage Models 1, 2, 11; the IBM 3333 Disk Storage and Control Models 1 and 11; and the 3350 Direct Access Storage operating in 3330/3333 Model 1 or 3330/3333 Model 2 compatibility mode.
- 3340 refers to the IBM 3340 Disk Storage, Models A2, B1 and B2; and, the 3344 Direct Access Storage, Model B2.
- 3350 refers to the IBM 3350 Direct Access Storage, Models A2 and B2, in native mode.

Any information pertaining to the IBM 2741 terminal also applies to the IBM 3767 terminal, Model 1, operating as a 2741, unless otherwise specified.

Information on the IBM 3344 Direct Access Storage Device and the IBM 3350 Direct Access Storage contained in this publication is for planning purposes only until the availability of the product.

An expanded glossary is available in the IBM Virtual Machine Facility/370: Glossary and Master Index, Order No. GC20-1813.

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

---

[1] Trademark of IBM

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

PREREQUISITE PUBLICATIONS

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

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

IBM OS/VS and VM/370 Assembler Programmer's Guide, GC33-4021.

IBM OS/VS, DOS/VS, and VM/370 Assembler Language, GC33-4010.

IBM Virtual Machine Facility/370: Operating Systems in a Virtual Machine, Order No. GC20-1821.

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

COREQUISITE PUBLICATIONS

IBM Virtual Machine Facility/370:

Planning and System Generation Guide, Order No. GC20-1801

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

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

CMS User's Guide, Order No. GC20-1819.

Operator's Guide, Order No. GC20-1806

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

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

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

System Logic and Problem Determination Guide, Order No. SY20-0885

Note: References in text to titles of corequisite VM/370 publications will be given in abbreviated form.

OS/VS Data Management Macro Instructions, Order No. GC26-3793.

OS/VS Supervisor Service and Macro Instructions, Order No. GC27-6979.

IBM 2821 Control Unit Component Description Order No. GA24-3312.

IBM 3211 Printer, 3216 Interchangeable Train Cartridge, and 3811 Printer Control Unit Component Description and Operator's Guide, Order No. GA24-3543.

IBM OS/VS Linkage Editor and Loader, Order No. GC26-3813.

Introduction to the IBM 3704 and 3705 Communications Controllers, Order No. GA27-3051.

IBM 3704 and 3705 Communications Controllers Operator's Guide, Order No. GA27--3055.

If the IBM 3767 Communication Terminal is used by the system programmer as a virtual machine console, the IBM 3767 Operator's Guide, Order No. GA18-2000 is also a corequisite publication.

## CMS SUPPORTS VSAM FUNCTIONS

New: Program Feature

CMS supports the Virtual Storage Access Method (VSAM), providing indexed file capability to high-level language programs executing under CMS, providing data compatability with DOS/VS and OS/VS, and providing enlarged file capacity for CMS end-user applications.

CMS VSAM support is provided via the DOS/VS VSAM component of the DOS/VS supervisor and I/O functions.

This support is described in the section "CMS Support for OS and DOS VSAM Functions."

## CMS SUPPORTS DOS FUNCTIONS IN THE CMS/DOS ENVIRONMENT

New: Program Feature

CMS allows the execution of many DOS programs by simulating DOS/VS supervisor and I/O functions. In the CMS/DOS environment CMS appears to DOS programs as a single-partition non-multitasking DOS/VS system.

This support is described in the section "DOS/VS Support Under CMS."

## CMS EXTENDS ITS USE OF SHARED SEGMENTS

New: Program Feature

CMS extends its use of named saved systems to include support for one or more discontiguous saved segments which can be shared among all virtual machines and which will be outside the address space currently defined for each of the sharing virtual machines.

This support is described in the section "Generating Saved Systems."

## CP SUPPORTS NEW DIAGNOSE INSTRUCTION CODES

New: Program Feature

CP now supports DIAGNOSE instruction codes X'60' and X'64'. DIAGNOSE code X'60' allows you to edit the text of error messages. DIAGNOSE code X'64' provides subcodes that allow you to control the use of discontiguous saved segments. The general functions of these subcodes are to load, release, or find the address of a discontiguous saved segment.

This support is described in the section "DIAGNOSE Instruction in a Virtual Machine."

## CP EXTENDS ITS PROTECTION OF SHARED SEGMENTS

New: Program Feature

CP protection of shared segments is extended to allow the virtual machine assist hardware to be activated for virtual machines running named shared systems, thus allowing shared CMS systems to take advantage of the performance improvements of the virtual machine assist feature.

This support is described in the section "Shared Segment Protection."

## CP PROVIDES IMPROVED SPOOL FILE RECOVERY

New: Program Feature

CP recovery of spool files is extended to allow recovery after system or hardware failure.

This support is described in the section "Spool File Recovery."

## VM/370 MEASUREMENT FACILITY

New: Program Feature

A new commmand (INDICATE) and an expansion of the MONITOR command provide a way to dynamically measure system performance. The general user can have displayed, at his terminal, certain certain system load conditions and his virtual system's usage of system resources. The system analyst can sample and record a wide variety of system load data, I/O activity, resource utilization, response data and simulation data.

A new section, "Performance Observation and Analysis" has been added to "Part 2: Control Program (CP)."

## VM/VS HANDSHAKING FEATURE

New: Program Feature

The VM/VS Handshaking feature is a communication path between VM/370 and OS/VS1 that makes each system control program aware of certain capabilities and requirements of the other. The following changes to this manual reflect this support:

- A new operand, PAGEX, is added to the CP SET command in "Part 1: Debugging with VM/370."
- A new section, "VM/VS Handshaking" is added to "Part 2: Control Program (CP)."
- A new Diagnose code 0 is added to the "Diagnose Instruction in a Virtual Machine" section in "Part 2: Control Program (CP)."

## IBM 3270 REMOTE SUPPORT

New: Program Feature

VM/370 now supports the IBM 3270 Information Display System as a remote virtual machine console attached via nonswitched point-to-point lines to a 2701 Data Adapter Unit, 2703 Transmission Control Unit, or a 3704/3705 Communications Controller in emulation mode. The remote 3270 user also has the capability of copying an entire screen display on a 3284, 3286, or 3288 printer at the remote location.

The following changes to this manual reflect this new support:

- A new operand, PFnn COPY is added to the CP SET command in "Part 1: Debugging with VM/370."
- The section on "CP Restrictions" is updated to include restrictions to this new support.
- "Figure 11. CP Control Block Relationships" is updated.
- "Part 4: IBM 3704 and 3705 Communications Controllers" is updated to include remote 3270 support.

## NEW OPERANDS FOR SET COMMAND

New: Program Feature

Two new operands to the SET command described in "Part 1: Debugging with VM/370" allow the virtual machine user to enable and disable the ECMODE and/or ISAME options, dynamically.

## USER FORMATTED ACCOUNTING RECORDS

New: Program Feature

A virtual machine user may now initiate the punching of an accounting card containing up to 70 bytes of data, the content and format of which he can determine. The fcllowing changes to this manual reflect this support:

- "Accounting Records for Virtual Machine Users" in "Part 2: Control Program (CP)" is updated to describe the implementation of this support.
- The section "Diagnose Instruction in a Virtual Machine" in "Part 2: Control Program (CP)" is updated to expand the function of Diagnose code X'4C' to include this new support.

## NEW DEVICE SUPPORT

New: Program Feature

The 3340 Direct Access Storage Facility is now supported by VM/370. This support includes:

- 3348 Data Module, Models 35 and 70
- Rotational Position Sensing
- Fixed Head Feature

This device support is reflected in the following changes to this publication:

- "Figure 12. CP Device Classes, Types, Models and Features" is updated.
- The INPUT AND OUTPUT control statements for the DASD Dump Restore Program, described in "Part 2: Control Program (CP)," are changed.

New: Documentation Only

VM/370 support for the IBM 3767 Communications Terminal (at 300bps) as an IBM 2741 Communications Terminal is reflected in an update to "Figure 12. CP Device Classes, Types, Models, and Features".

## NEW VM/370 COMPONENT

New: Program Feature

The Remote Spooling Communications Subsystem (RSCS) has been included as a component of the VM/370 system. Together with the Control Program (CP) of VM/370, it manages telecommunication I/O devices and lines used to automatically transfer files between:

- VM/370 users and remote stations.
- Remote stations and other remote stations.
- VM/370 users and remote HASP/ASP type batch systems.
- Remote stations and remote HASP/ASP type batch systems.
- Remote stations and a CMS Batch virtual machine.

The addition of this new component is reflected in the fcllowing changes and additions to this publication:

- The Spooling Functions" section in "Part 2: Control Program (CP)" has been updated to include the remote spooling capabilities of RSCS and the addition of the spool file tag field to all output spool files.
- The "Diagnose Instruction in a Virtual Machine" section in "Part 2: Control Program (CP)" has been updated to include a new subfunction code X'0FFF' to Diagnose code 14. RSCS uses this new option to retrieve spool file block and tag data for files that it is to process for transmission.
- The "CMS Batch Facility" section in "Part 3: Conversational Monitor System (CMS) has been updated to include remote job entry via RSCS.
- "Part 5: Remote Spooling Communications Subsystem (RSCS)" has been added to provide the system programmer with pertinent information on the new component of VM/370.

## CMS USERS CAN READ DOS FILES

New: Program Feature

CMS now supports the reading of DOS files as well as OS data sets. This support is described in the "OS Data Management Simulation" section of "Part 3: Conversational Monitor System (CMS)". The "VM/370: Restrictions" section in "Part 1: Debugging with VM/370" is updated to remove the restriction against reading DOS files.

## ENHANCEMENTS TO THE VIRTUAL MACHINE

New: Program Feature

Programs such as DOS/VS, VS1 and VS2 that use block multiplexer channel operations can now be run under VM/370 in virtual block multiplexer mode. The mode of operation for all channels, except channel 0 and any channel to which a channel-to-channel Adapter (CTCA) is attached, is selectable via a DIRECTORY option or the DEFINE Command.

This new feature is described under "Functional Information" in "Part 2: Control Program (CP)".

## PUBLICATION CONTENT CHANGED

Changed: Documentation Only

Information on planning considerations and generation of the 3704/3705 control program, formerly in "Part 4: IBM 3704 and 3705 Communications Controllers" has been moved to the VM/370: Planning and System Generation Guide.

The information about generating and testing the standalone program that controls the 2780 formerly in "Part 5: IBM 2780 Data Transmission Terminal" has been moved to the VM/370: Planning and System Generation Guide.

## MISCELLANEOUS CHANGES

Maintenance: Program and Documentation

Two new ABEND codes, PGT008 and PRG019, have been added to "Figure 10. CP ABEND Codes". Many other changes, too numerous to detail, have also been included in this publication.

# Contents


PART 1. DEBUGGING WITH VM/370. . . . . 11

INTRODUCTION TO DEBUGGING. . . . . 13
How To Start Debugging . . . . . 13
  Does a Problem Exist? . . . . . 14
  Identifying the Problem. . . . . 16
  Analyzing the Problem. . . . . 22
How To Use VM/370 Facilities To Debug. . 26
  ABEND. . . . . 26
    CP ABEND . . . . . 26
    CP Termination without a Dump. . . . 27
    CMS ABEND. . . . . 28
    Virtual Machine ABEND (Other than CMS). . . . . 31
  Unexpected Results . . . . . 32
    Unexpected Results in CP . . . . . 32
    Unexpected Results in a Virtual Machine . . . . . 32
  Loop . . . . . 33
    CP Disabled Loop . . . . . 33
    Virtual Machine Disabled Loop. . . . 34
    Virtual Machine Enabled Loop . . . . 34
  WAIT . . . . . 35
    CP Disabled Wait . . . . . 35
    CP Enabled Wait. . . . . 36
    Virtual Machine Disabled Wait. . . . 36
    Virtual Machine Enabled Wait . . . . 37
    RSCS Virtual Machine Disabled Wait . 37
    RSCS Virtual Machine Enabled Wait. . 38
Summary of VM/370 Debugging Tools. . . . 39
Comparison of CP and CMS Facilities for Debugging . . . . . 44

DEBUGGING WITH CP. . . . . 45
CP Commands Used To Debug in the Virtual Machine . . . . . 45
ADSTOP . . . . . 46
BEGIN. . . . . 48
DISPLAY. . . . . 49
DUMP . . . . . 55
SET. . . . . 58
STORE. . . . . 65
SYSTEM . . . . . 69
TRACE. . . . . 70
CP Commands for System Programmers and System Analysts . . . . . 75
DCP. . . . . 76
DMCP . . . . . 79
LOCATE . . . . . 82
MONITOR. . . . . 84
QUERY. . . . . 85
SAVENCP. . . . . 86
SAVESYS. . . . . 87
STCP . . . . . 88
DASD Dump Restore Program (Standalone Version) . . . . . 89
  DDR Control Statements . . . . . 89
    I/O Definition Statements. . . . . 89
    INPUT/OUTPUT Control Statement . . . 90
    SYSPRINT Control Statement . . . . . 91
    Function Statement . . . . . 91
    PRINT/TYPE Function Statement. . . . 95
Debugging CP on a Virtual Machine. . . . 96
CP Internal Trace Table. . . . . 96
CP Restrictions. . . . . 99
Dynamically Modified Channel Programs. . 99
Minidisk Restrictions. . . . . 99
Timing Dependencies. . . . . .100
CPU Model-Dependent Functions. . . . . .102
Virtual Machine Characteristics. . . . .102
CMS Restrictions . . . . . .104
Miscellaneous Restrictions . . . . . .106
ABEND Dumps. . . . . .106
  Using the VMFDUMP Command. . . . . .107
  How To Print a CP Abend Dump From Tape. . . . . .108
Reading CP ABEND Dumps . . . . . .108
  Reason for the ABEND . . . . . .109
  Collect Information. . . . . .126
  Register Usage . . . . . .126
  Save Area Conventions. . . . . .127
  Virtual and Real Control Block Status.128
    VMBLOK . . . . . .128
    VCHBLOK. . . . . .131
    VCUBLOK. . . . . .131
    VDEVBLOK . . . . . .132
    RCHBLOK. . . . . .133
    RCUBLOK. . . . . .134
    RDEVBLOK . . . . . .134
  Identifying a Pageable Module. . . . .139

DEBUGGING WITH CMS . . . . . .140
CMS Debugging Commands . . . . . .140
  DEBUG. . . . . .141
  SVCTRACE . . . . . .166
DASD Dump Restore Service Program and How To Use It . . . . . .170
  Invoking DDR under CMS . . . . . .170
  Invoking DDR as a Standalone Program .170
Nucleus Load Map . . . . . .171
Load Map . . . . . .171
Reading CMS ABEND Dumps. . . . . .173
  Reason for the ABEND . . . . . .173
  Collect Information. . . . . .178
  Register Usage . . . . . .180

PART 2. CONTROL PROGRAM (CP) . . . . . .181

VM/370 . . . . . .183
Introduction to the VM/370 Control Program . . . . . .183
  Virtual Machine Time Management. . . .184
  Virtual Machine Storage Management . .184
  Virtual Machine I/O Management . . . .186
  Spooling Functions . . . . . .187
  Spool File Recovery. . . . . .188
  CP Commands. . . . . .189

PROGRAM STATES . . . . . .191

USING CPU RESOURCES. . . . . .192
Queue 1. . . . . .192
Queue 2. . . . . .192

INTERRUPTION HANDLING. . . . . .194
Program Interrupt. . . . . .194
Machine Check Interrupt. . . . . .194
SVC Interrupt. . . . . .194
External Interrupt . . . . . .195

FUNCTIONAL INFORMATION . . . . . .196
Performance Guidelines . . . . . .208
General Information. . . . . .208
Virtual Machine I/O. . . . . .209
Paging Considerations. . . . . .210
Preferred Virtual Machines . . . . . .212
The Virtual Block Multiplexer Channel Option. . . . . .218

PERFORMANCE OBSERVATION AND ANALYSIS . .219
Load Indicators. . . . . .219
The Indicate Command . . . . . .219
The Class G INDICATE Command . . . . .220
The Class E INDICATE Command . . . . .222
The MONITOR Command. . . . . .226
Implemented Classes. . . . . .230
VM Monitor Response to Unusual Tape Conditions. . . . . .232
VM Monitor Considerations. . . . . .232
VM Monitor Data Volume and Overhead. .233
Load Environments of VM/370. . . . . .234

ACCOUNTING RECORDS . . . . . .237
Accounting Record for Virtual Machine Resource Usage. . . . . .237
Accounting Records for Dedicated Devices and Temporary Disk Space. . .238
Accounting Records Created by the User. . . . . .238
Operational Notes. . . . . .239
User Accounting Options. . . . . .239

GENERATING SAVED SYSTEMS . . . . . .241
The NAMESYS Macro For Saved Systems. . .241
Using the SAVESYS Command. . . . . .243
Shared Segments. . . . . .243
Special Considerations for Shared Segments. . . . . .244
Discontiguous Saved Segments . . . . . .244
User Requirements. . . . . .245
The NAMESYS Macro for Discontiguous Saved Segments. . . . . .245
Loading and Saving Discontiguous Shared Segments . . . . . .247
How The Interface Works. . . . . .247
Shared Segment Protection. . . . . .248
Virtual Machine Operation. . . . . .249

VM/VS HANDSHAKING. . . . . .251
Closing CP Spool Files . . . . . .252
Pseudo Page Faults . . . . . .252
VS1 Nonpaging Mode . . . . . .253
Miscellaneous Enhancements . . . . . .253

TIMERS IN A VIRTUAL MACHINE. . . . . .254
Interval Timer . . . . . .254
CPU Timer. . . . . .254
TOD Clock. . . . . .255
Clock Comparator . . . . . .255
Pseudo Timer . . . . . .255
Pseudo Timer Start I/O . . . . . .256
Pseudo Timer DIAGNOSE. . . . . .256

DIAGNOSE INSTRUCTION IN A VIRTUAL MACHINE . . . . . .257
DIAGNOSE Code 0 -- Store Extended-Identification Code. . . . . .257
DIAGNOSE Code 4 -- Examine Real Storage.258
DIAGNOSE Code 8 -- Virtual Console Function. . . . . .258
DIAGNOSE Code C -- Pseudo Timer. . . . .259
DIAGNOSE Code 10 -- Release Pages. . . .259
DIAGNOSE Code 14 -- Input Spool File Manipulation. . . . . .260
Subcode X'000'. . . . . .260
Subcode X'004'. . . . . .260
Subcode X'008'. . . . . .261
Subcode X'00C'. . . . . .261
Subcode X'010'. . . . . .261
Subcode X'014'. . . . . .261
Subcode X'018'. . . . . .262
Subcode X'FFF'. . . . . .262
DIAGNOSE Code 18 -- Standard DASD I/O. .262
DIAGNOSE Code 1C -- Clear I/O Recording.263
DIAGNOSE Code 20 -- General I/O. . . . .263
DIAGNOSE Code 24 -- Device Type and Features. . . . . .264
DIAGNOSE Code 28 -- Channel Program Modification. . . . . .265
DIAGNOSE Code 2C -- Return DASD Start of LOGREC . . . . . .266
DIAGNOSE Code 30 -- Read One Page of LOGREC Data . . . . . .266
DIAGNOSE Code 34 -- Read System Dump Spool File. . . . . .266
DIAGNOSE Code 38 -- Read System Symbol Table . . . . . .267
DIAGNOSE Code 3C -- VM/370 Directory . .267
DIAGNOSE Code 4C -- Generate Accounting Cards for the Virtual User. . . . . .267
DIAGNOSE Code 50 -- Save the 3704/3705 Control Program Image . . . . . .269
DIAGNOSE Code 54 -- Control The Function of the PA2 Function Key. . . .269
DIAGNOSE Code 58 -- 3270 Virtual Console Interface . . . . . .269
DIAGNOSE Code 5C: Error Message Editing.270
DIAGNOSE CODE X'60' - DETERMINING THE VIRTUAL MACHINE STORAGE SIZE. . . . .271
DIAGNOSE CODE X'64' - Finding, Loading and Purging a Named Segment .271

CP CONVENTIONS . . . . . .274
CP Coding Conventions. . . . . .274
CP Loadlist Requirements . . . . . .276

HOW TO ADD A CONSOLE FUNCTION TO CP. . .278

PRINT BUFFERS AND FORMS CONTROL. . . . .279
Adding New Print Buffer Images . . . . .280
UCS Buffer Images. . . . . .280
USCB Buffer Images . . . . . .282
Forms Control Buffer . . . . . .285

PART 3. CONVERSATIONAL MONITOR SYSTEM (CMS) . . . . . .287

INTRODUCTION TO CMS. . . . . .289
The CMS Command Language . . . . . .289
The File System. . . . . .290
Program Development. . . . . .291

INTERRUPT HANDLING IN CMS. . . . . .293
SVC Interruptions. . . . . .293
Internal Linkage SVCs. . . . . .293
Other SVCs. . . . . .293
Input/Output Interruptions . . . . . .294
Terminal Interruptions . . . . . .295
Reader/Punch/Printer Interruptions . . .295
User Controlled Device Interruptions . .295
Program Interruptions. . . . . .295
External Interruptions . . . . . .296
Machine Check Interruptions. . . . . .296

FUNCTIONAL INFORMATION . . . . . .297
Register Usage . . . . . .297
Structure of DMSNUC. . . . . .297
USERSECT (User Area) . . . . . .298
DEVTAB (Device Table). . . . . .298
Structure of CMS Storage . . . . . .298
Free Storage Management. . . . . .300
GETMAIN Free Storage Management. . . .300
DMSFREE Free Storage Management. . . .302
Releasing Allocated Storage. . . . . .307
DMSFREE Service Routines . . . . . .308
Error Codes from DMSFRES, DMSFREE, and DMSFRET . . . . . .310
CMS Handling of PSW Keys . . . . . .310
CMS SVC Handling . . . . . .312
SVC Types and Linkage Conventions. . .312
Search Hierarchy for SVC 202 . . . . .314
User and Transient Program Areas . . .315
Called Routine Start-up Table. . . . .317
Returning to the Calling Routine . . .317
CMS Interface for Display Terminals. . .320

HOW TO ADD A COMMAND OR EXEC PROCEDURE TO CMS. . . . . .321

OS MACRO SIMULATION UNDER CMS. . . . . .322
OS Data Management Simulation. . . . . .322
Handling Files that Reside on CMS Disks . . . . . .322
Handling Files that Reside on OS or DOS Disks . . . . . .323
Simulation Notes . . . . . .325
Access Method Support. . . . . .329
Reading OS Data Sets and DOS Files Using OS Macros . . . . . .331

DOS/VS SUPPORT UNDER CMS . . . . . .335
Hardware Devices Supported . . . . . .335
CMS Support of DOS/VS Functions. . . . .336
Logical Unit Assignment. . . . . .338
DOS/VS Supervisor and I/O Macros Supported by CMS/DOS. . . . . .340
Supervisor Macros. . . . . .340
Sequential Access Method -- Declarative Macros. . . . . .345
Sequential Access Method -- Imperative Macros . . . . . .355
DOS/VS Transient Routines. . . . . .355
EXCP Support in CMS/DOS. . . . . .356
DOS/VS Supervisor Control Blocks Simulated by CMS/DOS. . . . . .357
User Considerations and Responsibilities. . . . . .357
DOS/VS System Generation and Updating Considerations. . . . . .357
VM/370 Directory Entries . . . . . .358
CMS/DOS Storage Requirements . . . . . .358
When the DOS/VS System Must Be Online. .359
Performance. . . . . .360
Execution Considerations and Restrictions. . . . . .360

CMS SUPPORT FOR OS AND DOS VSAM FUNCTIONS . . . . . .361
Hardware devices Supported . . . . . .361
DOS/VS Supervisor Macros and Logical Transients Support for VSAM . . . . . .362
Storage Requirements . . . . . .363
Data Set Compatibility Considerations. .363
ISAM Interface Program (IIP) . . . . . .363

SAVING THE CMS SYSTEM. . . . . .364
The CMSSEG Discontiguous Saved Segment .364
CMSSEG Usage Options . . . . . .364
Saved System Restrictions for CMS. . . .365

CMS BATCH FACILITY . . . . . .366
Resetting Batch Facility System Limits .366
Writing Routines To Handle Special Installation Input. . . . . .366
BATEXIT1: Processing User-Specified Control Language. . . . . .367
BATEXIT2: Processing the Batch Facility /JOB Control Card. . . . . .367
EXEC Procedures for the Batch Facility Virtual Machine . . . . . .367
Data Security under the Batch Facility .368
Improved IPL Performance Using a Saved System. . . . . .368

AUXILIARY DIRECTORIES. . . . . .369
How To Add an Auxiliary Directory. . . .369
Generation of the Auxiliary Directory.369
Initializing the Auxiliary Directory .369
Establishing the Proper Linkage. . . .370
An Example of Creating an Auxiliary Directory . . . . . .371

ASSEMBLER VIRTUAL STORAGE REQUIREMENTS .373
Overlay Structures . . . . . .373
Prestructured Overlay. . . . . .373
Dynamic Load Overlay . . . . . .375

PART 4. IBM 3704 AND 3705 COMMUNICATIONS CONTROLLERS. . . . . .377

INTRODUCTION TO THE IBM 3704 and 3705 COMMUNICATIONS CONTROLLERS. . . . . .379
VM/370 Support of the 3704 and 3705. . .379
Emulation Program (EP) with VM/370 . .380
Network Control Program (NCP) with VM/370. . . . . .380
Partitioned Emulation Program (PEP) with VM/370 . . . . . .381
Generating a VM/370 System that Supports the 3704 and 3705. . . . . .381

LOADING THE 3704/3705 CONTROL PROGRAM. .382
Save the 3704/3705 Control Program Image on Disk . . . . . .382
The SAVENCP Command. . . . . .382
Execution of the SAVENCP Program . . .383
Load the 3704/3705 Control Program . . .384
The NETWORK LOAD Command Line. . . . .384

Execution of the NETWORK LOAD Command.384
Considerations for Loading 3704/3705 Control Programs. . . . . .385
Special Considerations for Loading the EP 3704/3705 Control Program. . .385
Special Considerations for Loading the NCP and PEP 3704/3705 Control Programs. . . . . .386
Logging on through the 3704/3705 . . . .386
Special Sign-on Procedures for 3704/3705 Lines in NCP Mode and with the MTA Feature . . . . . .387
Logging on After an NCP Control Program Has Abnormally Terminated . .388
Applying PTFs to the 3704/3705 Load Library . . . . . .388
The ZAP Service Program. . . . . .388
ZAP Input Control Records. . . . . .390
Special Considerations for Using the ZAP Service Program . . . . . .395

TESTING THE 3704/3705 CONTROL PROGRAM. .396
NETWORK Command Usage for Remote 3270. .396
Resource Identification for Remote 3270s . . . . . .396
NETWORK. . . . . .397
How to Use the NETWORK Command . . . .397
NCPDUMP Service Program and How to Use It. . . . . .406
Using the NCPDUMP Command. . . . . .406

PART 5. REMOTE SPOOLING COMMUNICATIONS SUBSYSTEM (RSCS). . . . . .409

INTRODUCTION TO RSCS . . . . . .411
Locations And Links. . . . . .411
Remote Stations. . . . . .411
VM/370 Spool System Interface. . . . . .412
RSCS Command Language. . . . . .412

STRUCTURE OF RSCS VIRTUAL STORAGE. . . .414
RSCS Supervisor. . . . . .415
Supervisor Queue Extension . . . . . .415
Free Storage . . . . . .415
System Control Task. . . . . .416
Free Storage and Line Drivers. . . . . .416
Line Allocation Task . . . . . .416
Spool File Access Task . . . . . .416

FUNCTIONAL INFORMATION . . . . . .417
Virtual Storage Management . . . . . .417
Page Allocation. . . . . .417
Queue Element Management . . . . . .417
File Management. . . . . .418
Tag Slot Queues. . . . . .418
Spool File Access. . . . . .418
Task-to-Task Communication . . . . . .419
RSCS Command Processing. . . . . .419
RSCS Message Handling. . . . . .420
Interruption Handling. . . . . .420
External Interruptions . . . . . .420
SVC Interruptions. . . . . .420
I/O Interruptions. . . . . .421

LOGGING I/O ACTIVITY . . . . . .422
The SML Log Record . . . . . .422
The NPT Log Record . . . . . .423

APPENDIX A: SYSTEM/370 INFORMATION . . .425
Control Registers. . . . . .425

APPENDIX B: MULTI-LEAVING. . . . . .429
MULTI-LEAVING in VM/370. . . . . .429
MULTI-LEAVING Philosophy . . . . . .429
MULTI-LEAVING Control Specification. . .431
Record Control Byte (RCB). . . . . .432
Sub-Record Control Byte (SRCB) . . . .433
String Control Byte (SCB). . . . . .435
Block Control Byte (BCB) . . . . . .435
Function Control Sequence (FCS). . . .436

APPENDIX C: VM MONITOR TAPE FORMAT AND CONTENT . . . . . .437
Header Record. . . . . .437
Data Records . . . . . .438

INDEX. . . . . .445

## Figures


Figure 1. ABEND Messages.....14
Figure 2. VM/370 Problem Types.....18
Figure 3. Does a Problem Exist?.....23
Figure 4. Debug Procedures for Waits and Loops.....24
Figure 5. Debug Procedures for Unexpected Results and an ABEND.....25
Figure 6. Summary of VM/370 Debugging Tools.....39
Figure 7. Comparison of CP and CMS Facilities for Debugging.....44
Figure 8. Annotated Sample of Output from the TYPE and PRINT Functions of the DDR Program.....94
Figure 9. CP Trace Table Entries.....98
Figure 10. CP ABEND Codes.....111
Figure 11. CP Control Block Relationships.....129
Figure 12. CP Device Classes, Types, Models, and Features.....137
Figure 13. Summary of SVC Trace Output Lines.....169
Figure 14. Sample CMS Load Map.....172
Figure 15. CMS Control Blocks.....174
Figure 16. CMS ABEND Codes.....175
Figure 17. CP Initialization.....197
Figure 18. Real I/O Control Blocks.....198
Figure 19. Virtual I/O Control Blocks..199
Figure 20. SVC Interrupt Handling.....200
Figure 21. External Interrupt Handling.201
Figure 22. Program Interrupt Handling..202
Figure 23. Paging.....203
Figure 24. Virtual Spooling.....204
Figure 25. Real Spooling.....205
Figure 26. Virtual Tracing.....206
Figure 27. Virtual-to-Real Address Translation.....207
Figure 28. Storage in a Virtual=Real Machine.....215
Figure 29. Formats of Pseudo Timer Information.....256
Figure 30. Addressable Storage Before and After a LOADSYS Function.....272
Figure 31. UCSB Associative Field Chart.....283
Figure 32. CMS File System.....292
Figure 33. Devices Supported by a CMS Virtual Machine.....299
Figure 34. CMS Storage Map.....301
Figure 35. CMS Command (and Request) Processing.....316
Figure 36. PSW Fields When Called Routine Starts.....317
Figure 37. Register Contents When Called Routine Starts.....317
Figure 38. Simulated OS Supervisor Calls.....324
Figure 39. Summary of Changes to CMS Commands to Support CMS/DOS.....337
Figure 40. Physical IOCS Macros Supported by CMS/DOS.....341
Figure 41. DOS/VS Macros Supported Under CMS.....342
Figure 42. CMS/DOS Support of DTFCD Macro.....346
Figure 43. CMS/DOS Support of DTFCN Macro.....347
Figure 44. CMS/DOS Support of DTFDI Macro.....348
Figure 45. CMS/DOS Support of DTFMT Macro.....350
Figure 46. CMS/DOS Support of DTFPR Macro.....352
Figure 47. CMS/DOS Support of DTFSD Macro.....353
Figure 48. DOS/VSAM Macros Supported by CMS.....362
Figure 49. An Overlay Structure.....374
Figure 50. RSCS Command Summary.....413
Figure 51. RSCS Storage Allocation.....414
Figure 52. Control Register Allocation.425
Figure 53. Control Register Assignments.....426
Figure 54. The Extended Control PSW (Program Status Word).....427
Figure 55. A Typical MULTI-LEAVING Transmission Block.....430

# Part 1. Debugging with VM/370

This debugging section contains the following information:

Introductory Information

- How to start debugging
- How to use VM/370 facilities to debug ABENDs, unexpected results, loops, and waits
- Summary of VM/370 debugging tools
- Comparison of CP and CMS debugging tools

Control Program Information

- Debugging CP on a virtual machine
- Commands useful in debugging
- DASD Dump Restore program
- Internal trace table
- Restrictions
- ABEND dumps
- Reading CP ABEND dumps
- Control block summary

Conversational Monitor System Information

- Debugging commands
- DASD Dump Restore Program
- Nucleus load map
- Reading CMS ABEND dumps
- Control block summary

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

1. The Control Program (CP), which controls the resources of the real machine.

2. The virtual machine operating system running under the control of CP, such as CMS, RSCS, OS, or DOS.

3. The problem program, which executes under the control of a virtual machine operating system.

Once the area causing the problem is identified, the appropriate person should take all available information and determine the cause of the problem. Most likely, system support personnel handle all problems with CP, CMS, and RSCS; information that is helpful in debugging CP and CMS is contained in this publication. The application programmer handles all problem program errors; techniques for application program debugging are found in the VM/370: CMS User's Guide. The IBM Field Engineering Program System Representative or the installation system programmer may also reference the publication VM/370: Interactive Problem Control System (IPCS) User's Guide for information on how to use IPCS for debugging.

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 that other publication. System support personnel most likely handle 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 the VM/370: Planning and System Generation Guide for detailed information on applying PTFs.

## HOW TO START DEBUGGING

Before you can correct any problem, you must recognize that one exists. Next, you must identify the problem, collect information and determine the cause so that the problem can be fixed. When running VM/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:

1. Loop
2. Wait state
3. ABEND (Abnormal End)
4. Incorrect results

The most obvious indication of a problem is the abnormal termination of a program. Whenever a program abnormally terminates, a message is issued. Figure 1 lists the possible ABEND messages and identifies the type of ABEND for these messages.

| Message | Type of ABEND |
|---|---|
| (Alarm rings)<br>DMKDMP908I SYSTEM FAILURE CODE xxxxxx | CP ABEND, system dumps to disk. Restart is automatic. |
| Optional Messages: | |
| DMKDMP905W SYSTEM DUMP FAILURE; PROGRAM CHECK<br>DMKDMP906W SYSTEM FAILURE; MACHINE CHECK, RUN SEREP<br>DMKDMP907W SYSTEM DUMP FAILURE; FATAL I/O ERROR | If the dump program encounters a program check, machine check, or fatal I/O error, a message is issued indicating the error. CP enters the wait state with code 003 in the PSW. |
| DMKCKP900W SYSTEM RECOVERY FAILURE; PROGRAM CHECK<br>DMKCKP901W SYSTEM RECOVERY FAILURE; MACHINE CHECK, RUN SEREP<br>DMKCKP902W SYSTEM RECOVERY FAILURE; FATAL I/O ERROR - NUCL CYL - WARM CYL<br>DMKCKP922W SYSTEM RECOVERY FAILURE; INVALID SPOOLING DATA<br>DMKCKP910W SYSTEM RECOVERY FAILURE; INVALID WARM START CYLINDER<br>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 007 in the PSW. |
| DMKCKS903W SYSTEM RECOVERY FAILURE; VOLID xxxxxx ALLOCATION ERROR CYLINDER xxx<br>DMKCKS912W SYSTEM RECOVERY FAILURE; VOLID xxxxxx NOT MOUNTED<br>DMKCKS915E PERMANENT I/O ERROR ON CHECKPOINT CYLINDER<br>DMKCKS916E ERROR ALLOCATING SPOOL FILE BUFFERS<br>DMKCKS917E CHECKPOINT CYLINDER INVALID; CLEAR STORAGE AND COLD START | If the checkpoint start program encounters a severe error, a message is issued indicating the error and CP enters the wait state with code 00E in the PSW. |

Figure 1. ABEND Messages (Part 1 of 3)

| Message | Type of ABEND |
|---|---|
| DMKWRM921W SYSTEM RECOVERY FAILURE; UNRECOVERABLE I/O ERROR<br>DMKWRM903W SYSTEM RECOVERY FAILURE; VOLID xxxxxx ALLOCATION ERROR CYLINDER xxx<br>DMKWRM904W SYSTEM RECOVERY FAILURE; INVALID WARM START DATA<br>DMKWRM912W SYSTEM RECOVERY FAILURE; VOLID xxxxxx NOT MOUNTED<br>DMKWRM920W NO WARM START DATA; CKPT START FOR RETRY | If the warm start program encounters a severe error, a message is issued indicating the error and CP enters the wait state with code 009 in the PSW. |
| DMKDMP908I SYSTEM FAILURE, CODE xxxxxx<br>DMKCKP960I SYSTEM WARM START DATA SAVED<br>DMKCKP961W SYSTEM SHUTDOWN COMPLETE | CP ABEND, system dumps to tape or printer. The system stops; the operator must IPL the system to start again. |
| Optional Messages | |
| DMKDMP905W SYSTEM DUMP FAILURE; PROGRAM CHECK<br>DMKDMP906W SYSTEM DUMP FAILURE; MACHINE CHECK, RUN SEREP<br>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 003 in the PSW.<br><br>If the dump cannot find a defined dump device and if no printer is defined for the dump, CP enters a disabled wait state with code 004 in the PSW. |
| | CP termination with automatic restart. |
| DMKMCH610I MACHINE CHECK SUPERVISOR DAMAGE | The machine check handler encountered a nonrecoverable error with the VM/370 control program. |
| DMKMCH611I MACHINE CHECK SYSTEM INTEGRITY LOST | The machine check handler encountered an error that cannot be diagnosed; system integrity, at this point, is not reliable. |

Figure 1. ABEND Messages (Part 2 of 3)

| Message | Type of ABEND |
|---|---|
| | CP termination without automatic restart. |
| 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 code 002 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, system will accept commands from the terminal. Enter the DEBUG command and then the DUMP subcommand to have CMS dump storage on the printer. |
| Others<br>Refer to OS and DOS 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 1. ABEND Messages (Part 3 of 3)

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

Unproductive processing time is another symptom of a problem. This problem is not as easily recognized, especially in a time sharing environment.

## IDENTIFYING THE PROBLEM

Two types of problems are easily identified: abnormal termination is indicated by an error message, and unexpected results become apparent once the output is examined. The looping and wait state conditions are not as easily identified.

When using VM/370, you are normally sitting at a terminal and do not have the lights of the CPU control panel to help you. You may have a looping condition if your program takes longer to execute than you anticipated. Also, check your output. If the number of output records or print lines is greater than expected, the output may really be the same information repeated many times. Repetitive output usually indicates a program loop.

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

The wait state is also difficult to recognize when at the terminal. Again, the console lights are unavailable. If your program is taking longer than expected to execute, the virtual machine may be in a wait state. Display the current PSW on the terminal. Periodically, issue the CP command

```
QUERY TIME
```

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

Figure 2 helps you to identify problem types and the areas where they may occur.

| Problem Type | Where ABEND Occurs | Distinguishing Characteristics |
|---|---|---|
| ABEND | CP ABEND | The alarm rings and the message<br>DMKDMP908I SYSTEM FAILURE, CODE xxxxxx<br>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:<br>DMKDMP905W SYSTEM DUMP FAILURE; PROGRAM CHECK<br>DMKDMP906W SYSTEM DUMP FAILURE; MACHINE CHECK, RUN SEREP<br>DMKDMP907W SYSTEM DUMP FAILURE; FATAL I/O ERROR<br>DMKCKP900W SYSTEM RECOVERY FAILURE; PROGRAM CHECK<br>DMKCKP901W SYSTEM RECOVERY FAILURE; MACHINE CHECK, RUN SEREP<br>DMKCKP902W SYSTEM RECOVERY FAILURE; FATAL I/O ERROR<br>DMKCKP922W SYSTEM RECOVERY FAILURE; INVALID SPOOLING DATA<br>DMKCKP910W SYSTEM RECOVERY FAILURE; INVALID WARM START CYLINDER<br>DMKCKP911W SYSTEM RECOVERY FAILURE; WARM START AREA FULL<br>DMKCKS903W SYSTEM RECOVERY FAILURE; VOLID xxxxxx ALLOCATION ERROR CYLINDER xxx n<br>DMKCKS912W SYSTEM RECOVERY FAILURE; VOLID xxxxxx NOT MOUNTED<br>DMKCKS915E PERMANENT I/O ERROR ON CHECKPOINT CYLINDER<br>DMKCKS917E CHECKPOINT CYLINDER INVALID; CLEAR STORAGE AND COLD START<br>DMKWRM921W SYSTEM RECOVERY FAILURE; UNRECOVERABLE I/O ERROR<br>DMKWRM903W SYSTEM RECOVERY FAILURE; VOLID xxxxxx ALLOCATION ERROR CYLINDER xxx<br>DMKWRM904W SYSTEM RECOVERY FAILURE; INVALID WARM START DATA<br>DMKWRM912W SYSTEM RECOVERY FAILURE; VOLID xxxxxx NOT MOUNTED |
| | CP ABEND | The following messages appear on the CPU console:<br>DMKDMP908I SYSTEM FAILURE, CODE xxxxxx<br>DMKDMP960I SYSTEM WARM START DATA SAVED<br>DMKDMP961W SYSTEM SHUTDOWN COMPLETE<br>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 programs, CP enters the wait state after issuing one or more of the following messages:<br>DMKDMP905W SYSTEM DUMP FAILURE; PROGRAM CHECK<br>DMKDMP906W SYSTEM DUMP FAILURE; MACHINE CHECK, RUN SEREP<br>DMKDMP907W SYSTEM DUMP FAILURE; FATAL I/O ERROR |

Figure 2. VM/370 Problem Types (Part 1 of 5)

| Problem Type | Where ABEND Occurs | Distinguishing Characteristics |
|---|---|---|
| ABEND (cont.) | CP ABEND (cont.) | DMKCKP900W SYSTEM RECOVERY FAILURE; PROGRAM CHECK<br>DMKCKP901W SYSTEM RECOVERY FAILURE; PROGRAM CHECK\| RUN SEREP<br>DMKCKP902W SYSTEM RECOVERY FAILURE; FATAL I/O ERROR<br>DMKCKP910W SYSTEM RECOVERY FAILURE; INVALID WARM START CYLINDER<br>DMKCKP911W SYSTEM RECOVERY FAILURE; WARM START AREA FULL |
| | CP termination with automatic start | An unrecoverable machine check error has occurred. One of the following messages:<br>DMKMCH610I MACHINE CHECK SUPERVISOR DAMAGE<br>DMKMCH611I MACHINE CHECK INTEGRITY LOST<br>appears on the CPU console. The system is automatically restarted. |
| | CP termination without automatic restart | An unrecoverable channel check error has occurred. The message:<br>DMKCCH603W CHANNEL ERROR, RUN SEREP, RESTART SYSTEM<br>appears on the CPU console, and CP enters wait state. |
| | Virtual Machine ABEND (CMS) | The CMS message<br>DMSABM148T SYSTEM ABEND xxx CALLED FROM xxxxxx<br>appears on the terminal. The system stops and waits for a command to be entered on the terminal. In order 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.<br>VM/370 may terminate or reset a virtual machine if a nonrecoverable channel check or machine check occurs in that virtual machine. One of the following messages:<br>DMKMCH616I MACHINE CHECK; USER userid TERMINATED<br>DMKCCH604I CHANNEL ERROR; DEV xxx; USER userid; MACHINE RESET<br>to the system operator at the CPU console. Also, the virtual user is notified that his machine was terminated or reset by one of the following messages:<br>DMKMCH619I MACHINE CHECK; OPERATOR TERMINATED<br>DMKCCH606I CHANNEL ERROR; OPERATOR TERMINATED |

Figure 2. VM/370 Problem Types (Part 2 of 5)

| Problem Type | Where ABEND Occurs | Distinguishing Characteristics |
|---|---|---|
| Unexpected Results | CP | If an operating system, other than CMS, executes properly on a real machine, but not properly with CP, a problem exists. Inaccurate data on disk or system files (such as spool files) is an error. |
| | Virtual Machine | 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. |
| Wait | Disabled CP wait | The CPU wait light is on. Also, pressing the REQUEST key on the operator's console, or the equivalent action, leaves the REQUEST PENDING light on. If the message<br>DMKMCH612W MACHINE CHECK TIMING FACILITIES DAMAGE, RUN SEREP<br>appears on the CPU console, a machine check (probable hardware error) caused the CP disabled wait state. If the message<br>DMKCCH603W CHANNEL ERROR, RUN SEREP, RESTART SYSTEM<br>appears on the CPU console, a channel check (probable hardware error) caused the CP disabled wait state. If the message<br>DMKCPI955W INSUFFICIENT STORAGE FOR VM/370<br>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<br>DMKPAG415E CONTINUOUS PAGING ERRORS FROM DASD xxx<br>appears on the CPU console, the control program (CP) has entered a disabled wait state with code 00F in the PSW. Consecutive hardware errors are occurring on one or more VM/370 paging devices.<br>If the system is being controlled at an alternate console, messages DMKCKP910I, DMKCKP911W, and DMKCKP960I are not generated before the system goes into a wait state. |
| | Enabled CP wait | The CPU console light is on, but the system accepts interrupts from I/O devices. |
| | 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 messages:<br>DMKDSP450W CP ENTERED; DISABLED WAIT PSW<br>DMKDSP451W CP ENTERED; INVALID PSW |

Figure 2. VM/370 Problem Types (Part 3 of 5)

| Problem Type | Where ABEND Occurs | Distinguishing Characteristics |
|---|---|---|
| Wait (cont.) | Disabled virtual machine wait (cont.) | DMKDSP452W CP ENTERED; EXTERNAL INTERRUPT LOOP<br>DMKDSP453W CP ENTERED; PROGRAM INTERRUPT LOOP |
| | Enabled virtual machine wait | A PSW enabled for I/O interrupts is loaded. Nothing happens if an I/O device fails to issue an I/O interrupt. If a program is taking longer to execute than expected, periodically issue the CP command, QUERY TIME. If the processing time remains unchanged, there is probably a virtual machine enabled wait.<br>CMS types a blip character for every 2 seconds of elapsed processing time. If the program does not end and blip characters stop typing, an enabled wait state probably exists. |
| | Disabled RSCS wait | The RSCS operator is notified of the wait state by CP issuing the message<br><br>DMKDSP450W CP ENTERED; DISABLED WAIT PSW<br><br>If, in addition, the message<br><br>DMTINI402T IPL DEVICE READ I/O ERROR<br><br>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 011 in the PSW.<br><br>If a program check occurs before the program check handler is activated, RSCS enters a disabled wait state with a code of 007 in the PSW.<br><br>If a program check occurs after the program check handler is activated, RSCS enters a disabled wait state with a code of 001 in the PSW. One of the following messages may also appear on the RSCS console:<br><br>DMTREX090T PROGRAM CHECK IN SUPERVISOR — RSCS SHUTDOWN<br>DMTREX091T INITIALIZATION FAILURE — RSCS SHUTDOWN |
| | Enabled RSCS wait | RSCS has no task ready for execution. A PSW, enabled for external and I/O interrupts, is loaded with a wait code of all zeroes. |

Figure 2. VM/370 Problem Types (Part 4 of 5)

| Problem Type | Where ABEND Occurs | Distinguishing Characteristics |
|---|---|---|
| Loop | CP disabled loop | The CPU console wait light is off. The problem state bit of the real PSW is off. No I/O interrupts are accepted. |
| | CP enabled loop | There is no such condition. |
| | Virtual machine disabled loop | The program is taking longer to execute than anticipated. Signalling attention from the terminal does not cause an interrupt in the virtual machine. The virtual machine operator cannot communicate with the virtual machine's operating system by signalling attention. |
| | Virtual machine enabled loop | Excessive processing time is often an indication of a loop. Use the CP QUERY TIME command to check the elapsed processing time. In CMS, the continued typing of the blip characters indicates that processing time is elapsing. If time has elapsed, periodically display the virtual PSW and check the instruction address. If the same instruction, or series of instructions, continues to appear in the PSW, a loop probably exists. |

Figure 2. VM/370 Problem Types (Part 5 of 5)

## ANALYZING THE PROBLEM

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

The section, "How To Use VM/370 Facilities To Debug," describes procedures to follow in determining the cause of various problems that can occur in the Control Program or in the virtual machine. See the VM/370: CMS User's Guide for information on using VM/370 facilities to debug a problem program.

If it becomes necessary to apply a Program Temporary Fix (PTF) to a VM/370 component, refer to the VM/370: Planning and System Generation Guide for detailed information on applying PTFs. Figure 3, Figure 4, and Figure 5 summarize the debugging process from identifying the problem to finding the cause.

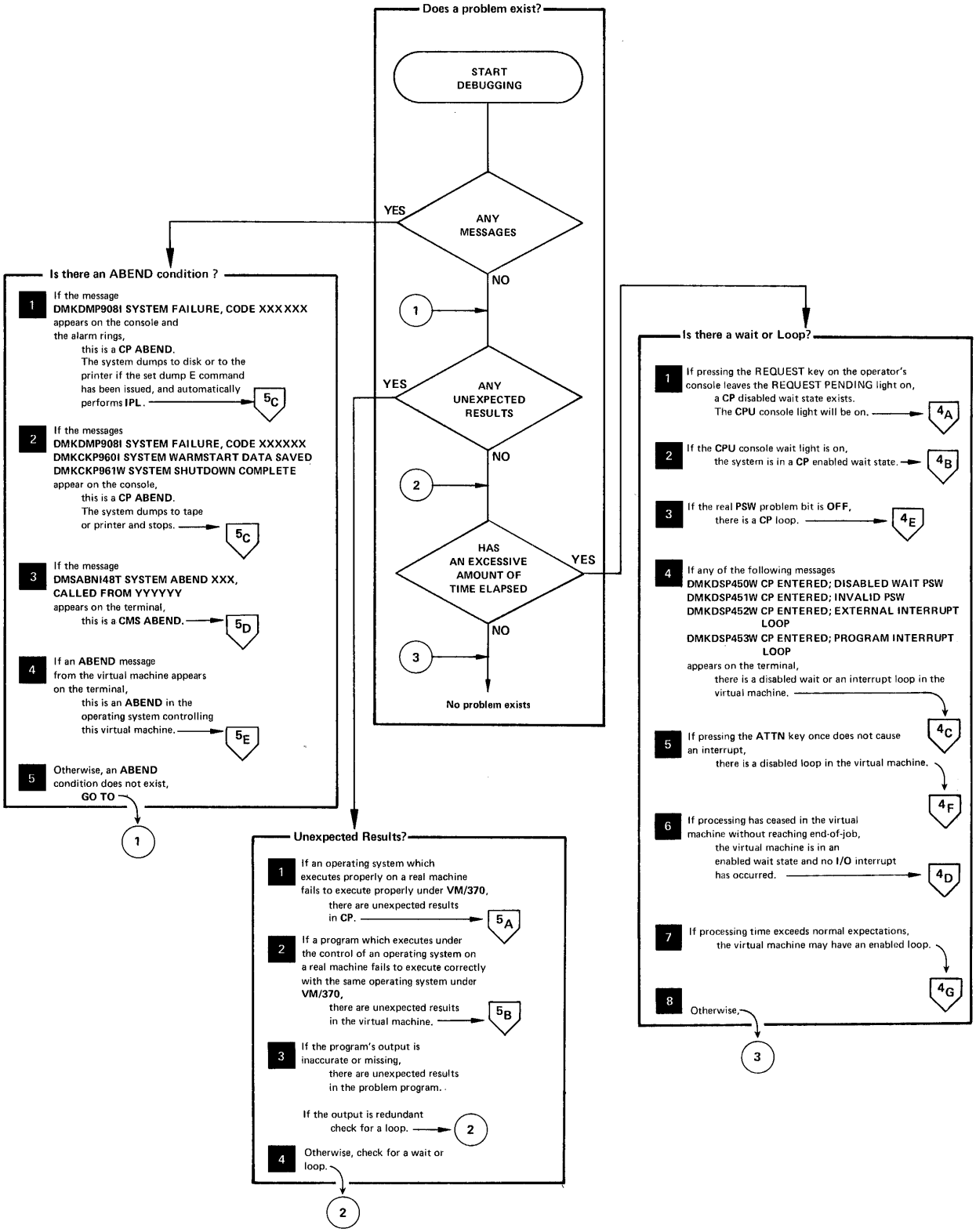



**Figure 3. Does a Problem Exist?**

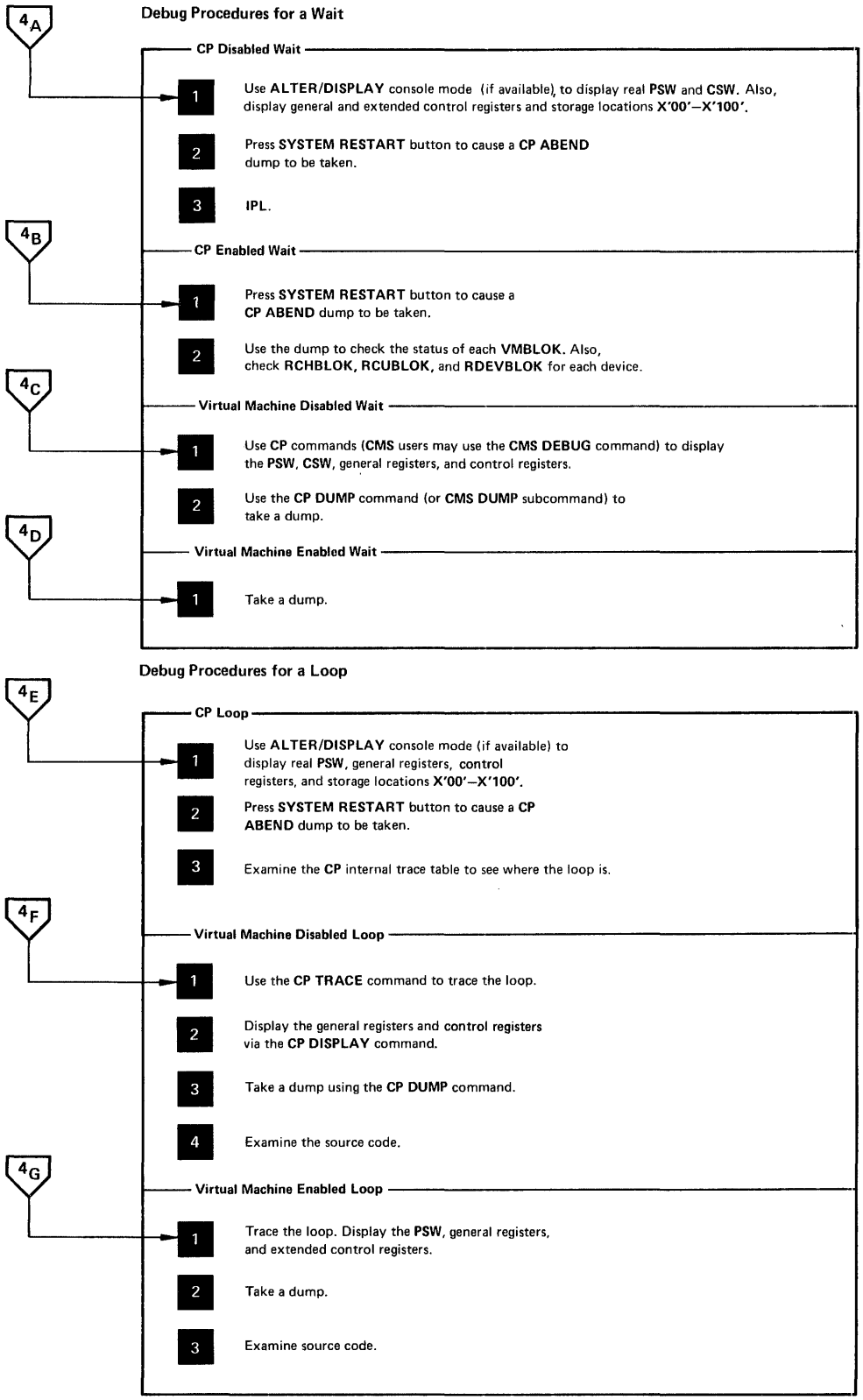
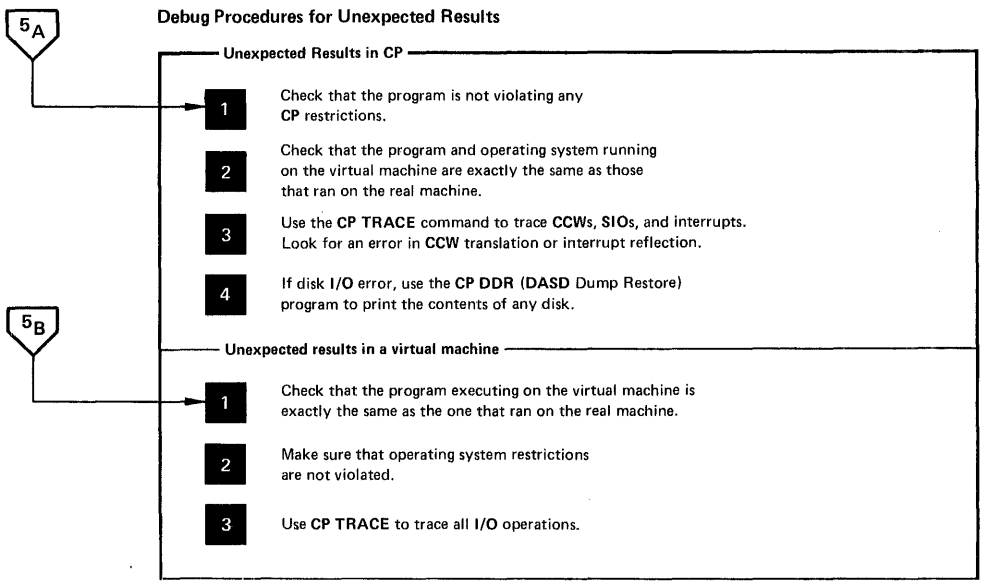



Figure 4. Debug Procedures for Waits and Loops



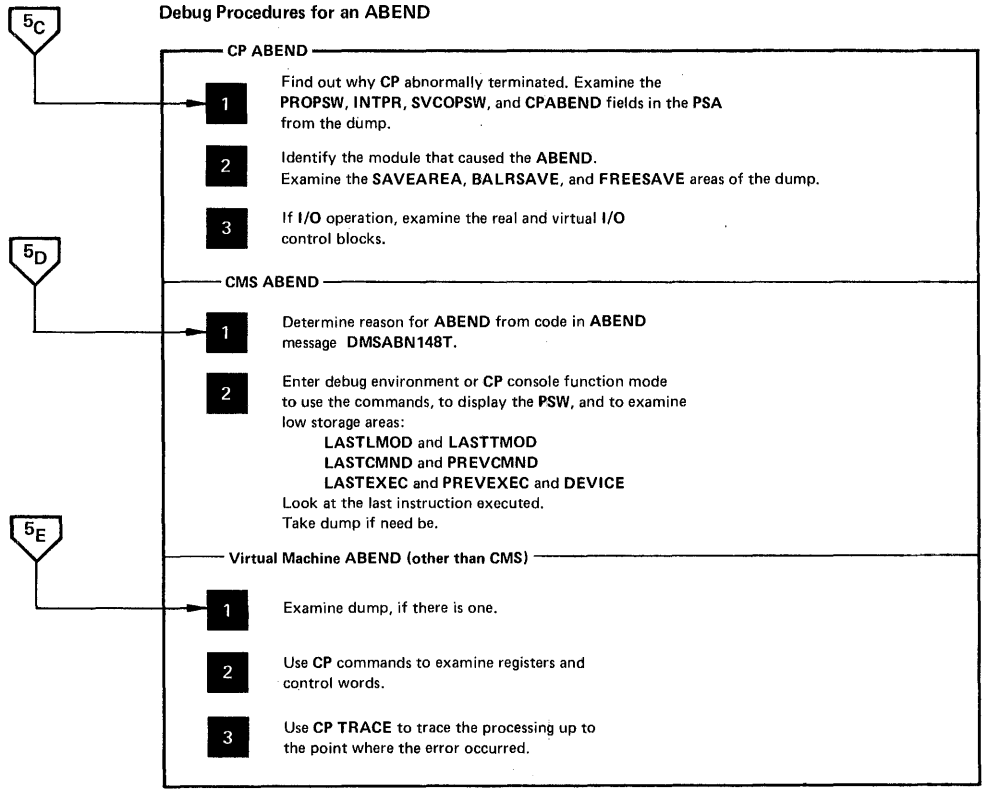




Figure 5. Debug Procedures for Unexpected Results and an ABEND

## HOW TO USE VM/370 FACILITIES TO DEBUG

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

The following sections describe specific debugging procedures for the various error conditions. The procedures will tell you what to do and what debug tool to use. For example, the procedure may say dump storage using the CP DUMP command. The procedure will not tell you how to use the debug tool. Refer to the "CP Commands to Debug the Virtual Machine" and "CMS Debugging Commands" sections for a detailed description of each debug tool, including how to invoke it.

### ABEND

When a system does not know how to continue, it abnormally terminates.

### CP ABEND

When the VM/370 Control Program abnormally terminates, a dump is taken. This dump can be directed to tape or printer, or dynamically allocated to a direct access storage device. The output device for a CP ABEND dump is specified by the CP SET command. See the "ABEND Dumps" section for a description of the SET and VMFDUMP commands.

Use the dump to find what caused the Control Program to terminate. First, find why the system abnormally terminated and then see how the condition can be corrected. See the "Reading CP ABEND Dumps" discussion for detailed information on reading a CP ABEND dump.

REASON FOR THE ABEND: CP will terminate and take an abnormal termination dump under three conditions:

1. Program Check in CP

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

2. Module Issuing an SVC 0

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

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

3. Operator Pressing SYSTEM RESTART Button on CPU Console

   Examine the old PSW at location X'08' to find the location of the instruction that was executing when the operator pressed SYSTEM

RESTART. The operator presses SYSTEM RESTART when CP is in a disabled wait state or loop.

EXAMINE LOW STORAGE AREAS: The information in low storage tells you the status of the system at the time CP terminated. Status information is stored in the Prefix Storage Area (PSA). You should be able to tell the module that was executing by looking at the PSA. Refer to the appropriate save area (SAVEAREA, BALRSAVE, or FREESAVE) to see how that module started to execute. The Prefix Storage Area is described in the VM/370: Data Areas and Control Blocks Logic publication.

Examine the real and virtual control blocks to find the status of I/O operations. Figure 11 shows the relationship of CP Control Blocks.

Examine the CP internal trace table. This table can be extremely helpful in determining the events that preceded the ABEND. The "CP Internal Trace Table" description tells you how to use the trace table.

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

If the program check old PSW (PROPSW) or the SVC old PSW (SVCOPSW) points to an address beyond the end of the resident nucleus, the module that caused the ABEND is a pageable module. Refer to "Reading CP ABEND Dumps" to find out how to identify that pageable module. Use the CP load map that was created when the VM/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 interrupt code is invalid. These severe machine checks cause the control program 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 the control program, the message

```
DMKMCH610I MACHINE CHECK SUPERVISOR DAMAGE
```

appears on the CPU console. The control program 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, the control program is terminated and automatically restarted.

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

## CMS ABEND

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 will probably want to use its debug facilities to examine status. You may be able to determine the cause of the ABEND without taking a dump.

The debug program is located in the resident nucleus of CMS and has its own save and work areas. Because the debug program itself does not alter the 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 in debugging knowing that you cannot inadvertently overlay storage because the CP and CMS storage areas are completely separate.

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

1. Program Exception

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

   ```
   DMSITP141T xxxxxxxx EXCEPTION OCCURRED AT xxxxxx IN ROUTINE
              xxxxxxxx
   ```

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

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

2. ABEND Macro

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

3. Halt Execution (HX)

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

4. System ABEND

   A CMS system routine can abnormally terminate by issuing the DMSABN macro. The first three hexadecimal digits of the system ABEND code type 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 DMSABN149T system termination message.

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

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

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

WHAT TO DO WHEN CMS ABNORMALLY TERMINATES: After an ABEND, two courses of action are available in CMS. In addition, by signalling attention, you can enter the CP command mode and use CP's debugging facilities.

Two courses of action available in CMS are:

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

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

The ABEND recovery function performs the following:

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 master file directory.

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

4. All link blocks allocated by DMSSLN are freed.

5. All FCB pointers are set to zero.

6. All user storage is released.

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

8. 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.

A DEBUGGING PROCEDURE: When a CMS ABEND occurs, you will 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" 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" for a detailed description of how to use the CP commands. Also refer to Figure 7 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 trying to determine the module that caused the ABEND. The CMS storage-resident nucleus routines reside in fixed storage locations.

   Also check the interruption code in the PSW.

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

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

| Field | Contents |
|---|---|
| LASTCMND | Contains the name of the last command issued. |
| PREVCMND | Contains the name of the next-to-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 the listing.

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

## Virtual Machine ABEND (Other than CMS)

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

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

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 can be 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:

DMKMCH616I MACHINE CHECK; USER userid TERMINATED

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

appears on the CPU console.

## UNEXPECTED RESULTS

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

### Unexpected Results in CP

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

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

Next, be sure 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
- The same copy of the operating system (and program)
- The same libraries

If the problem still is not found, look for an I/O problem. Try to reproduce the problem, this time tracing all CCWs, SIOs, and interrupts 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 inadvertently violated, or that an error occurred in the Control Program.

### Unexpected Results in a Virtual Machine

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

If unexpected results occur (such as TEXT records interspersed in printed output), you may wish to examine the contents of the system or user disk files. Non-CMS users may execute any of the utilities included in the operating system they are using to examine and rearrange

files. Refer to the utilities publication for the operating system running in the virtual machine for information on how to use the utilities.

CMS users should use the DASD Dump Restore (DDR) service program to print or move the data stored on direct access devices. The 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:

1. DUMP -- dumps part, or all of the data from a DASD device to magnetic tape.

2. 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.

3. 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. In order to copy one tape to another, the original tape must have been created by the DDR DUMP function.

4. PRINT -- selectively prints the hexadecimal and EBCDIC representation of DASD and tape records on the virtual printer.

5. TYPE -- selectively displays the hexadecimal and EBCDIC representation of DASD and tape records on the terminal.

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

## LOOP

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

### CP Disabled Loop

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

1. Use the alter/display console mode to display the real PSW, general registers, control registers and storage locations X'00' - X'100'.

2. Press the SYSTEM RESTART button to cause an ABEND dump to be taken.

3. Save the information collected for the system programmer or system support personnel.

After the CPU operator has collected the information, the system programmer or system support personnel examine 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 in a virtual machine exists, the virtual machine operator cannot communicate with the virtual machine's operating system. That means that signalling attention does not cause an interrupt.

Enter the CP console function mode.

1. Use the CP TRACE command to trace the entire loop. Display general and extended control registers via the CP DISPLAY command.

2. Take a dump via the CP DUMP command.

3. Examine the source code.

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

## Virtual Machine Enabled Loop

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

1. Use the CP TRACE 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 manner of loop entry is still undetermined, assume that a wild branch has occurred and begin a search for its origin.

## WAIT

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

### CP Disabled Wait

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

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

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

```
DMKMCH612W MACHINE CHECK TIMING FACILITIES DAMAGE; RUN SEREP

DMKCCH603W CHANNEL ERROR, RUN SEREP, RESTART SYSTEM
```

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:

1. The generated system is larger than the real machine size OR

2. A hardware malfunction occurs which reduced 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 followed by the CPU operator to record the needed information.

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

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

3. IPL the system.

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

## CP Enabled Wait

If you determine that CP is in an enabled wait state, but that no I/O interrupts are occurring, there may be an error in 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" 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.

- PSW
- CSW
- 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 7 compares the two.

## Virtual Machine Enabled Wait

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

The Control Program treats one case of an enabled wait in a virtual machine the same as a disabled wait. If the virtual machine does not have the "real timer" option and loads a PSW enabled only for external interrupts, CP issues the message

```
DMKDSP450W   CP ENTERED; DISABLED WAIT STATE
```

Since 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 can cause an external interrupt which will allow processing to resume.

## 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 three 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 has been terminated due to 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 resident 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 due to 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 IPL to restart the system. If the problem persists, notify the 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 the installation 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 overwriting 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 the installation 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 zeroes. This occurs during normal RSCS operation and does not indicate an error condition. An external interrupt due to command entry or an I/O interrupt due to the arrival of files automatically resumes processing.

## SUMMARY OF VM/370 DEBUGGING TOOLS

Figure 6 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 the specified address. CMS allows 16 address stops to be active while CP allows only one | ADSTOP hexloc | DEBUG<br><br>BREAK id {symbol / hexloc} |
| Resume execution. | Resume execution where program was interrupted | BEGIN | DEBUG<br>GO |
| | Continue execution at a specific location | BEGIN hexloc | DEBUG<br><br>GO {symbol / hexloc} |
| Dump data. | Dump the contents of specific storage locations | DUMP {hexloc1 / Lhexloc1} [{- / :} hexloc2 / END] [{. } bytecount / END] [*dumpid] | DEBUG<br>DUMP symbol1 / hexloc1 / 0 [symbol2 / hexloc2 / * / 176] [ident] |

Figure 6. Summary of VM/370 Debugging Tools (Part 1 of 5)

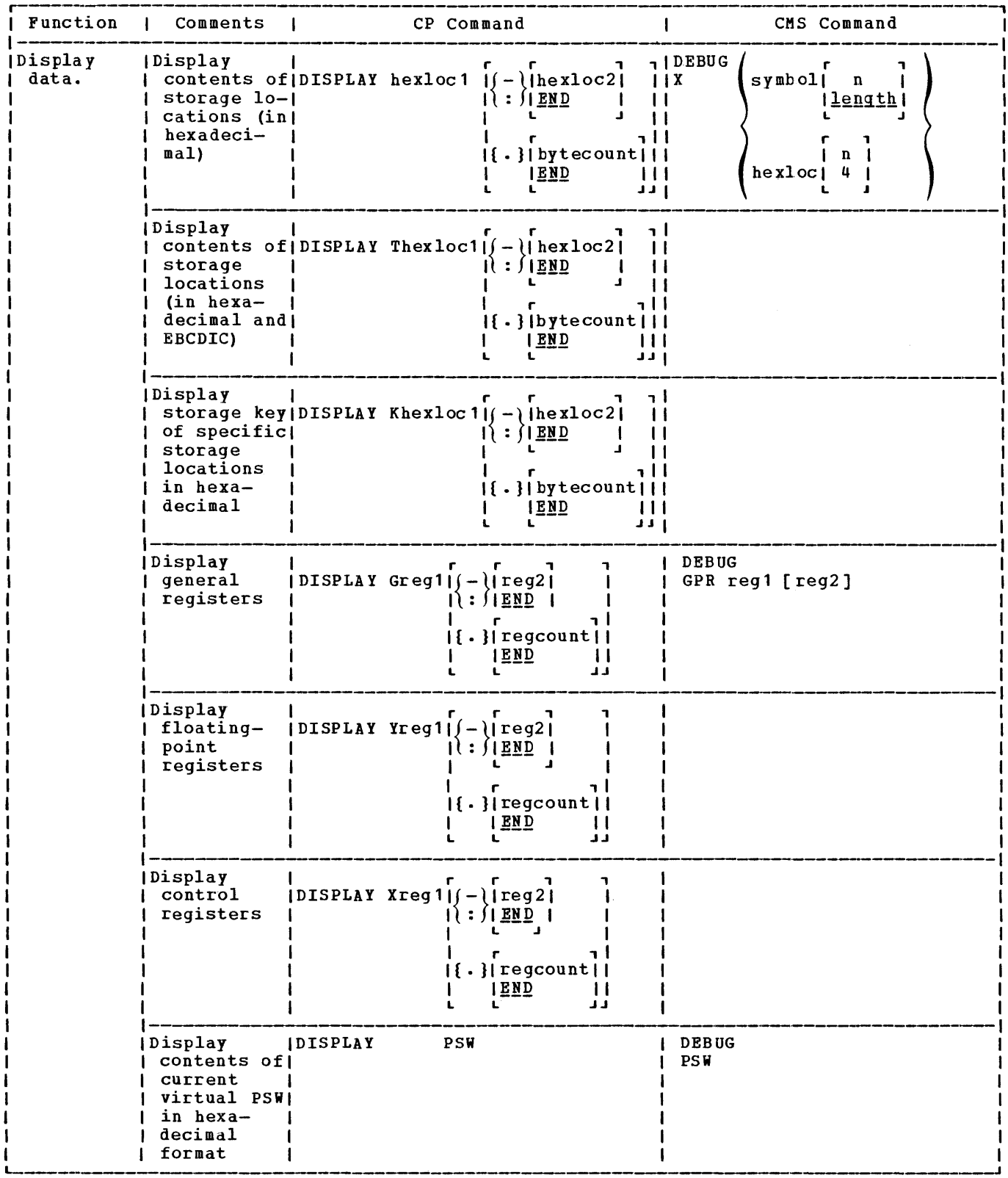



Figure 6. Summary of VM/370 Debugging Tools (Part 2 of 5)

| Function | Comments | CP Command | CMS Command |
|---|---|---|---|
| Display data (cont.) | Display contents of CAW | DISPLAY CAW | DEBUG<br>CAW |
| | Display contents of CSW | DISPLAY CSW | DEBUG<br>CSW |
| Store data. | Store specified information into consecutive storage locations without alignment | STORE Shexloc hexdata... | DEBUG<br>STORE {symbol / hexloc}<br>hexinfo[ hexinfo[ hexinfo]] |
| | Store specified words of information into consecutive fullword storage locations | STORE {hexloc / Lhexloc}<br>{hexword1[ hexword2...]} | |
| | Store specified words of information into consecutive general registers | STORE Greg hexword1<br>[hexword2...] | DEBUG<br>SET GPR reg hexinfo[ hexinfo] |
| | Store specified words of information into consecutive floating-point registers | STORE Yreg hexword1<br>[hexword2...] | |

Figure 6. Summary of VM/370 Debugging Tools (Part 3 of 5)

| Function | Comments | CP Command | CMS Command |
|---|---|---|---|
| Store data (cont.) | Store specified words of data into consecutive control registers | STORE Xreg hexword1 [hexword2...] | |
| | Store information into PSW | STORE PSW [hexword1] hexword2 | DEBUG<br>SET PSW hexinfo [hexinfo] |
| | Store information in CSW | | DEBUG<br>SET CSW hexinfo [hexinfo] |
| | Store information in CAW | | DEBUG<br>SET CAW hexinfo |
| 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 | |

Figure 6. Summary of VM/370 Debugging Tools (Part 4 of 5)

| Function | Comments | CP Command | CMS Command |
|---|---|---|---|
| Trace execution (cont.) | Trace virtual and real CCWs | TRACE SIO<br>TRACE CCW | |
| | Trace all user interrupts and successful branches | TRACE BRANCH | |
| | Trace all instructions | TRACE INSTRUCTION | |
| | End all tracing activity | TRACE END | SVCTRACE OFF |
| Trace real machine events | Trace events in real machine | MONITOR START CPTRACE | |
| | Stop tracing events in the real machine | MONITOR STOP CPTRACE | |

Figure 6. Summary of VM/370 Debugging Tools (Part 5 of 5)

## COMPARISON OF CP AND CMS FACILITIES FOR DEBUGGING

If you are debugging problems while running CMS, you can choose the CP or CMS debugging tools. Refer to Figure 7 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 general and floating-point registers are printed at the beginning of the dump. |
| Displaying the contents of storage and control registers at the terminal. | The display is typed in hexadecimal format with EBCDIC translation. The CP command displays storage keys, floating-point registers and control registers. | The display is typed in hexadecimal format. The CMS commands do not display storage keys, floating-point registers or control registers as the CP command does. |
| Storing information. | The amount of information stored by the CP command is limited only by the length of the input line. The information can be fullword aligned when stored. CP stores data in the PSW, but not in the CAW or CSW. However, data can be stored in the CSW or CAW by specifying the hardware address in the STORE command. CP also stores the status of the virtual machine in the extended logout area. | The CMS command stores up to 12 bytes of information. CMS stores data in the general registers but not in the floating-point or control registers. CMS stores data in the PSW, CAW, and CSW. |
| Tracing information. | CP traces:<br>• All interrupts, instructions and branches<br>• SVC interrupts<br>• I/O interrupts<br>• Program interrupts<br>• External interrupts<br>• Privileged instructions<br>• All user I/O operations<br>• Virtual and real CCW's<br>• All instructions<br><br>The CP trace is interactive. You can stop and display other fields. | CMS traces all SVC interrupts. CMS displays the contents of general and floating-point registers before and after a routine is called. The parameter list is recorded before a routine is called. |

Figure 7. Comparison of CP and CMS Facilities for Debugging

This section contains information you may want to refer to while debugging and a discussion of when and how to use the CP debugging tools. Also included is a discussion of how to read a CP ABEND dump.

The first section, "Introduction to Debugging," described the debugging procedures to follow and this section tells you how to use the debugging tools and commands mentioned in that first section. The following topics are discussed in this section.

- Debugging CP in a virtual machine
- CP commands useful for debugging
- DASD dump restore program
- CP Internal Trace Table
- CP restrictions
- ABEND dumps
- Reading ABEND dumps
- Control block summary

## CP COMMANDS USED TO DEBUG IN 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.

Response

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.

## Using the ADSTOP Command when Debugging

The address stop should be set after the program is loaded, but before it executes. When the specified location is reached during program execution, execution halts and the CP environment is entered. The virtual machine operator may issue other CP commands to examine and alter the status of the program at this time.

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

Note: In order to successfully set an address stop, the virtual instruction address must be in real storage at the time the ADSTOP command is issued.

## 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

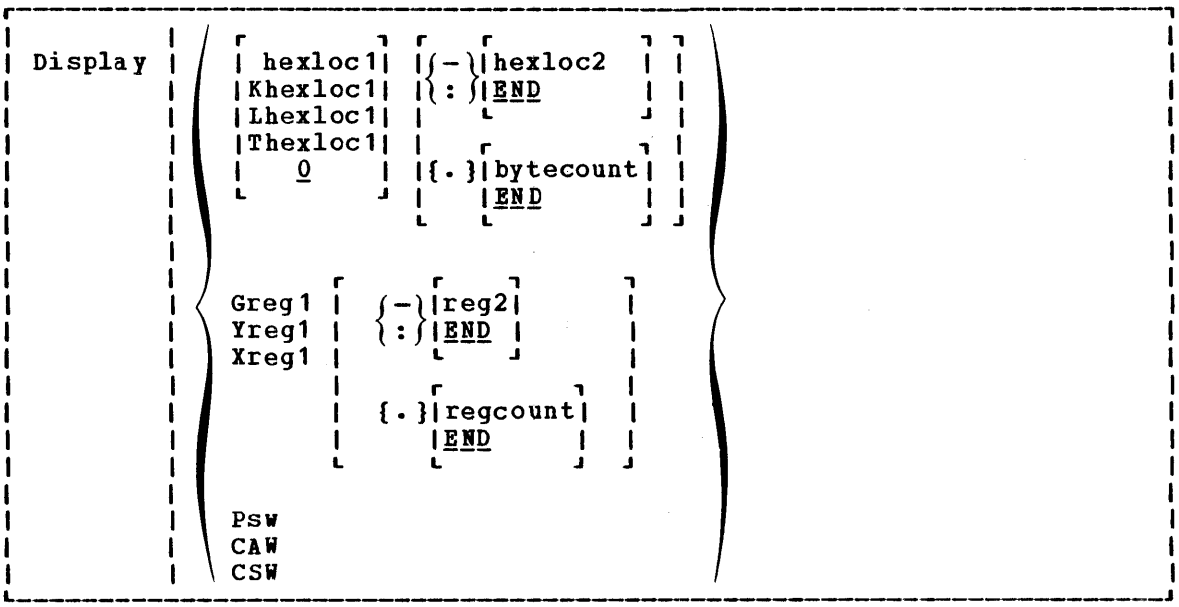
Privilege Class: G

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:




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 not 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  .xxxxxxxxxxxxxxxx  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

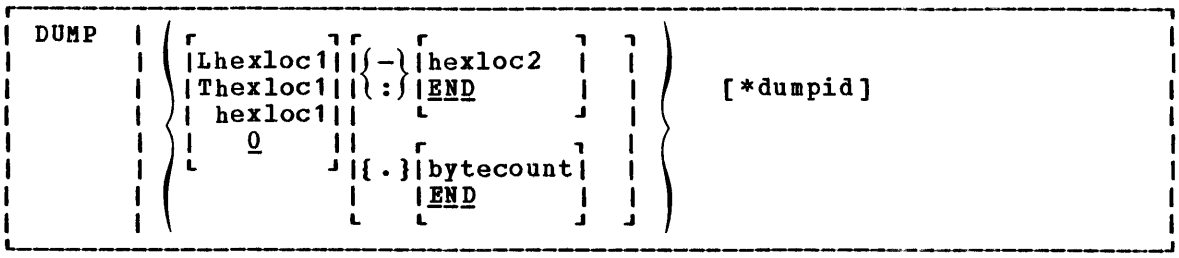
**Privilege Class:** G

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:




**where:**

**Lhexloc1**
**Thexloc1**
**hexloc1**
**0**

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 hexloc1 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**

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.

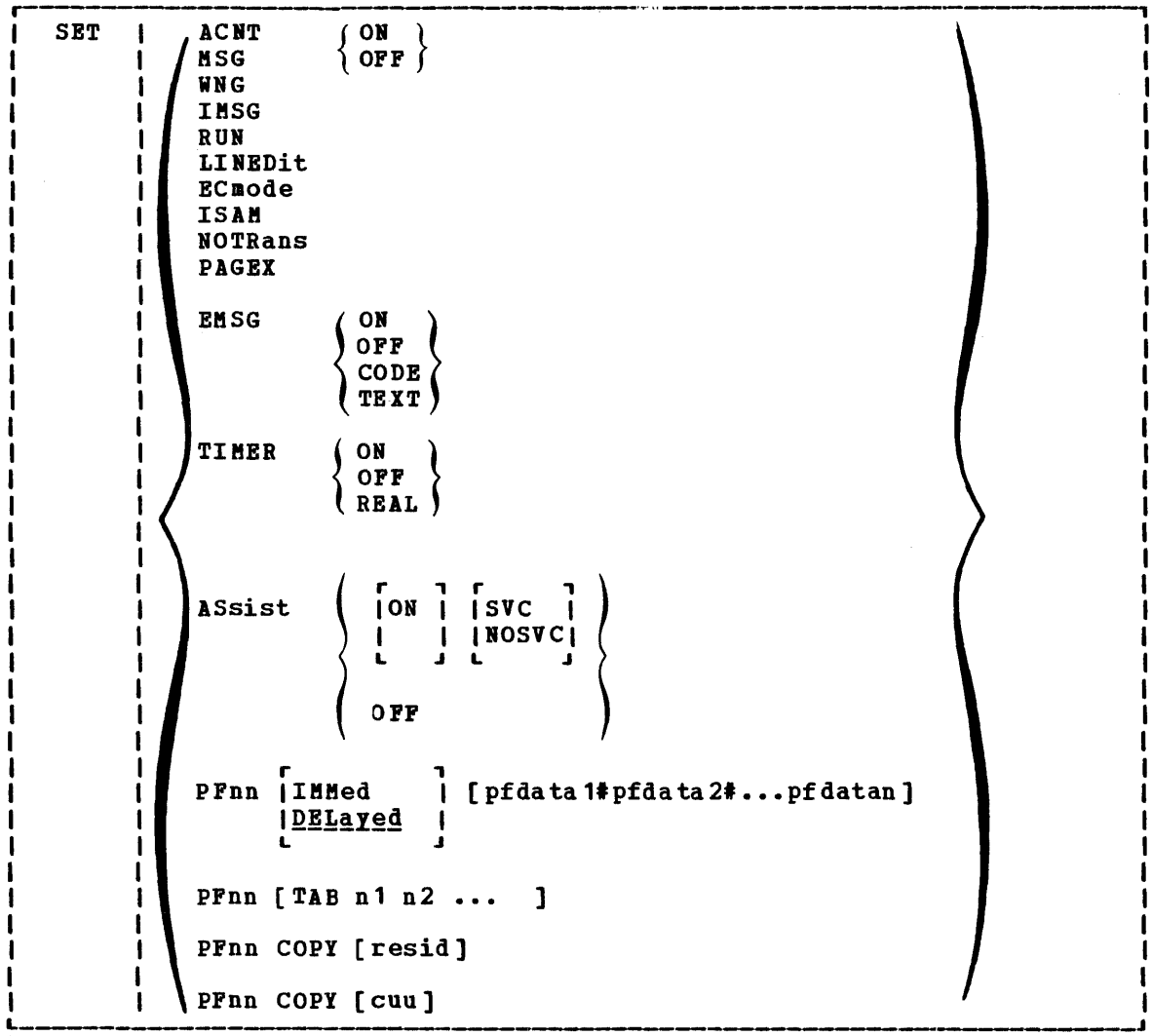
COMMAND COMPLETE

is the response indicating normal completion of the dump function.

## SET

**Privilege Class:** G

Use the SET command to control various functions within your virtual system. The format of the SET command is:




**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 "Virtual Machine Assist Feature" discussion in "Part 2: Control Program (CP)" for information on how to use the assist feature.

```
PFnn ┌        ┐ [pfdata1#pfdata2#...pfdatan]
     |IMMED   |
     |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

Using the SET Command when Debugging

Use the SET command to control various systems options. In particular, set the MSG, WNG, and EMSG options ON when debugging. The messages printing at the terminal may provide information that is immediately helpful.

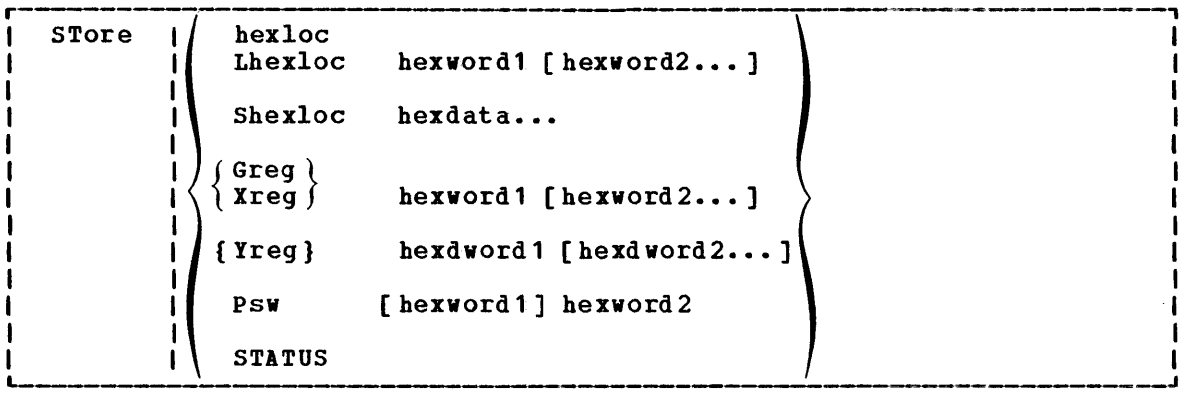
## STORE

**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:




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.

## Using the STORE Command when Debugging

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

One debugging use of STORE STATUS would be as follows:

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

## 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

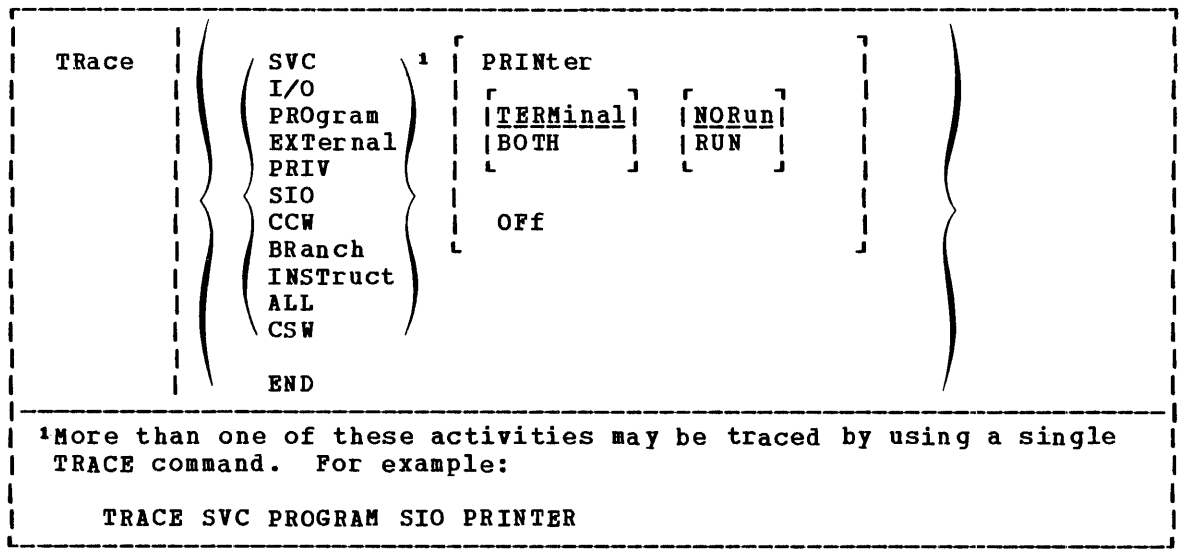
is the response given if the command SYSTEM RESET is entered.

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:




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.

TERMINAL 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.

NORUN stops program execution after the trace output to the terminal and enters the CP command environment.

Note: 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.

Usage Notes

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.

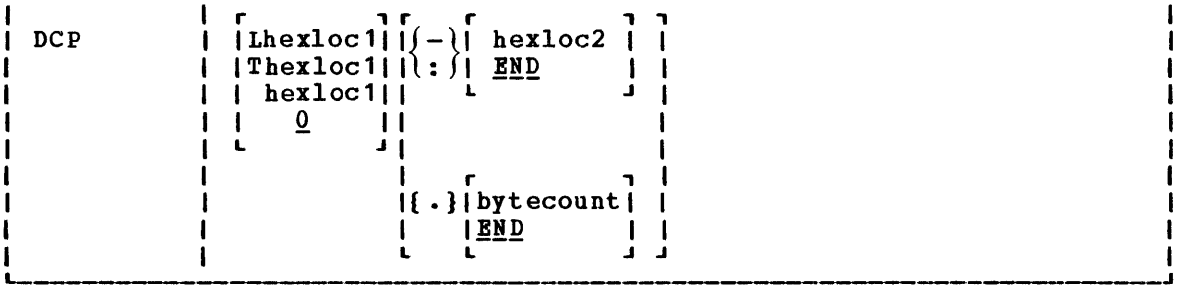
System programmers and analysts may also want to use the CP internal trace table. This table records events that occur on the real machine.

## DCP

**Privelege 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:




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 of 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 1          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 1 . 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).

## Using the DCP Command

Use the DCP command to display real storage locations at the terminal.

The requested locations are typed in the following format:

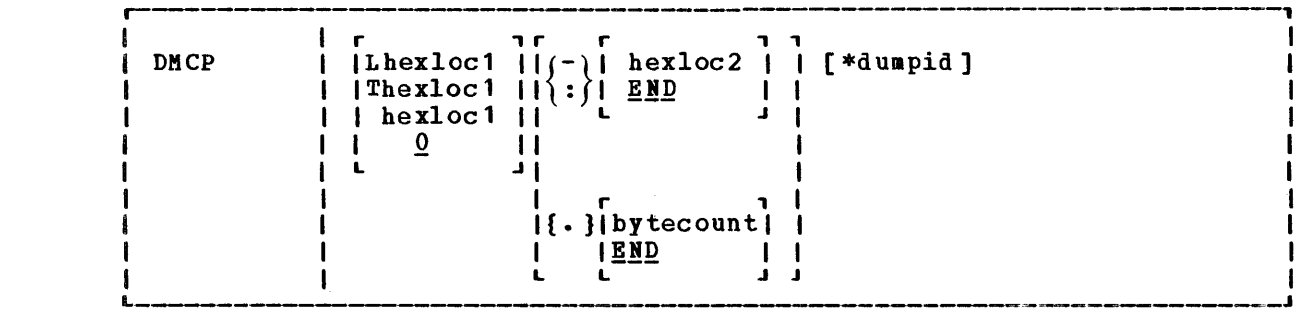
```
xxxxxx = WORD1 WORD2 WORD3 WORD4 [EBCDIC translation]
```

where xxxxxx is the real storage location of WORD1. WORD1 is displayed (word aligned) for a single hexloc specification. Up to four words are displayed on a line. If required, multiple lines are printed. The EBCDIC translation is displayed if Thexloc is specified.

## 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:




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 1        dmcp 1-       dmcp t.       dmcp t-end
dmcp t        dmcp t-       dmcp 0-       dmcp 0:end
dmcp -        dmcp 1:       dmcp 0:       dmcp 1.end
dmcp :        dmcp t:       dmcp 0.       dmcp 1.end
dmcp .        dmcp 1.       dmcp 1-end    dmcp 0.end
```

Each of the following produces three full dumps because of the embedded blanks:

```
dmcp 1 . 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.

## Using the DMCP Command

Use the DMCP command to dump the contents of real storage locations to your virtual spooled printer. The output format is eight words per line with EBCDIC translation. If a dumpid is used, it may be up to 100 characters, including blanks. In order to print the output at the real printer, the virtual spooled printer must be terminated with a CLOSE.

## 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 Blocks. The format of the LOCATE command is:

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
```

### Using the LOCATE Command

Use the LOCATE command to find the addresses of the system control blocks associated with a particular user, a user's virtual device, or a real system device.

Once you know the location of the system control blocks you can examine (dump or display) the block you want to see. When you want to examine specific control blocks, use the commands LOCATE and DUMP or DISPLAY to examine the control blocks, instead of taking a dump. A discussion of the most important fields of the VMBLOK, VCHBLOK, VCUBLOK, VDEVBLOK, RCHBLOK, RCUBLOK, and RDEVBLOK are included in the "Reading CP ABEND Dumps" section.

## 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. Refer to the "CP Internal Trace Table" section of this manual for information on finding and using the internal trace table. 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.

## Using the QUERY Command

The QUERY command tells you the value of the paging activity index and the priority. This information can be useful in evaluating the usefulness of the performance options and in examining dispatching functions.

## SAVENCP

See "Part 4. IBM 3704 and 3705 Communications Controllers" for a description of this command.

## SAVESYS

### Privilege Class: E

Use the SAVESYS command to save a virtual machine storage space with registers and PSW as they currently exist. The format of the SAVESYS command is:

```
┌──────────────────────────────────────────────────────────────────────┐
│ SAVESYS │        systemname                                          │
└──────────────────────────────────────────────────────────────────────┘
```

where:

systemname — must be a predefined name representing a definition of installation requirements of the named system. The definition indicates the number of pages to be saved, the DASD volume on which the system is to be saved, and the shared segments (if any).

### Response

SYSTEM SAVED

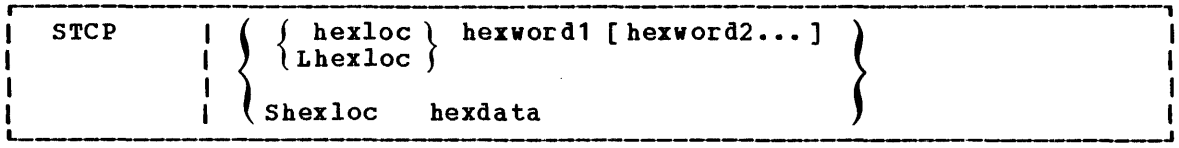
### Using the SAVESYS Command

See the "Generating Named Systems" section of "Part 2. Control Program (CP)" for a complete discussion of when and how to save a named system.

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:




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

Using the STCP Command

Use the STCP command to alter the contents of real storage. The real PSW or real registers may not be altered by this command.

## DASD DUMP RESTORE PROGRAM (STANDALONE VERSION)

The DASD Dump Restore (DDR) program can be run standalone in the real or virtual machine. To run DASD Dump Restore standalone, IPL an input device that contains all the necessary control statements. The control statements necessary to run the DDR program are:

- I/O Definition Statements
- Function Statements

### DDR CONTROL STATEMENTS

Control statements describe the processing that is to take place and the I/O devices that are to be used. I/O definition statements must be specified first.

All control statements may be entered from the system console or a card reader. Only columns 1 to 71 are inspected by the program. All data after the last possible parameter 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 recorded cylinders. 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 detection of 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 must be repeated. Failure to use INPUT or OUTPUT statements to separate function specifications when input is from the card reader or a CMS file will result in an error message (DMKDDR702E) being displayed. 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:).

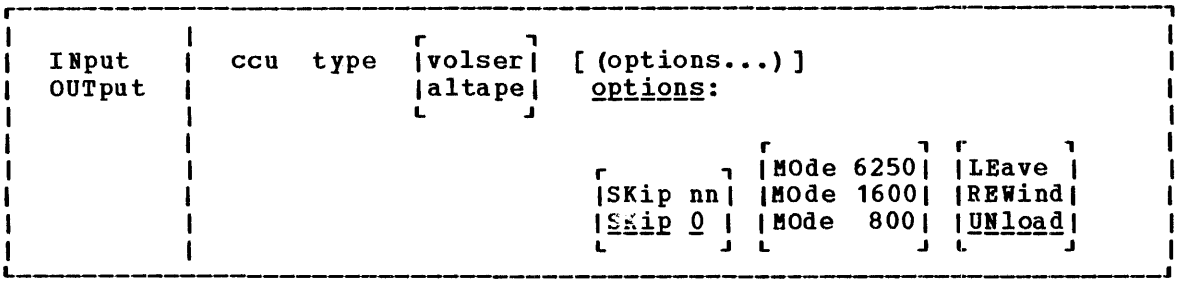
The PRINT and TYPE statements work differently in that they operate on only one data extent at a time. If the input is from a tape created by the dump function, it must be positioned at the header record for each step. The PRINT and TYPE statements have an implied output of either the console (type) 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:




where:

ccu is the unit address of the device.

type is the device type (2314, 2319, 3330, 3330-11, 3340-35 (3340 access device equipped with a 3348-35 megabyte disk pack), 3340-70 (3340 access device equipped with a 3348-70 megabyte disk pack), 3350, 2305-1, 2305-2, 2400, 2420, or 3420). There is no 7 track support.

Note: Specify a 3350 device in 3330-1 compatibility mode as a 3330, and a 3350 in 3330-11 compatibility mode as a 3330-11. Specify a 3333 as a 3330, and a 3340-70F or 3344 as a 3340-70.

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 displays the following at the end of the reel: "END OF VOLUME CYL xxx HD xx, MOUNT NEXT TAPE." After the new tape is mounted, DDR continues automatically.

Options

SKIP nn 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
MODE 1600
MODE 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 mode. All subsequent tapes mounted are also set to the specified mode. If no mode option is specified, then no mode set is performed.

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.

The REWIND, UNLOAD, and LEAVE options are ignored if the wrong tape has been mounted. Message DMKDDR709E is issued when this occurs.

## SYSPRINT Control Statement

Use the SYSPRINT control statement to describe a printer device that is used to print data extents specified by the PRINT statement for the standalone version of DDR. It is also used to 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. The SYSPRINT control statement is used by the standalone version of DDR to define the printer device if it is other than 00E. DDR, running under the control of CMS, ignores this control statement since the CMS printer is 00E. The format of the SYSPRINT control statement is:

```
┌──────────────────────────────────────────────────────────────────────┐
|   SYsprint   |   ccu                                                 |
└──────────────────────────────────────────────────────────────────────┘
```

where:

ccu specifies the unit address of the device.

## Function Statement

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 or end of job 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 as it was dumped from. It is possible to dump from a real disk and restore to a minidisk.

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). If cyl2 is not specified, only the first cylinder (cyl1) is operated on. The REORDER operand causes the output to be reordered, starting at the specified cylinder (cyl3) or at the starting cylinder (cyl1) if (cyl3) is not specified. The REORDER operand may not be used with the CPVOL, ALL, or NUCLEUS operands.

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 disks should be in CP format, that is, they must have been formatted by the CP Format/Allocate program.

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 will be dumped, copied, or restored.

Restrictions:

1. Each track must contain a valid home address, containing the real cylinder and track location.

2. Record zero must not contain more than eight key and/or data characters.

3. For the IBM 2314, 2319, 3340, and 2305, flagged tracks will be treated as any other track , that is, no attempt will be made to substitute the alternate track data when a defective primary track is read. In addition, tracks will not be inspected to determine whether they were previously flagged when written. Therefore, volumes containing flagged tracks should be restored to the volume from which they were dumped. The message DMKDDR715E is displayed each time a defective track is dumped, copied, or restored, and the operation continues.

4. For the IBM 3330 or 3350, flagged tracks are automatically handled by the control unit and should never be detected by the program. However, if a flagged track is detected, message DMKDDR715E is displayed 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 mode 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 room on the first tape, it continues dumping onto the alternate device (181). While dumping, a map of the cylinders dumped is printed on unit 00F. 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 cylinders dumped 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: The cylinder map that is printed on the SYSPRINT device is preceded by a header line containing the date, time, and time zone. This information is controlled by the DMKDDR module and, as distributed, generates Greenwich Meridian Time (GMT). In order to change this to the local time zone, the installation must apply a local modification as described in the DMKDDR source listing.

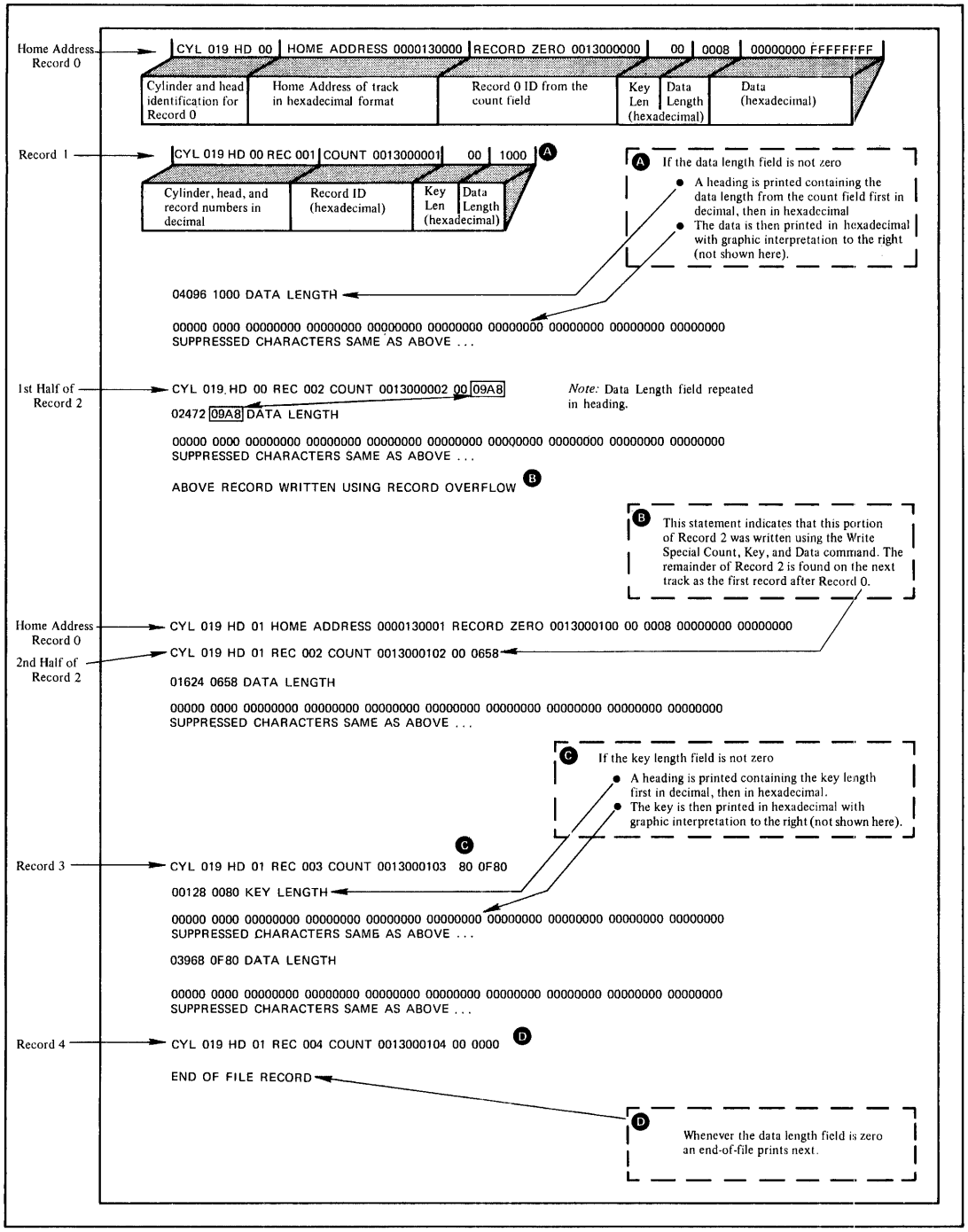



**Figure 8. Annotated Sample of Output from the TYPE and PRINT Functions of the DDR Program**

## PRINT/TYPE Function Statement

Use the PRINT and TYPE function statement to print or 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 | cc1 [hh1 [rr1]] [To cc2 [hh2 [rr2 ]]] [(options)]
| TYpe  |
|       | options:
|       | [Hex] [Graphic] [Count]
```

where:

cc1 is the starting cylinder.

hh1 is the starting track. If present, it must follow the cc1 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] cc2 is the ending cylinder. If more than 1 cylinder is to be printed or displayed "TO cc2" must be specified.

hh2 is the ending track. If present, it must follow the cc2 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 8 shows the information that would be displayed at the console (TYPE function) or system printer (PRINT function) by the DDR program. The listing has been annotated to describe some of the data fields.

## 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 Operating in a Virtual Machine Environment" section of "Part II. Control Program (CP)" for directions for setting up the virtual machine.

## CP INTERNAL TRACE TABLE

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

- External interruptions
- SVC interruptions
- Program interruptions
- Machine check interruptions
- I/O interruptions
- Free storage requests
- Release of free storage
- Entry into scheduler
- Queue drop
- Run user requests
- Start I/O
- Unstack I/O interruptions
- Storing a virtual CSW
- Test I/O
- Halt Device
- Unstack IOBLOK or TRQBLOK
- NCP BTU (Network Control Program Basic Transmission Unit)

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. Figure 9 describes the format of each type of trace table entry.

The trace table is allocated by the main initialization routine, DMKCPI. The first event traced is placed in the lowest trace table address. Each subsequent event is recorded in the next available trace table entry. Once the trace table is full, events are recorded at the

lowest address (overlaying the data previously recorded there). Tracing continues with each new entry replacing an entry from a previous cycle.

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

The CP internal trace table is initialized during IPL. If you do not wish to record events in the trace table, issue the MONITOR STOP command to suppress recording. The pages allocated to the trace table are not released and recording can be restarted at any time by issuing the MONITOR START command. If the VM/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.

| Type of Event | Module | Identification Code (hexadecimal) | Format of Trace Table Entry |
|---|---|---|---|
| External interrupt | DMKPSA | 01 | X'01' (0) \| X'0000000000' (1) \| Interrupt Code (6) \| External Old PSW (8–15) |
| SVC interrupt | DMKPSA | 02 | X'02' (0) \| GR14 or GR15 (See Note 1) (1) \| Instruction Length Code (4) \| Interrupt Code (6) \| SVC Old PSW (8–15) |
| Program interrupt | DMKPRG | 03 | X'03' (0) \| First 3 bytes of VMPSW (1) \| Instruction Length Code (4) \| Interrupt Code (6) \| Program Old PSW (8–15) |
| Machine Check Interrupt | DMKMCH | 04 | X'04' (0) \| Address of VMBLOK (1) \| First 4 bytes of 8-byte Interrupt Code (4) \| Machine Check Old PSW (8–15) |
| I/O interrupt | DMKIOS | 05 | X'05' (0) \| X'00' (1) \| Device Address (2) \| I/O Old PSW + 4 (4) \| CSW (8–15) |
| Free Storage (FREE) | DMKFRE | 06 | X'06' (0) \| Address of VMBLOK (1) \| GR 0 at entry (4) \| GR 1 at exit (8) \| GR 14 (12–15) |
| Return storage (FRET) | DMKFRE | 07 | X'07' (0) \| Address of VMBLOK (1) \| GR 0 at entry (4) \| GR 1 at entry (8) \| GR 14 (12–15) |
| Enter Scheduler | DMKSCH | 08 | X'08' (0) \| Address of VMBLOK (1) \| Value of VMRSTAT, VMDSTAT, VMOSTAT, and VMQSTAT (4) \| VMQLEVEL VMTLEVEL (8) \| VMIOINT (10) \| VMPEND (12) \| (13–15) |
| Queue drop | DMKSCH | 09 | X'09' (0) \| Address of VMBLOK (1) \| X'0000' (4) \| New Priority (6) \| Number of Resident Pages (8) \| Projected Working Set (10) \| Number of Referenced Pages (12) \| Current Page load (PSA) (14–15) |
| Run user | DMKDSP | 0A | X'0A' (0) \| X'000000' (1) \| RUNUSER value from PSA (4) \| RUNPSW value from PSA (8–15) |
| Start I/O | DMKCNS DMKIOS DMKVIO | 0B | X'0B' (0) \| Condition Code (1) \| Device Address (2) \| Address of IOBLOK (4) \| CAW (8) \| For CC = 1, CSW + 4 otherwise this field is not used (12–15) |
| Unstack I/O interrupt | DMKDSP | 0C | X'0C' (0) \| X'00' (1) \| Virtual Device Address (2) \| Address of VMBLOK (4) \| Virtual CSW (8–15) |
| Virtual CSW store | DMKVIO | 0D | X'0D' (0) \| Instruction Operation Code (1) \| Virtual Device Address (2) \| Address of VMBLOK (4) \| Virtual CSW (8–15) |
| Test I/O | DMKCNS DMKIOS DMKVIO | 0E | X'0E' (0) \| Condition Code (1) \| Device Address (2) \| Address of IOBLOK (4) \| CAW (8) \| For CC = 1, CSW + 4 otherwise this field is not used (12–15) |
| Halt Device | DMKCNS DMKIOS DMKVIO | 0F | X'0F' (0) \| Condition Code (1) \| Device Address (2) \| Address of IOBLOK (4) \| CAW (8) \| For CC = 1, CSW + 4 otherwise this field is not used (12–15) |
| Unstack IOBLOK or TRQBLOK | DMKDSP | 10 | X'10' (0) \| Address of VMBLOK (1) \| Value of VMRSTAT, VMDSTAT, VMOSTAT, and VMQSTAT (4) \| Address of IOBLOK or TRQBLOK (8) \| Interrupt Return Address (12–15) |
| NCP BTU (See Note 2) | DMKRNH | 11 | X'11' (0) \| X'00' (1) \| CONSRID (2) \| CONDEST (4) \| CONRTAG (6) \| CONSYSR CONEXTR (8) \| CONTCMD (10) \| CONFUNC CONDFLG (12) \| CONDCNT (14–15) |

*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 9. CP Trace Table Entries**

## 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 Part 2.

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 OFF, 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 discontiguous 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).

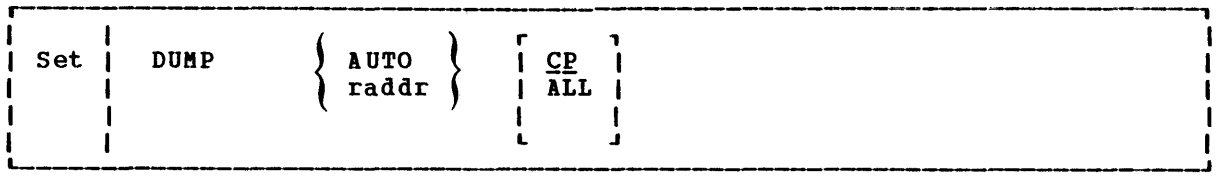
5. If an I/O device (such as a disk or tape drive) drops ready status while it is processing virtual I/O activity, any virtual machine users performing I/O on that device are unable to continue processing or to log off. Also, the LOGOFF and FORCE commands are not effective because they do not complete 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.

## ABEND DUMPS

There are three kinds of abnormal termination dumps possible when using CP. If the problem program cannot continue, it terminates and in some cases attempts to issue a dump. Likewise, if the operating system for your virtual machine cannot continue, it terminates and, in some cases, attempts to issue a dump. In the VM/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.

The third type of dump occurs when the CP system cannot continue. The CP abnormal termination dumps can be directed to a printer or tape or be dynamically allocated to DASD. If the dump is directed to a tape,

the dumped data must fit on one reel of tape. Multiple tape volumes are not supported by VM/370. The historical data on the tape is in print line format and can be processes by user created programs or via CMS commands. Specify the output device for CP ABEND dumps with the CP SET command. The format of the SET command used is:




where:

AUTO automatically directs the ABEND dump to disk.

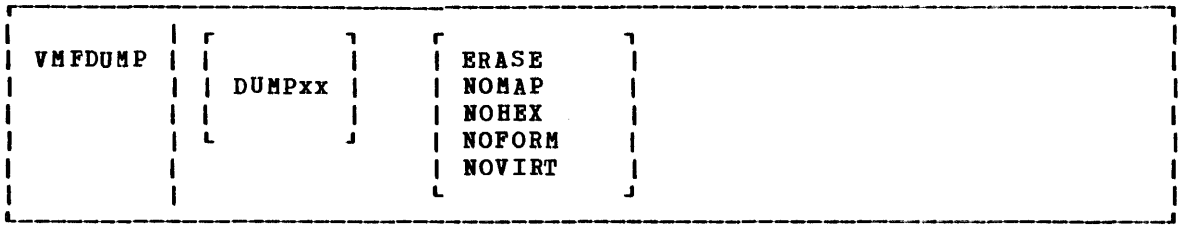
raddr directs the ABEND dump to the specified unit address (either a printer or a tape unit). If the address specifies a tape device, the dump data must fit on one reel; VM/370 does not support multiple tape volumes.

CP dumps only the CP storage area.

ALL dumps all of real storage.

## USING THE VMFDUMP COMMAND

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




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 will 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 will be 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 long, unblocked, and contain carriage control characters.

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

```
FILEDEF ddname1 PRINTER (RECFM F LRECL 131)

FILEDEF ddname2 {TAP2} (9-track DEN 1600 RECFM F LRECL 131 BLOCK 131)
                {TAP1}
```

Then use the MOVEFILE command to print the tape:

```
MOVEFILE ddname2 ddname1
```

An extended form of the VMFDUMP command may be used via the facilities of IPCS, the Interactive Problem Control System, by Field Engineering Program Systems Representatives and installation system programmers. For information on IPCS, refer to the publication VM/370: Interactive Problem Control System (IPCS) User's Guide.

## 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

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 the Control Program can no longer continue and abnormally terminates, you must first determine the condition that caused the ABEND, and then find the cause of that condition. You should know the structure and function of the Control Program. "Part 2: Control Program (CP)" contains information that will help you understand the major functions of CP. The following discussion on reading CP dumps includes many references to CP control blocks and control block fields. Refer to VM/370: Data Areas and Control Blocks Logic for a description of the CP control blocks. Figure 11 shows the relationships of the CP control blocks. Also, you will need the current load map for CP to be able to identify the modules from their locations.

## 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.

1. 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." Figure 54 in "Appendix A: System/370 Information" describes the format of an Extended Control PSW.

2. Examine the SVC old PSW, the SVC interrupt code, and the ABEND code to find 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 | DMKNLD | DMKSCH |
| DMKCKS | DMKPGT | DMKTDK |
| DMKCPI | DMKPRG | DMKUDR |
| DMKCVT | DMKPSA | DMKVDB |
| DMKDRD | DMKPTR | DMKVDR |
| DMKDSP | DMKRGF | DMKVIO |
| DMKFRE | DMKRNH | DMKVMA |
| DMKHVC | DMKRPA | DMKVSP |
| DMKIOS | | |

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 Figure 10 for the possible values of CPABEND.

Use the CP system load map to identify the module issuing the SVC 0. If you cannot find the module using the CP system load map, refer to "Identifying a Pageable Module". Figure 54 in Appendix A describes the format of an Extended Control PSW.

3. 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. Figure 54 in Appendix A describes the format of an Extended Control PSW.

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

| 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:<br>X'01' LOAD SYS<br>X'02' FIND SYS<br>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 10. 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 standalone 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 10. 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 10. 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 10. 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 10. 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 10. 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 10. 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 10. 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 10. 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 (LRA) LOAD REAL ADDRESS 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 10. 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:<br>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.<br>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 10. 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 embedded teleprocessing code of 02, 03, or 06. | IPL to restart the system. If the problem persists, call IBM for software 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 10. 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. |
| RPA002 | The virtual address supplied to DMKRPAPT is outside of the virtual storage being referenced. | 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. |
| 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 10. 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 |
| TDK002 | A program is attempting to deallocate cylinder(s) of T-disk space that are not marked allocated. | 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. |
| 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 10. CP ABEND Codes (Part 14 of 15)

| ABEND Code | Reason for ABEND | Action |
|---|---|---|
| VIO002 (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. |
| VIO003 | DMKIOS 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 10. CP ABEND Codes (Part 15 of 15)

## 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.

1. CP Running Status Field

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

| Value of CPSTAT | Comments |
|---|---|
| X'80' | CP is in wait state |
| X'40' | CP is running the user in RUNUSER |
| X'20' | CP is executing a stacked request |

2. Current User

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

Also, examine the CP internal trace table to determine the events that preceded the abnormal termination. Start with the last event recorded in the trace table and 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 time. 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. And, 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. Figure 9, earlier in this section, describes the format of each of the 16 possible types of trace table entries.

## REGISTER USAGE

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

The 16 general registers have many uses that vary depending upon the operation. The following table shows the general use of some of the general registers.

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

The following general registers always contain the same information.

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

Use these registers along with the CP control blocks and the data in the Prefix Storage Area to determine the error that caused the CP ABEND.

## SAVE AREA CONVENTIONS

There are three save areas that may be helpful in debugging CP. If a module was called by an SVC, examine the SAVEAREA. SAVEAREA is not in the PSA; the address of the SAVEAREA is found in general register 13. If a module was called by a branch and link, the general registers are saved in the PSA in an area called BALRSAVE (X'240'). The DMKFRE save area and work area is also in the PSA: these areas are 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 backwards and find the previous module executed.

1. SAVEAREA

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

2. BALRSAVE

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

3. FREESAVE

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

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

## VIRTUAL AND REAL CONTROL BLOCK STATUS

Examine the virtual and real control blocks for more information on the status of the CP system. Figure 11 describes the relationship of the CP control blocks; several are described in detail in the following paragraphs.

### VMBLOK

The address of the VMBLOK is in general register 11.

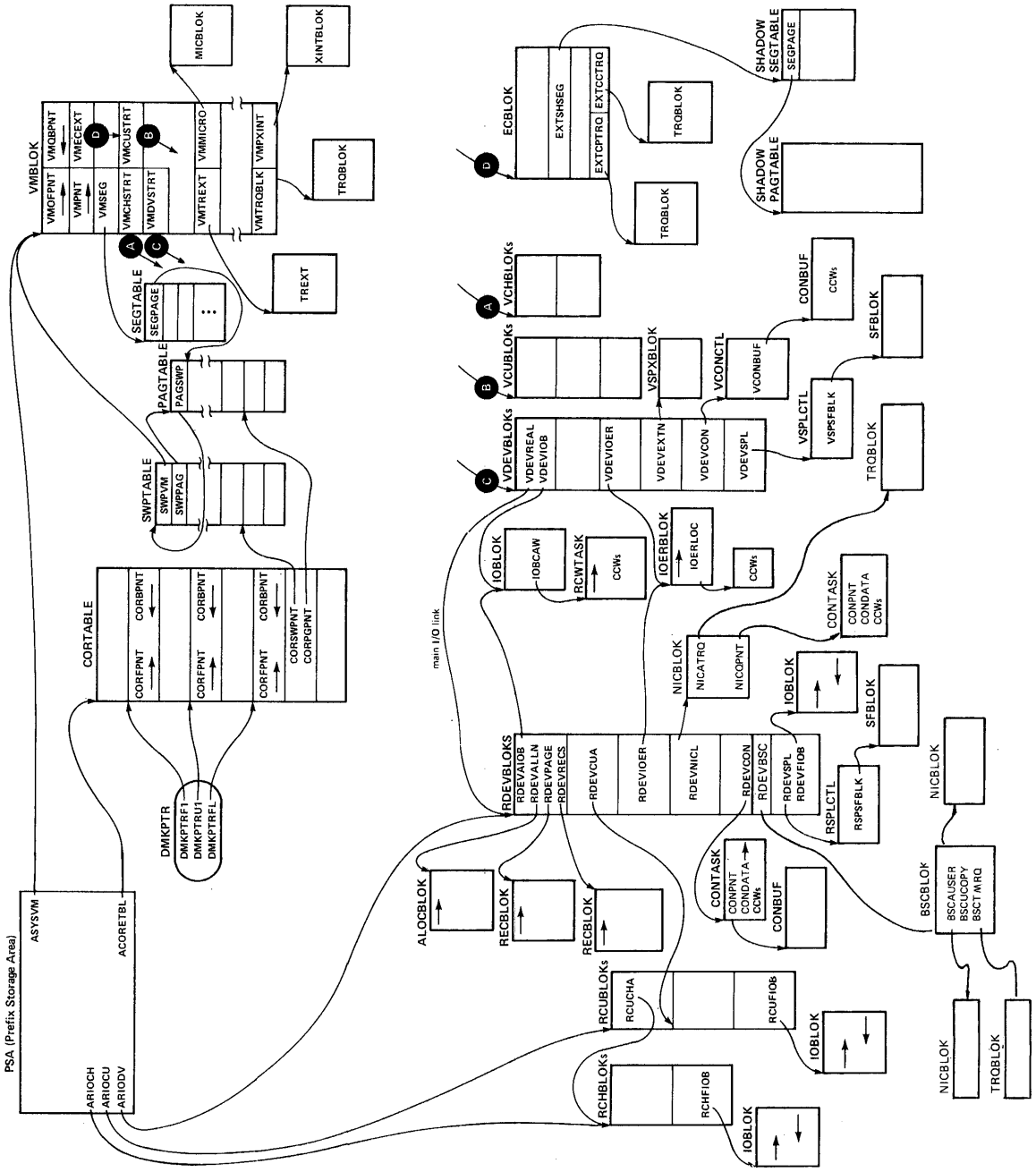
Examine the following VMBLOK fields:

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

| Value of VMRSTAT | Comments |
|---|---|
| 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 |

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

| Value of VMDSTAT | Comments |
|---|---|
| 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 runnable |
| X'04' | Virtual machine in a queue |




**| Figure 11. CP Control Block Relationships**

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

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

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

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

| Value of VMPEND | Comments |
|---|---|
| X'40' | Virtual PER (Program Event Recording) interrupt pending |
| X'20' | Virtual program interrupt deferred |
| X'10' | Virtual SVC interrupt deferred |
| X'02' | Virtual I/O interrupt pending |
| X'01' | Virtual external interrupt pending |

   b. VMEXTINT (displacement X'68') contains the external interrupt pending flags. The value of the flag identifies the external interrupt.

| Value of VMEXTINT | Comments |
|---|---|
| X'08' | Clock comparator interrupt pending |
| X'04' | CPU timer interrupt pending |

| Value of VMEXTINT +1 | Comments |
|---|---|
| X'80' | Interval timer interrupt pending |
| X'40' | Operator's external button interrupt pending |
| X'2F' | External signals pending |

   c. 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.

| Value of VMIOINT | Comments |
|---|---|
| 10000000 00000000 | Interrupt pending channel 0 |
| 01000000 00000000 | Interrupt pending channel 1 |
| . | . |
| . | . |
| . | . |
| 00000000 00000001 | Interrupt pending channel 15 |

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

## VCHBLOK

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

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

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

| Value of VCHSTAT | Comments |
|---|---|
| X'80' | Virtual channel busy |
| X'40' | Virtual channel class interrupt pending |
| X'01' | Virtual channel dedicated |

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

| Value of VCHTYPE | Comments |
|---|---|
| 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 fields:

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

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

| Value of VCUSTAT | Comments |
|---|---|
| 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 |

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

| Value of VCUTYPE | Comments |
|---|---|
| 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 fields:

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

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

| Value of VDEVSTAT | Comments |
|---|---|
| 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 |

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

| Value of VDEVFLAG | Comments |
|---|---|
| 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 |

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

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

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

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

| Value of VDEVCFLG | Comments |
|---|---|
| X'80' | User signalled 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 |

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

| Value of VDEVSFLG | Comments |
|---|---|
| X'80' | Spool reader -- last command was a feed |
| X'80' | Spool output -- transferred 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 |

9. 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 fields:

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

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

| Value of RCHSTAT | Comments |
|---|---|
| X'80' | Channel busy |
| X'40' | IOB scheduled on channel |
| X'20' | Channel disabled |
| X'01' | Channel dedicated |

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

| Value of RCHTYPE | Comments |
|---|---|
| X'80' | Selector channel |
| X'40' | Block multiplexer channel |
| X'20' | Byte multiplexer channel |
| X'01' | S/370 type channel (S/370 instruction support) |

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

## RCUBLOK

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

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

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

| Value of RCUSTAT | Comments |
|---|---|
| X'80' | Control unit busy |
| X'40' | IOB scheduled on control unit |
| X'20' | Control unit disabled |
| X'01' | Control unit dedicated |

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

| Value of RCUTYPE | Comments |
|---|---|
| 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 |

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

## RDEVBLOK

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

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

2. The values of the RDEVSTAT (displacement X'04') and RDEVSTA2 (displacement X'45') fields describe the status of the real device:

| Value of RDEVSTAT | Comments |
|---|---|
| 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'01' | Dedicated device (attached to a user) |

| Value of RDEVSTA2 | Comments |
|---|---|
| X'80' | Active device is being reset |
| X'40' | Device is busy with the channel |
| X'20' | Contingent connection present |

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

| Value of RDEVFLAG | Comments |
|---|---|
| X'80' | DASD -- ascending order seek queuing |
| 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 |
| 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 |

4. The value of the RDEVTYPC field (displacement X'06') describes the device type class and the value of the RDEVTYPE field (displacement X'07') describes the device type. Refer to Figure 12 for the list of possible device type class and device type values.

5. The RDEVAIOB field (displacement X'24') contains the address of the active IOBLOK.

6. The RDEVUSER field (displacement X'28') points to the VMBLOK for a dedicated user.

7. The RDEVATT field (displacement X'2C') contains the attached virtual address.

8. The RDEVIOER field (displacement X'48') contains the address of the IOERBLOK for the last CP error.

9. For spooling unit record devices, the RDEVSPL field (displacement X'18') points to the active RSPLCTL block.

10. For real 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 for re-enable. Also, the RDEVMAX field (displacement X'2E') is the highest valid NCP resource name and the RDEVNCP field (displacement X'30') is the reference name of the active 3705 NCP.

11. For terminal devices, additional flags are defined. The value of the RDEVTFLG field (displacement X'3E') describes the additional flags:

| Value of RDEVTFLG | Comments |
|---|---|
| X'80' | Terminal -- logon process has been initiated |
| X'40' | Terminal -- terminal in reset process |
| X'20' | Terminal -- suppress attention signal |
| X'80' | Graphic -- screen full, in hold status |
| 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 -- CP command interrupt pending |

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

| Value of RDEVTMCD | Comments |
|---|---|
| X'10' | UASCII -- 8 level |
| X'0C' | APL correspondence |
| X'08' | APL PTTC/EBCD |
| X'04' | Correspondence |
| X'00' | PTTC/EBCD |

DEVICE CLASS CODES (Column 33 in Accounting Card)

| Code | Device Class |
|---|---|
| X'80' | Terminal Device |
| X'40' | Graphics Device |
| X'20' | Unit Record Input Device |
| X'10' | Unit Record Output Device |
| X'08' | Magnetic Tape Device |
| X'04' | Direct Access Storage Device |
| X'02' | Special Device |

DEVICE TYPE CODES (Column 34 in Accounting Card)

- For Terminal Device Class

| Code | Device Type |
|---|---|
| X'40' | 2700 Binary Synchronous Line |
| X'40' | 2955 Communication Line |
| X'20' | Telegraph Terminal Control Type II |
| X'20' | Teletype Terminal |
| X'10' | IBM Terminal Control Type I |
| X'18' | IBM 2741 Communication Terminal |
| X'18' | IBM 3767 Communication Terminal |
| X'14' | IBM 1050 Data Communication System |
| X'1C' | Undefined Terminal Device |
| X'00' | IBM 3210 Console |
| X'00' | IBM 3215 Console |
| X'00' | IBM 2150 Console |
| X'00' | IBM 1052 Console |
| X'00' | IBM 7412 Console |

- For Graphics Device Class

| Code | Device Type |
|---|---|
| X'80' | IBM 2250 Display Unit |
| X'40' | IBM 2260 Display Station |
| X'20' | IBM 2265 Display Station |
| X'10' | IBM 3066 Console |
| X'08' | IBM 1053 Printer |
| X'04' | IBM 3277 Display Station |
| X'02' | IBM 3284 Printer |
| X'02' | IBM 3286 Printer |

Figure 12. CP Device Classes, Types, Models, and Features (Part 1 of 3)

- For Unit Record Input Device Class

| Code | Device Type |
|---|---|
| X'80' | Card Reader |
| X'81' | IBM 2501 Card Reader |
| X'82' | IBM 2540 Card Reader |
| X'84' | IBM 3505 Card Reader |
| X'88' | IBM 1442 Card Reader/Punch |
| X'90' | IBM 2520 Card Reader/Punch |
| X'40' | Timer |
| X'20' | Tape Reader |
| X'21' | IBM 2495 Magnetic Tape Cartridge Reader |
| X'22' | IBM 2671 Paper Tape Reader |
| X'24' | IBM 1017 Paper Tape Reader |

- For Unit Record Output Device Class

| Code | Device Type |
|---|---|
| X'80' | Card Punch |
| X'82' | IBM 2540 Card Punch |
| X'84' | IBM 3525 Card Punch |
| X'88' | IBM 1442 Card Punch |
| X'90' | IBM 2520 Card Punch |
| X'40' | Printer |
| X'41' | IBM 1403 Printer |
| X'42' | IBM 3211 Printer |
| X'44' | IBM 1443 Printer |
| X'20' | Tape Punch |
| X'24' | IBM 1018 Paper Tape Punch |

- For Magnetic Tape Device Class

| Code | Device Tape |
|---|---|
| X'80' | IBM 2401 Tape Drive |
| X'40' | IBM 2415 Tape Drive |
| X'20' | IBM 2420 Tape Drive |
| X'10' | IBM 3420 Tape Drive |
| X'08' | IBM 3410/3411 Tape Drive |

- For Direct Access Storage Device Class

| Code | Device Type |
|---|---|
| X'80' | IBM 2301 Parallel Drum |
| X'80' | IBM 2303 Serial Drum |
| X'80' | IBM 2311 Disk Storage Drive |
| X'80' | IBM 2321 Data Cell Drive |
| X'40' | IBM 2314 Disk Storage Facility |
| X'40' | IBM 2319 Disk Storage Facility |
| X'10' | IBM 3330 Disk Storage Facility |
| X'10' | IBM 3333 Disk Storage and Control |
| X'08' | IBM 3350 Disk Storage Facility |
| X'02' | IBM 2305 Fixed Head Storage Device |
| X'01' | IBM 3340 Disk Storage Facility |

Figure 12. CP Device Classes, Types, Models, and Features (Part 2 of 3)

- For Special Device Class

| Code | Device Type |
|---|---|
| X'80' | Channel-to-Channel Adapter (CTCA) |
| X'40' | 3704/3705 Programmable Communications Controller |
| X'01' | Device unsupported by VM/370 |

MODEL CODES (Column 35 in Accounting Card)

As specified in the RDEVICE macro at system generation.

FEATURE CODES (Column 36 in Accounting Card)

- For Printer Devices

| Code | Feature |
|---|---|
| X'01' | UCS |

- For Magnetic Tape Devices

| Code | Feature |
|---|---|
| X'80' | 7-Track |
| X'40' | Dual Density |
| X'20' | Translate |
| X'10' | Data Conversion |

- For Direct Access Storage Devices

| Code | Feature |
|---|---|
| X'80' | Rotational Position Sensing (RPS) |
| X'40' | Extended Sense Bytes (24 bytes) |
| X'20' | Top Half of 2314 Used as 2311 |
| X'10' | Bottom Half of 2314 Used as 2311 |
| X'08' | 35MB Data Module (mounted) |
| X'04' | 70MB Data Module (mounted) |
| X'02' | Reserved/Release are valid CCW operation codes |

- For special devices

| Code | Feature |
|---|---|
| X'10' | Type I channel adapter for 3704/3705 |
| X'20' | Type II channel adapter for 3704/3705 |

Figure 12. CP Device Classes, Types, Models, and Features (Part 3 of 3)

## 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 tells you where the end of the resident nucleus is.

Go to the address indicated in the PSW. Backtrack to the beginning of that page frame. The first eight bytes of that page frame (the page frame containing the address pointed to by the PSW) contains the name of the 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.

# 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
- Load maps
- Reading CMS dumps
- Control block summary

## CMS DEBUGGING COMMANDS

CMS provides two commands that are useful in debugging: DEBUG and SVCTRACE. Both commands execute from the terminal.

The debug environment is entered whenever:

- The DEBUG command is issued
- A breakpoint is reached
- An external or program interrupt occurs

CMS will not accept other commands while in the debug environment. However, while in the debug environment, the options of the DEBUG command can:

- Set breakpoints (address stops) which stop program execution at specific locations.
- Display the contents of the CAW (channel address word), CSW (channel status word), old PSW (program status word), or general registers at the terminal.
- Change the contents of the control words (CAW, CSW and PSW) and general registers.
- Dump all or part of virtual storage at the printer.
- Display the contents of up to 56 bytes of virtual storage at the terminal.
- Store data in virtual storage locations.
- Allow an origin or base address to be specified for the program.
- Assign symbolic names to specific storage locations.
- Close all open files and I/O devices and update the master file directory.
- Exit from the debug environment.

The SVCTRACE command records information for all SVC calls. When the trace is terminated, the information recorded up to that point is printed at the system printer.

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

## DEBUG

The DEBUG command provides support for debugging programs at a terminal. The virtual machine operator can stop the program at a specified location 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. The DEFINE subcommand must be used to create all entries in the DEBUG symbol table.

3. The DEBUG subcommands can be truncated. The following is a list of all valid DEBUG subcommands and their minimum truncation.

| Subcommand | Minimum Truncation |
|---|---|
| BREAK | BR |
| CAW | CAW |
| CSW | CSW |
| DEFINE | DEF |
| DUMP | DU |
| GO | GO |
| GPR | GPR |
| HX | HX |
| ORIGIN | OR |
| PSW | PSW |
| RETURN | RET |
| SET | SET |
| STORE | ST |
| X | X |

One way to enter the debug environment is to issue the DEBUG command. The message

```
DMSDBG728I DEBUG ENTERED
```

appears at the terminal. Any of the DEBUG subcommands may be entered. To continue normal processing, issue the RETURN subcommand.

Whenever a program check occurs, the DMSABN routine gains control. Issue the DEBUG command at this time if you wish CMS to enter 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 interrupt, 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

The format of the BREAK subcommand is:

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.

Use the BREAK subcommand to set breakpoints that 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.

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. The virtual machine operator 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 the virtual machine operator finishes analyzing the program at this point in its execution, he issues the GO subcommand to resume program execution.

## Setting Breakpoints

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 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 the user's 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 typed, the keyboard is unlocked to accept any DEBUG subcommands except RETURN. A breakpoint is cleared when it is encountered during program execution.

It is the responsibility of the user 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 user'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

The user entered more than two operands.

## CAW

The format of the CAW subcommand is:

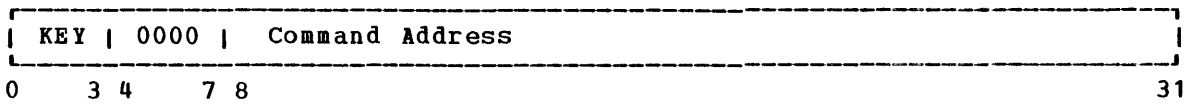
```
┌──────────────────────────────────────────────────────────────────────┐
│ CAW │                                                                │
└──────────────────────────────────────────────────────────────────────┘
```

The CAW subcommand has no operands.

The CAW subcommand may be issued whenever the debug environment is entered. 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 typed in hexadecimal representation at the terminal.

The format of the CAW is:

```
┌──────────────────────────────────────────────────────────────────────┐
│ KEY │ 0000 │  Command Address                                        │
└──────────────────────────────────────────────────────────────────────┘
0    3 4    7 8                                                      31
```



| Bits | Contents |
|---|---|
| 0-3 | The protection key for all commands associated with Start I/O. The protection key in the CAW is compared to 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 in order 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 user, 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.

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 you can examine it.

## 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

The format of the CSW subcommand is:

```
┌──────────────────────────────────────────────────────────────────────┐
|  CSW  |                                                              |
└──────────────────────────────────────────────────────────────────────┘
```

The CSW subcommand has no operands.

The CSW subcommand may be issued whenever the debug environment is entered. 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 is:

```
┌──────────────────────────────────────────────────────────────────────┐
|KEY|0000|   Command Address  |  Status  |  Byte Count                |
└──────────────────────────────────────────────────────────────────────┘
0  3 4  7 8                  31 32      47 48                        63
```

| 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 number of bytes transferred to the I/O device. |

Whenever an I/O operation abnormally terminates, issue the CSW subcommand. 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).

## Error Messages

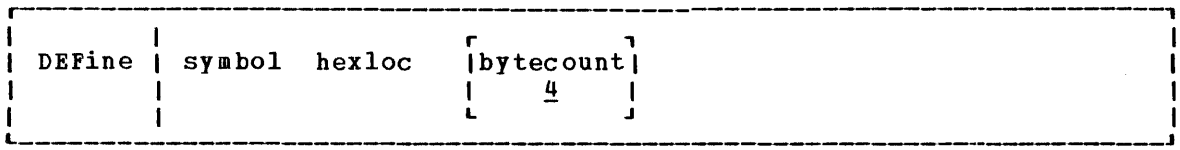
The following error message may appear when you enter the CSW subcommand.

TOO MANY OPERANDS

An operand was entered on the command line; the CSW subcommand has no operands.

DEFINE

The format of the DEFINE subcommand is:




where:

symbol — is the name to be assigned to the storage address derived from the second operand, hexloc.

hexloc — is the hexadecimal storage location, in relation to the current origin, to which the name specified in the first operand (symbol), is assigned.

bytecount — is a decimal number, between 1 and 56 inclusive, 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.

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 be used to refer to that address in any of the other DEBUG subcommands. However, the symbol is valid only in the debug environment.

The first operand (symbol) may be from one to eight 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.

The second operand (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. In order 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.

The third operand (bytecount), a decimal number between 1 and 56 inclusive, specifies the length of the field whose name is specified by symbol and whose starting address is specified by hexloc.

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.

## 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 user's virtual 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

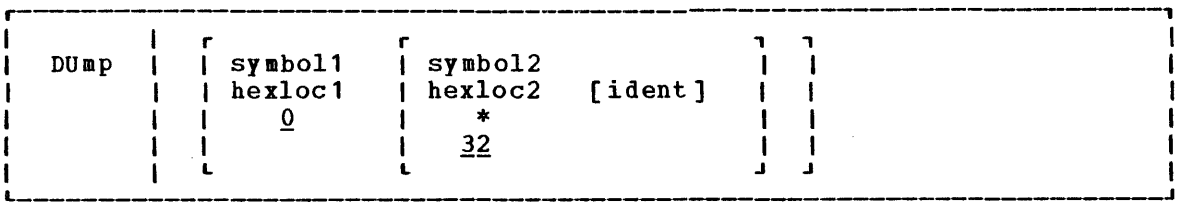
The DEFINE subcommand requires at least two operands and less than two were entered.

TOO MANY OPERANDS

Three is the maximum number of operands for the DEFINE subcommand and more than three were entered.

DUMP

The format of the DUMP subcommand is:




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.

Use the DUMP subcommand to print part or all of a user's virtual storage on the printer. 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 0-7 and 8-15, and the floating-point registers 0-6 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 DUMP is issued without the first and second operands, 32 bytes of storage are dumped starting at the current origin. If DUMP is issued without the second operand, 32 bytes of storage are dumped starting at the location indicated by the first operand.

The first and second operands, if specified, must be either valid symbols or hexadecimal numbers. When a symbol is specified, the DEBUG symbol table is searched. If a match is found, the storage location corresponding to that symbol is used as 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 user's virtual 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 dumped to 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

The format of the GO subcommand is:

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.

Use the GO subcommand to exit from the debug environment and begin execution in the CMS environment. The old PSW for the interrupt that caused the debug environment to be entered is saved and later loaded to resume processing. Issuing the GO subcommand loads the old PSW.

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, it is possible to 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 user's virtual storage size. Further, it is the user's 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 interrupt, or program interrupt, 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 user's virtual 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 interrupt. 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

The format of the GPR subcommand is:

```
| GPR | reg1  [reg2]                                                    |
```

where:

reg1 is a decimal number (from 0-15 inclusive) indicating the first or only general register whose contents are to be typed.

reg2 is a decimal number (from 0-15 inclusive) 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.

Use the GPR subcommand to print the contents of one or more general registers at the terminal. 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 0-15 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 between 0 and 15 inclusive, 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

The format of the HX subcommand is:

The HX subcommand has no operands.

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.

### 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

The format of the ORIGIN subcommand is:

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 user's virtual storage.

The ORIGIN subcommand sets an origin or base address to be used in the debug environment. Use the ORIGIN subcommand to set the origin equal to the program load point, then 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.

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 user's virtual storage size.

Any origin set by an ORIGIN subcommand remains in effect until another 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

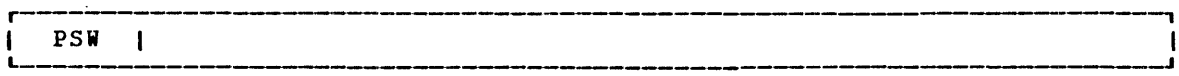
The ORIGIN subcommand requires one operand, and none was entered.

TOO MANY OPERANDS

The ORIGIN subcommand requires only one operand, and more than one was entered.

## PSW

The format of the PSW subcommand is:




The PSW subcommand has no operands.

Use the PSW subcommand to type the contents of the old PSW (program status word) for the interrupt that caused DEBUG to be entered. 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 interrupts 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 is used to control instruction sequencing and to hold and indicate the status of the system in relation to the program currently executing. Refer to Figure 54 in "Appendix A: System/370 Information" for a description of the PSW.

## 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

The format of the RETURN subcommand is:

The RETURN subcommand has no operands.

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.

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 interrupt, 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

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 interrupt 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). |

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 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.

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 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

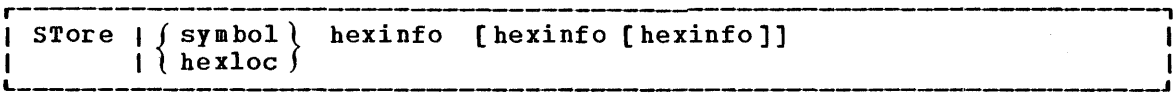
The minimum number of operands has not been entered.

TOO MANY OPERANDS

More than the required number of operands were specified.

## STORE

The format of the STORE subcommand is:




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.

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).

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 user's virtual 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 error 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

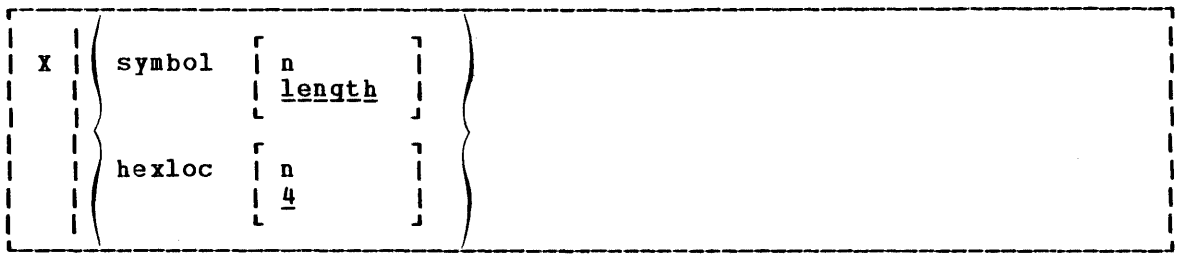
Less than two operands were specified.

TOO MANY OPERANDS

More than four operands were specified.

## X

The format of the X (examine) subcommand is:




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 to 56 inclusive, that specifies the number of bytes to be examined. If a hexadecimal location is specified without a second operand, four bytes are examined.

Note: 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.

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 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 used as 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 used as the location of the first byte to be examined. The derived address must not be greater than the user's virtual 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 between 1 and 56 inclusive. 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.

The SVCTRACE command traces 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 -, +, or *. 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 *ddddddddd*
       = h h h h h h h h *ddddddddd*
```

where h represents the contents of a general register in hexadecimal format and 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 registers 8-F on the second line. The hexadecimal contents of the registers are printed first, followed by the EBCDIC translation. The EBCDIC 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 and 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 and 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 and 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 and 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 and 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 13 summarizes the types of SVC trace output.

| Identification | Comments |
|---|---|
| { + / - / * } N/D | The SVC and the routine that 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 registers 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. |

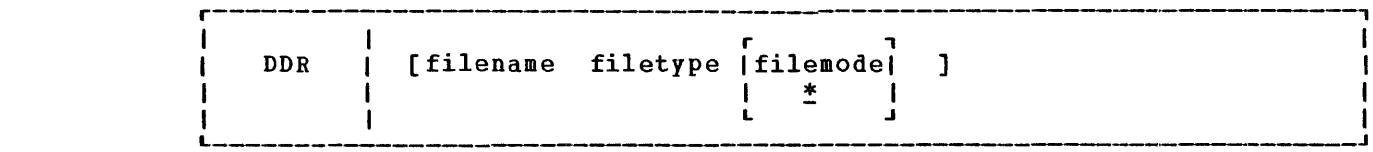
Figure 13. Summary of SVC Trace Output Lines

## 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:




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, 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.

See the "DDR Control Statements" discussion in the "Debugging with CP" section. The control statements for running standalone and under CMS are identical, except that CMS ignores the SYSPRINT control statement.

## 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 will not be 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 file identification of 'filename NUCMAP'. To determine the filename, issue the command

```
LISTF * NUCMAP S
```

To obtain a copy of the current nucleus load map, issue the command

```
PRINT filename NUCMAP
```

Figure 14 shows a sample CMS load map. Notice that the DEBUG work area (DBGSECT) and 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 replace cards that exist in the loaded files. The loadmap 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 several ways to get a load map.

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

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

```
| MODmap    | filename                                                    |
```

where:

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

```
FILE: LOAD     CMSMAP   C                    CONVERSATIONAL MONITOR SYSTEM

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 mm/dd/yy  15:37
ABBREV    AT 003000
USABRV    AT 0030D0
INVALID CARD...:READ  DMSINM   TEXT     C1 CMS191 mm/dd/yy  20:36
CMSTIMER  AT 003200
GETCLK    AT 003200
DMSINM    AT 003200
INVALID CARD...:READ  DMSTIO   TEXT     C1 CMS191 mm/dd/yy  10:33
TAPEIO    AT 003308
DMSTIO    AT 003308
```

Figure 14. Sample CMS Load Map

## READING CMS ABEND DUMPS

When CMS abnormally terminates, the terminal operator must enter 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 representation, eight words to the line, with EBCDIC translation at the right. The hexadecimal storage address corresponding to the first byte of each line is printed at the left.

When the Conversational Monitor System 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. Refer to "Part 3: Conversational Monitor System (CMS)" for functional information. The following discussion on reading CMS dumps refers to several CMS control blocks and fields in the control blocks. Refer to the VM/370: Data Areas and Control Blocks Logic for details on CMS control blocks. Figure 15 shows the relationships of CMS control blocks. You will also need a current CMS nucleus load map in order 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. Figure 16 lists all the CMS ABEND codes, identifies the module that caused the module to ABEND, and describes the action that should be taken whenever CMS abnormally terminates.

You may have to examine several fields in the Nucleus Constant Area (NUCON) of low storage.

1. Examine the program old PSW (PGMOPSW) at location X'28'. Using the PSW and current CMS load map, determine the failing address.
2. Examine the SVC old PSW (SVCOPSW) at location X'20'.
3. Examine the external old PSW (EXTOPSW) at location X'18'. If the virtual machine operator terminated CMS, this PSW points to the instruction executing when the termination request was recognized.
4. For a machine check, examine the machine check old PSW (MCKOPSW) at location X'30'. Refer to Figure 54 in "Appendix A: System/370 Information" for a description of the PSW.

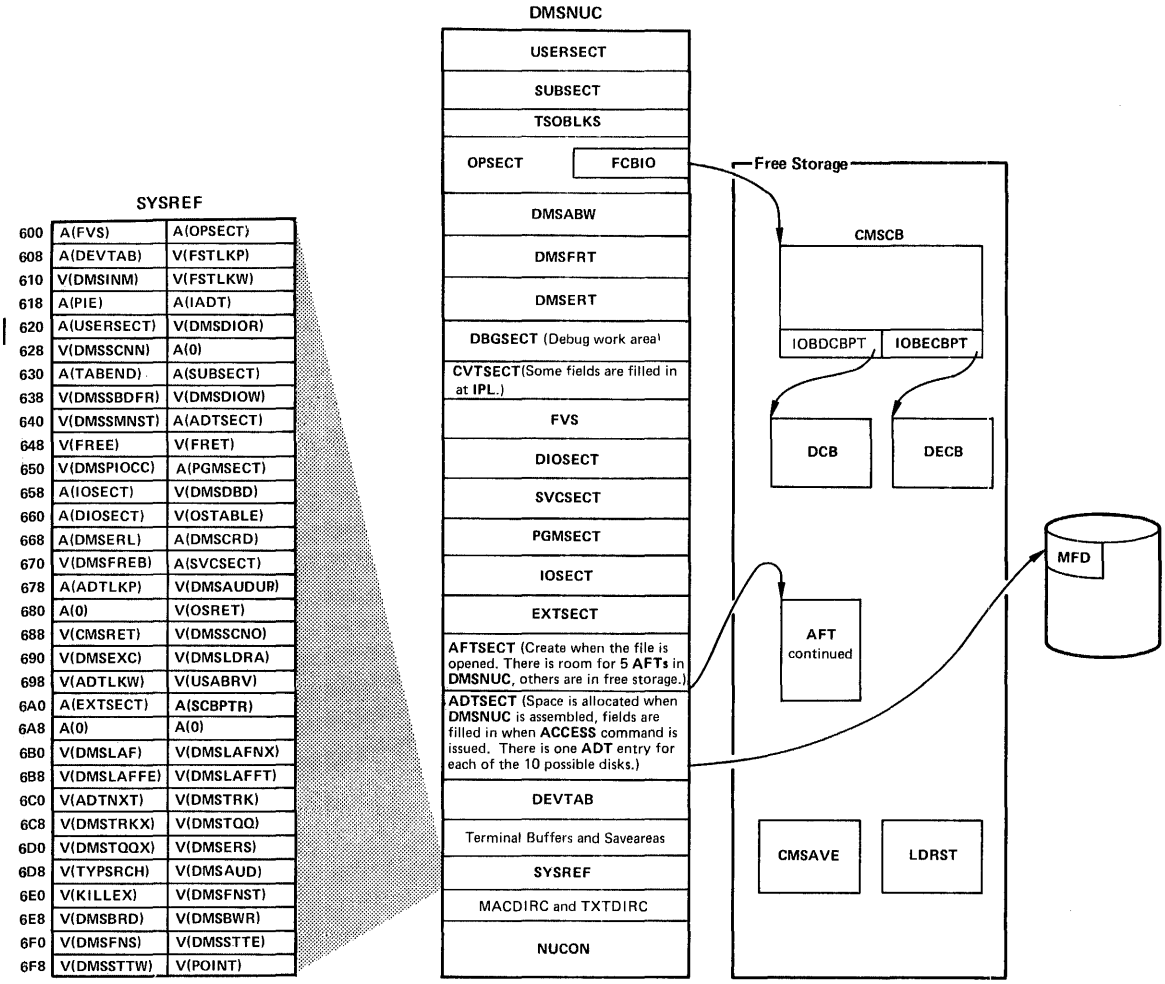



Figure 15. CMS Control Blocks

| 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 DOS/VS Messages Reference, 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:<br>x Type<br>0 IMPRECISE<br>1 OPERATION<br>2 PRIVILEGED OPERATION<br>3 EXECUTE<br>4 PROTECTION<br>5 ADDRESSING<br>6 SPECIFICATION<br>7 DECIMAL DATA<br>8 FIXED-POINT OVERFLOW<br>9 FIXED-POINT DIVIDE<br>A DECIMAL OVERFLOW<br>B DECIMAL DIVIDE<br>C EXPONENT OVERFLOW<br>D EXPONENT UNDERFLOW<br>E SIGNIFICANCE<br>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 16. 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 16. 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 16. CMS ABEND Codes (Part 3 of 3)

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

- Save Area for Low Storage

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

- Register Save Area

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

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

- Device

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

- Last Two Commands or Procedures Executed

| Field | Location | Contents |
|---|---|---|
| LASTCMND | X'2A0' | Last 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 |

- 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.

| Value of SFLAG | 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

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

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

The preceding information should help you to read a CMS dump. If it becomes necessary to trace file system control blocks, refer to Figure 32 in "Part 2. Conversational Monitor System" for more information. With a dump, the control block diagrams, and a CMS load map you should be able to find the cause of the ABEND.

# Part 2. Control Program (CP)

Part 2 contains the following information:

- Introduction to VM/370
- Program States
- Using CPU Resources
- Interruption Handling
- Functional Information
- Performance Guidelines
- Performance Observation and Analysis
- Accounting Information
- Generating Named Systems and Saving Systems
- VM/VS Handshaking
- OS/VS2 Release 2 Uniprocessor under VM/370
- DOS under VM/370
- Running VM/370 in a Virtual Machine
- Timers
- DIAGNOSE Instruction
- CP Conventions
- How to Add a Console Function
- How to Add a New Print or Forms Buffer Image

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

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

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

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

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

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

## INTRODUCTION TO THE VM/370 CONTROL PROGRAM

A virtual machine is created for a user when he logs on VM/370, on the basis of information stored in his VM/370 directory entry. The entry for each user identification includes a list of the virtual input/output devices associated with the particular virtual machine.

Additional information about the virtual machine is kept in the VM/370 directory entry. Included are the VM/370 command privilege class, accounting data, normal and maximum virtual storage sizes, dispatching priority, and optional virtual machine characteristics such as extended control mode.

The Control Program supervises the execution of virtual machines by (1) permitting only problem state execution except in its own routines, and (2) receiving control after all real computing system interrupts. CP intercepts each privileged instruction and simulates it if the current program status word of the issuing virtual machine indicates a virtual supervisor state; if the virtual machine is executing in virtual problem state, the attempt to execute the privileged instruction is reflected to the virtual machine as a program interrupt. All virtual machine interrupts (including those caused by attempting privileged instructions) are first handled by CP, and are reflected to the virtual machine if an analogous interrupt would have occurred on a real machine.

## VIRTUAL MACHINE TIME MANAGEMENT

The real CPU simulates multiple virtual CPUs. Virtual machines that are executing in a conversational manner 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 interrupts. The virtual machine is queued for subsequent CPU utilization 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 input/output operation has begun. The virtual machine cannot gain control of the real CPU if it is waiting for a page of storage, if it is waiting for an input/output operation to be translated and started, or if it is waiting for a CP command to finish execution.

A virtual machine can be assigned a priority of execution. Priority is a parameter affecting the execution of a particular virtual machine as compared with other virtual machines that have the same general execution characteristics. Priority is a parameter in the virtual machine's VM/370 directory entry. The system operator can reset the value with the Class A SET command.

## VIRTUAL MACHINE STORAGE MANAGEMENT

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

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

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

Inactive pages are kept on a direct access storage device. If an inactive page has been changed at some time during virtual machine

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

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

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

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

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

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

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

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

To use fetch protection, a virtual machine must execute the set storage key (SSK) instruction referring to the data areas to be protected, with the fetch protect bit set on in the key. VM/370 subsequently:

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

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

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

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

### Storage and CPU Utilization

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

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

The virtual=real option improves performance in the selected virtual machine since it removes the need for CP paging operations for the selected virtual machine. The virtual=real option is necessary whenever programs that contain dynamically modified channel programs (excepting those of OS ISAM and OS/VS TCAM Level 5) are to execute under control of CP. For additional information on running systems with dynamically modified channel programs, see "Dynamically Modified Channel Programs" in "Part 1. Debugging with VM/370."

## VIRTUAL MACHINE I/O MANAGEMENT

A real disk device can be shared among multiple virtual machines. Virtual device sharing is specified in the VM/370 directory entry or by a user command. If specified by the user, an appropriate password must be supplied before gaining access to the virtual device. A particular virtual machine may be assigned read-only or read/write access to a shared disk device. CP checks each virtual machine input/output

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 VM/370 directory entry of the virtual machine, or they may be attached to (or detached from) the virtual machine's configuration, dynamically, for the duration of the terminal session. Virtual devices may be dedicated, as when mapped to a fully equivalent real device; shared, as when mapped to a minidisk or when specified as a shared virtual device; or spooled by CP to intermediate direct access storage.

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

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

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

### Dedicated Channels

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

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

## SPOOLING FUNCTIONS

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

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

Spooling operations cease if the direct access storage space assigned to spooling is exhausted, and the virtual unit record devices appear in a not ready status. The system operator may make additional spooling space available by purging existing spool files or by assigning additional direct access storage space to the spooling function.

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

Files may also be spooled to remote stations via the Remote Spooling Communications Subsystem (RSCS), a component of VM/370. For a description of RSCS and the remote stations that it supports see "Part 5. Remote Spooling Communications Subsystem (RSCS)."

CP spooling includes many desirable options for the virtual machine user and the real machine operator. These options include printing multiple copies of a single spool file, backspacing any number of printer pages, and defining spooling classes for the scheduling of real output. Each output spool file has, associated with it, a 136 byte area known as the spool file tag. The information contained in this area and its syntax are determined by the originator and receiver of the file. For example, whenever an output spool file is destined for transmission to a remote location via the Remote Spooling Communications Subsystem, RSCS expects to find the destination identification in the file tag. Tag data is set, changed, and queried using the CP TAG command.

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

```
SPOOL CONSOLE START
```

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

```
SPOOL CONSOLE STOP
```

command.

## SPOOL FILE RECOVERY

If the system should suffer an abnormal termination, there are three degrees of recovery for the system spool files; warm start (WARM), checkpoint start (CKPT), and force start (FORCE). Warm start is

| automatically invoked if SET DUMP AUTO is in effect. Otherwise, the
| choice of recovery method is selected when the following message is
| issued;

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

| A cold (COLD) start does not recover any spool files.

### | Warm Start

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

### | Checkpoint Start

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

### | Force Start

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

## CP COMMANDS

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

CP commands are classified by privilege classes. The VM/370 directory entry for each user assigns one or more privilege classes. The classes are primary system operator, system resource operator,

system programmer, spooling operator, system analyst, service representative, and general user. Commands in the system analysts class may be used to inspect real storage locations, but may not be used to make modifications to real storage. Commands in the operator class provide real resource control capabilities. System operator commands include all commands related to virtual machine performance options, such as assigning a set of reserved page frames to a selected virtual machine. For descriptions of all the CP commands, see the CP Command Reference for General Users and the VM/370: Operator's Guide.

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

Only the Control Program may operate in the supervisor state on the real machine. All programs other than CP operate in the problem state on the real machine. All user interrupts, including those caused by attempted privileged operations, are handled by either the control program or the CPU (if the virtual machine assist feature is available). Only those interrupts that the user program would expect from a real machine are reflected to it. A problem program will execute on the virtual machine in a manner identical to its execution on a real System/370 CPU, as long as it does not violate the CP restrictions. See the "CP Restrictions" discussion in "Part 1: Debugging with CP" for a list of the restrictions.

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

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

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

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

## QUEUE 1

Virtual machines in Queue 1 (Q1) are considered conversational or interactive users, and enter this queue when an interrupt from a terminal is reflected to the virtual machine. There are two lists of users in Q1, executable and nonexecutable. The executable users are stacked in a first in, first out (FIFO) basis. When a nonexecutable user becomes executable, he is placed at the bottom of the executable list. If a virtual machine uses more than 50 milliseconds (ms) of CPU time without entering a virtual wait state, that user is placed at the bottom of the executable list.

Virtual machines are dropped from Q1 when they complete their time slice of CPU usage, and are placed in an "eligible list". Virtual machines entering CP command mode are also dropped from Q1. When the virtual machine becomes executable again (returns to execution mode) it is placed at the bottom of the executable list in Q1.

## QUEUE 2

Virtual machines in Queue 2 (Q2) are considered noninteractive users. Users are selected to enter Q2 from a list of eligible virtual machines (the "eligible list"). The list of eligible virtual machines is sorted on a FIFO basis within user priority (normally defined in the USER record in the VM/370 directory, but may be altered by the system operator).

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

If the calculated working set of the highest priority virtual machine in the eligible list is greater than the number of page frames available for allocation, CP continues through the eligible list in user priority order.

There are two lists of users in Q2, executable and nonexecutable. Executable virtual machines are sorted by "dispatching priority". This priority is calculated each time a user is dropped from a queue and is the ratio of CPU time used while in the queue to elapsed time in the queue. Infrequent CPU users are placed at the top of the list and are followed by more frequent CPU users. When a nonexecutable user becomes executable, he is placed in the executable list based on his dispatching priority.

When a virtual machine completes its time slice of CPU usage, it is dropped from Q2 and placed in the eligible list by user priority. When a user request in Q2 enters CP command mode, it is removed from Q2. When the request becomes executable (returns to virtual machine execution mode), it is placed in the eligible list based on user priority.

If a user's virtual machine is not in Q1 or Q2, it is because:

1. The virtual machine is on the "eligible list", waiting to be put on Q2, or

2. The virtual machine execution is suspended because the user is in CP mode executing CP commands.

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

In CP, interactive users (Q1), if any, are considered for dispatching before noninteractive users (Q2). This means that CMS users entering commands which do not involve disk or tape I/O operations should get fast responses from the VM/370 system even with a large number of active users.

An installation may choose to override the CP scheduling and dispatching scheme and force allocation of the CPU resource to a specified user, regardless of its priority or operating characteristics. The favored execution facility allows an installation to:

1. Specify that one particular virtual machine is to receive up to a specified percentage of CPU time.

2. Specify that any number of virtual machines are to remain in the queues at all times. Assignment of the favored execution option is discussed in the "Preferred Virtual Machines" section.

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

## PROGRAM INTERRUPT

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

## MACHINE CHECK INTERRUPT

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

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

- System termination (with automatic restart)
- System termination (CP disabled wait state)
- Selective virtual user termination
- Selective virtual machine reset
- Refreshing of damaged information with no effect on system configuration
- Refreshing of damaged information with the defective storage page removed from further systems 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 sufficiently initialized 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.

## SVC INTERRUPT

When an SVC interrupt occurs, the SVC interrupt routine is entered. If the machine is in problem mode, the type of interrupt (if it is other

than an SVC 76 or ADSTOP SVC) is reflected to the pseudo-supervisor (that is, the supervisor operating in the user's virtual machine). Control is transferred to the appropriate interrupt handler for ADSTOP SVCs and all SVC 76s.

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

## EXTERNAL INTERRUPT

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

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

## Functional Information

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

Figure 17 describes CP initialization process.

Figures 18 and 19 describe the real and virtual I/O control blocks used by CP in its I/O control.

Figures 20, 21, and 22 show how CP handles SVC, external, and program interrupts.

The CP paging function is described in Figure 23.

The CP spooling function (both virtual and real) is described in Figures 24 and 25.

Figure 26 shows how virtual tracing is performed.

Figure 27 shows the steps involved in translating a virtual address to a real address and gives an example of address translation.

The functional information contained in these diagrams is intended for system programmers and IBM Field Engineering program support representatives.

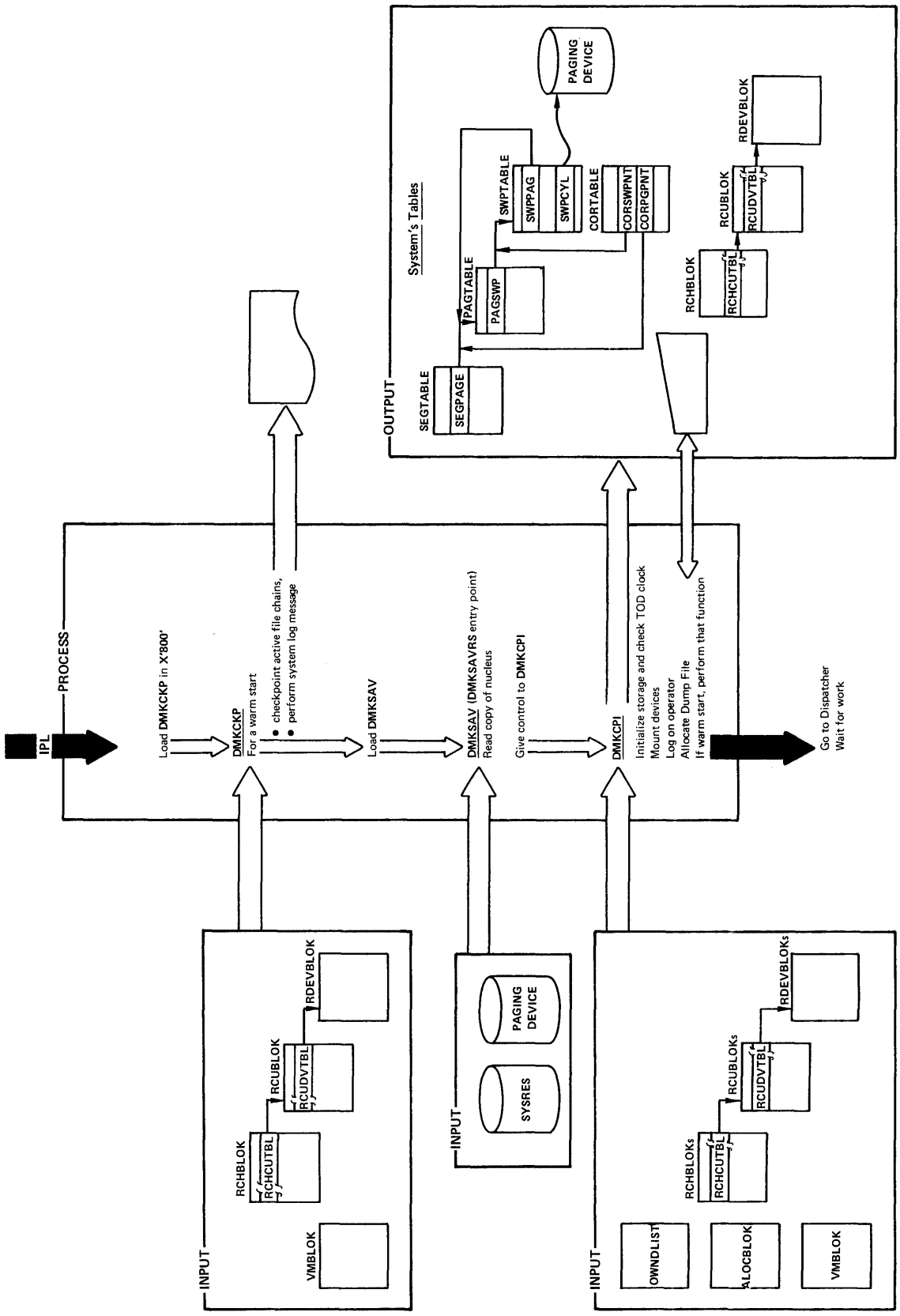



Figure 17. CP Initialization

**Figure 18. Real I/O Control Blocks**

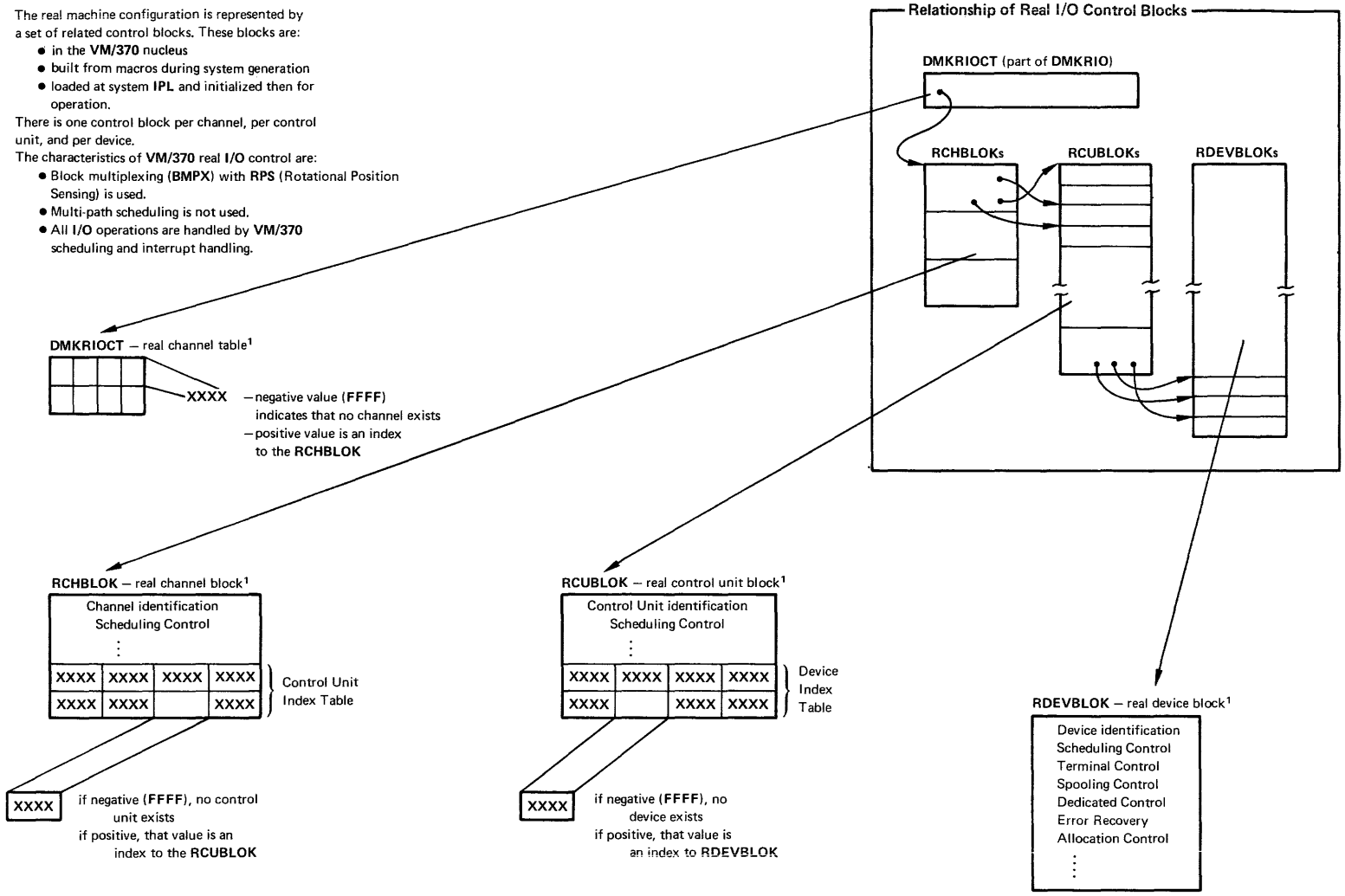
The real machine configuration is represented by a set of related control blocks. These blocks are:

- in the **VM/370** nucleus
- built from macros during system generation
- loaded at system **IPL** and initialized then for operation.

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

The characteristics of **VM/370** real **I/O** control are:

- Block multiplexing (**BMPX**) with **RPS** (Rotational Position Sensing) is used.
- Multi-path scheduling is not used.
- All **I/O** operations are handled by **VM/370** scheduling and interrupt handling.




[1] For a complete description of CP control blocks, see *IBM Virtual Machine Facility/370: Data Areas and Control Blocks*, Order No. SY20-0884.

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

Figure 19. Virtual I/O Control Blocks

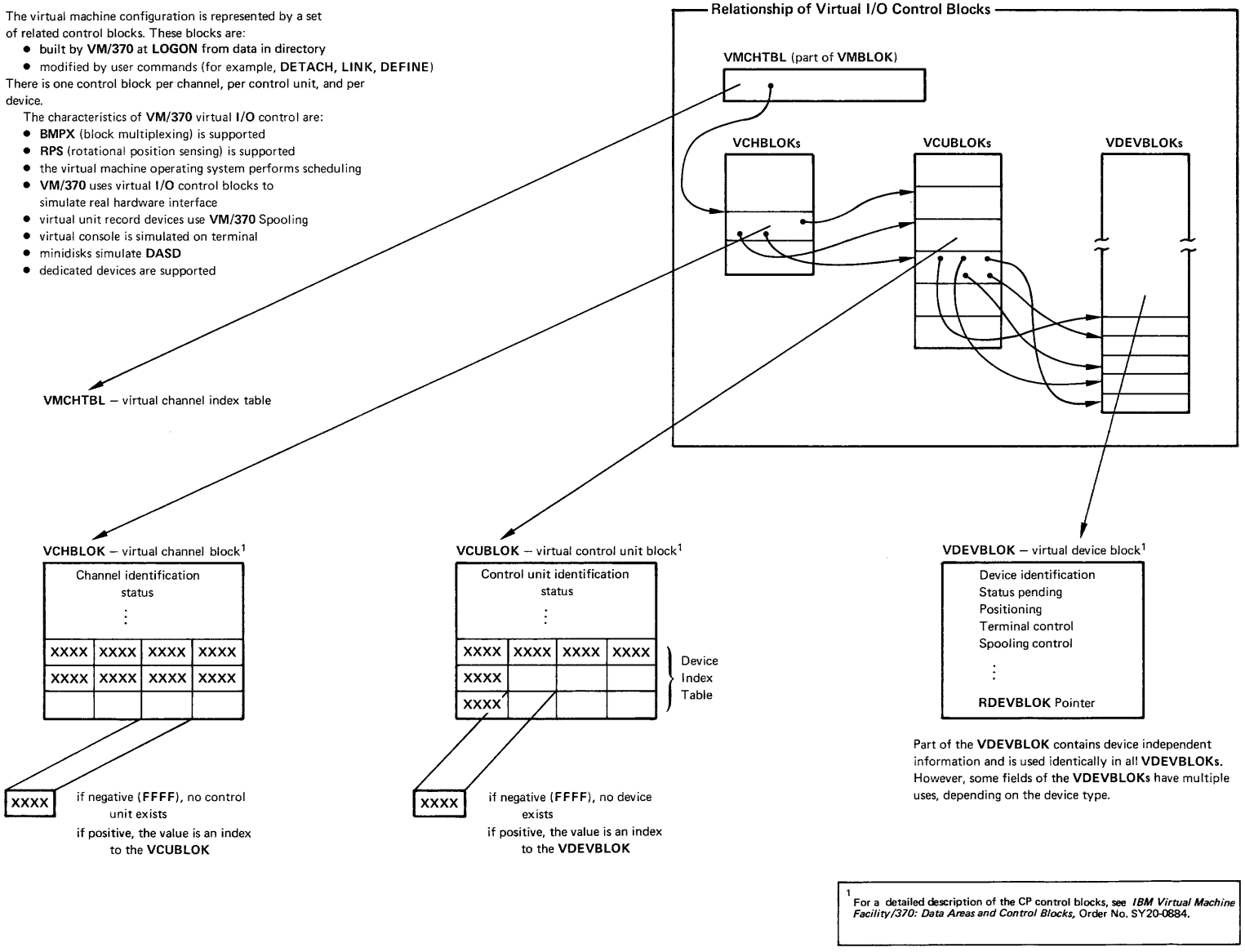
The virtual machine configuration is represented by a set of related control blocks. These blocks are:

- built by **VM/370** at **LOGON** from data in directory
- modified by user commands (for example, **DETACH, LINK, DEFINE**)

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

The characteristics of **VM/370** virtual **I/O** control are:

- **BMPX** (block multiplexing) is supported
- **RPS** (rotational position sensing) is supported
- the virtual machine operating system performs scheduling
- **VM/370** uses virtual **I/O** control blocks to simulate real hardware interface
- virtual unit record devices use **VM/370** Spooling
- virtual console is simulated on terminal
- minidisks simulate **DASD**
- dedicated devices are supported




[1] For a detailed description of the CP control blocks, see *IBM Virtual Machine Facility/370: Data Areas and Control Blocks*, Order No. SY20-0884.

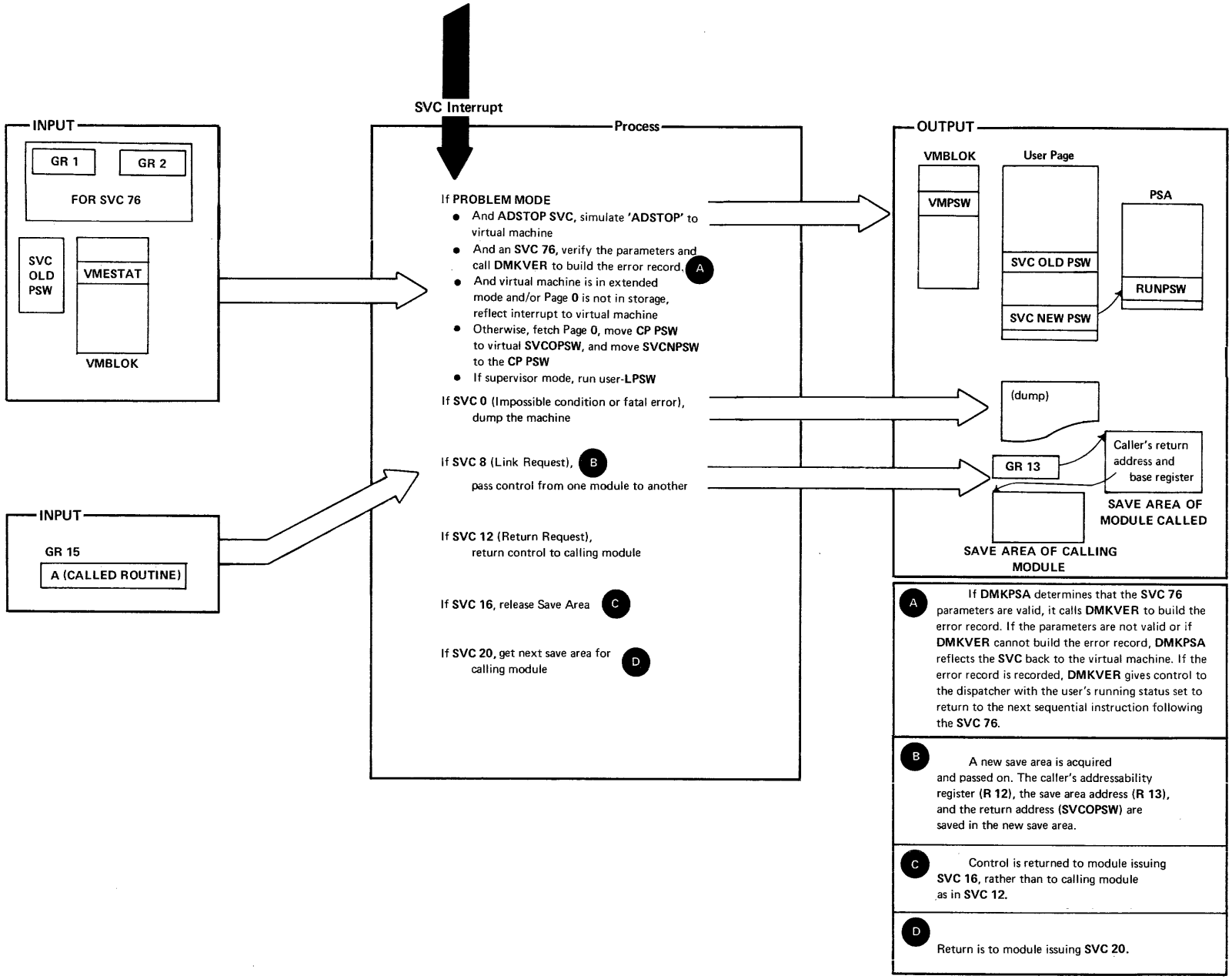



Figure 20. SVC Interrupt Handling

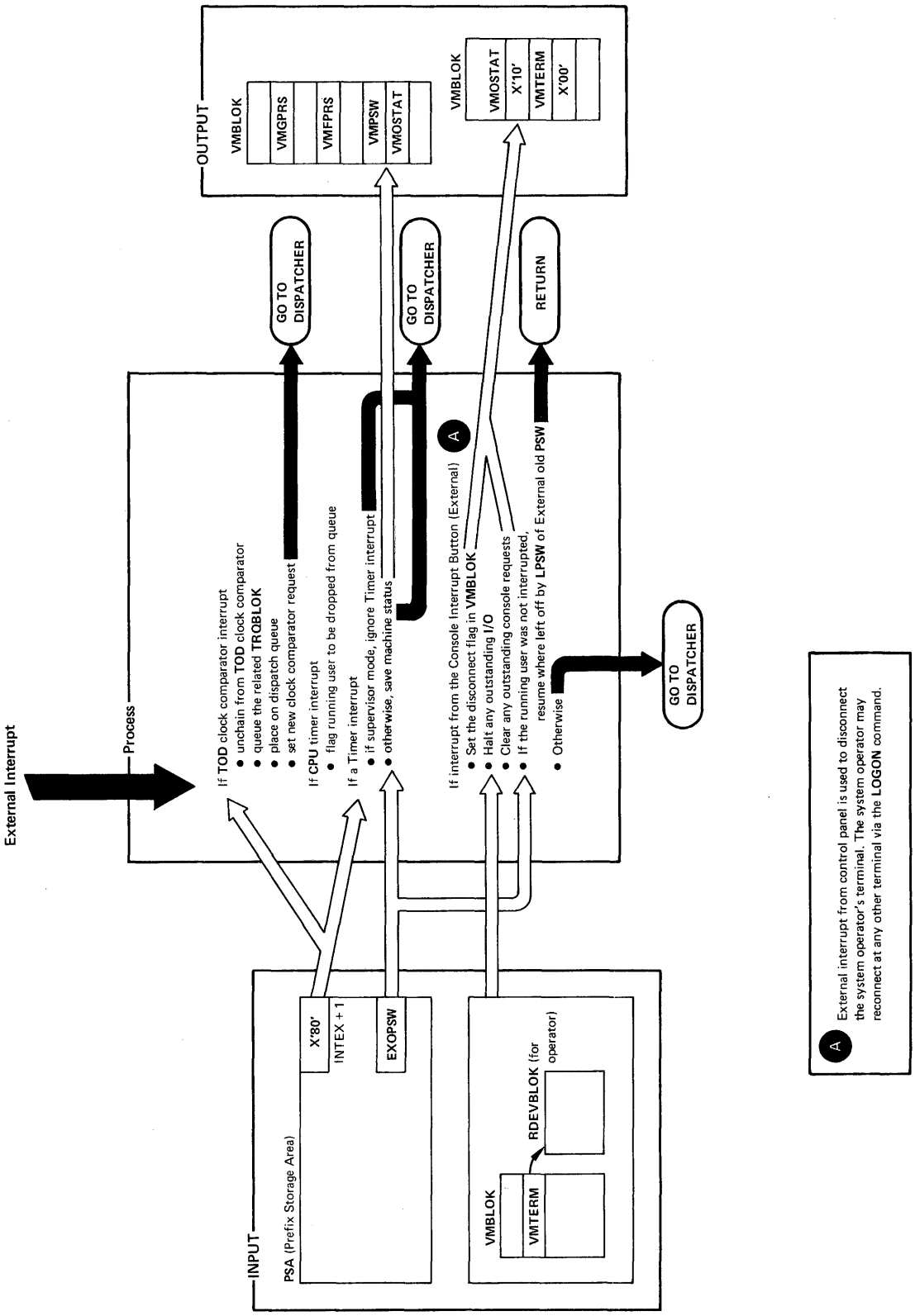



Figure 21. External Interrupt Handling

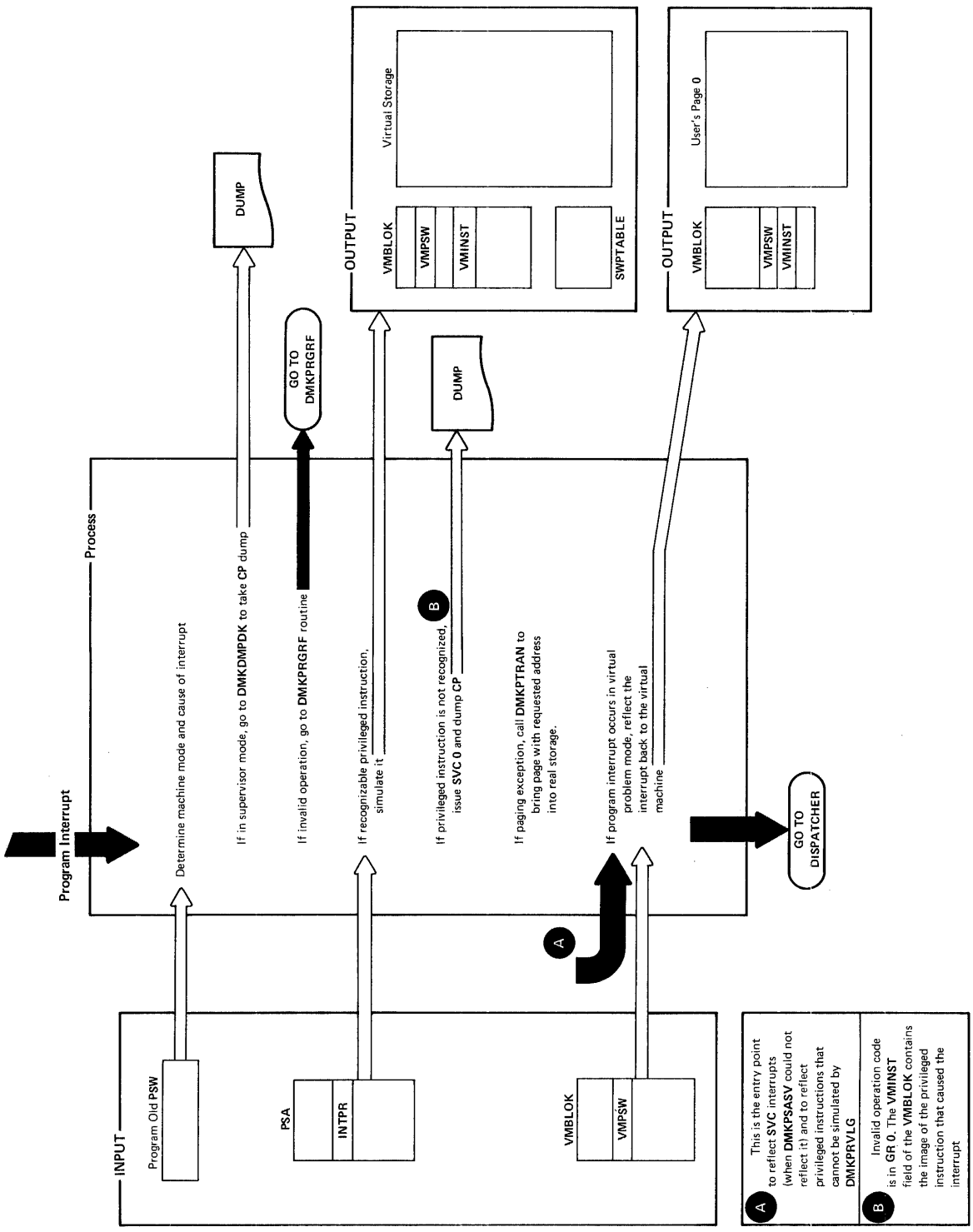



Figure 22. Program Interrupt Handling

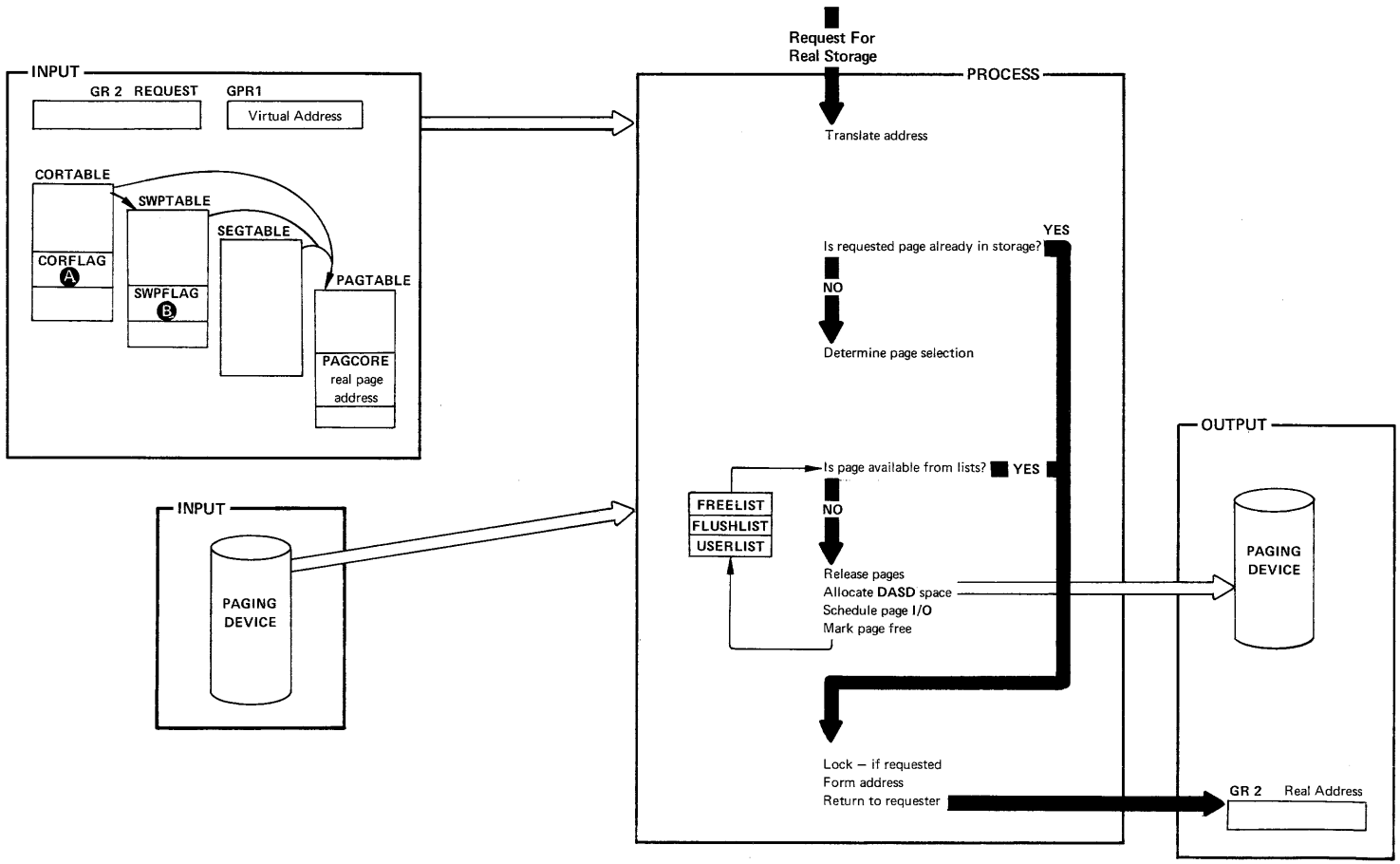



Figure 23. Paging

| A Bits defined for CORFLAG | | | | B Bits defined for SWPFLAG | | | |
|---|---|---|---|---|---|---|---|
| CORIOLCK | EQU | X'80' | Page locked for I/O | SWPTRANS | EQU | X'80' | Page in transit |
| CORCFLCK | EQU | X'40' | Page locked by console function | SWPRECMP | EQU | X'40' | Page permanently assigned |
| CORFLUSH | EQU | X'20' | Page is in flush list | SWPALLOC | EQU | X'20' | Page enqueued for allocation |
| CORFREE | EQU | X'10' | Page is in free list | SWPSHR | EQU | X'10' | Page shared |
| CORSHARE | EQU | X'08' | Page is shared | SWPREF1 | EQU | X'08' | 1st half page referenced |
| CORRSV | EQU | X'04' | Page is reserved | SWPCHG1 | EQU | X'04' | 1st half page changed |
| CORDISA | EQU | X'01' | Page disabled – not available | SWPREF2 | EQU | X'02' | 2nd half page referenced |
| | | | | SWPCHG2 | EQU | X'01' | 2nd half page changed |

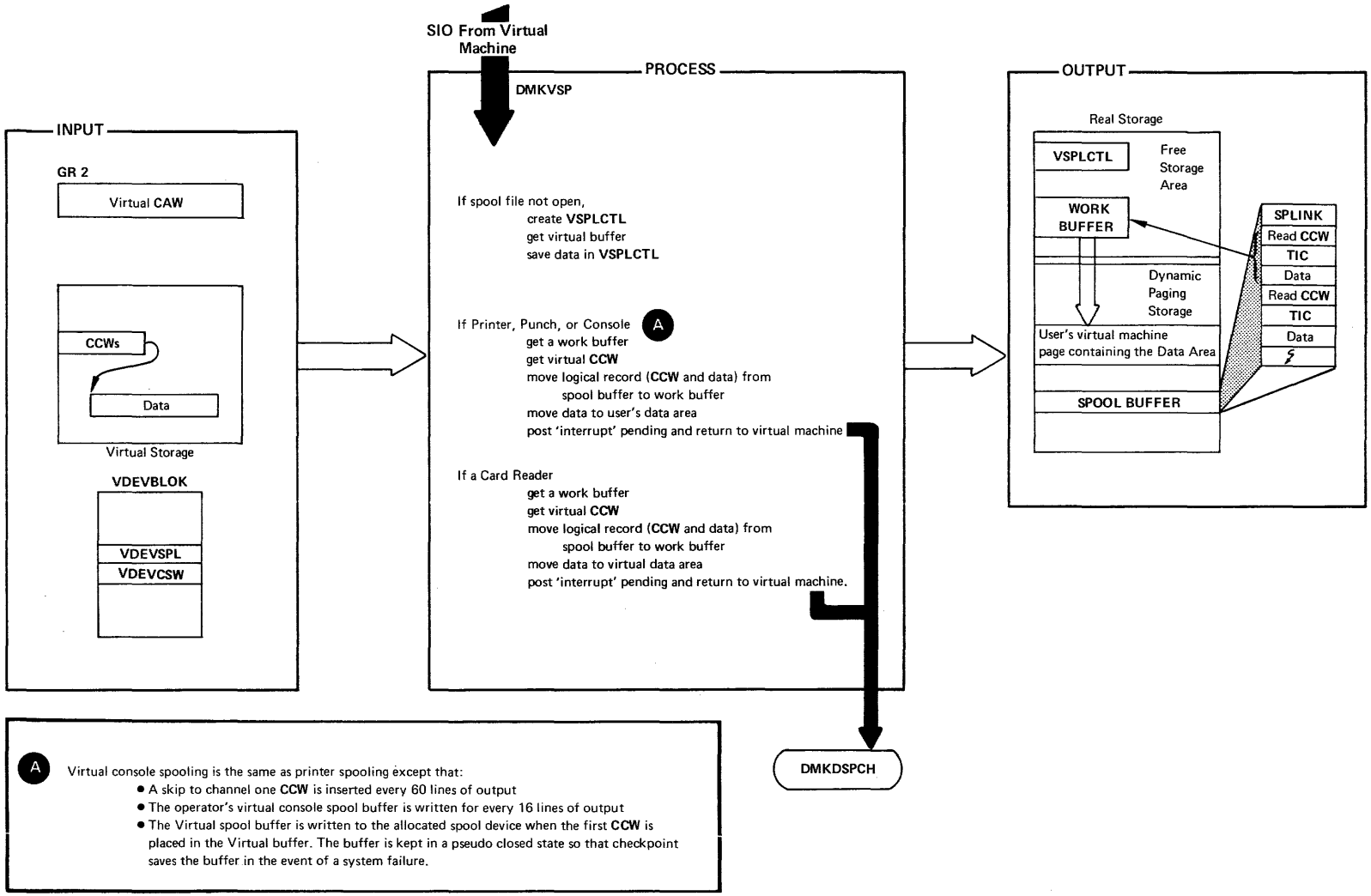



Figure 24. Virtual Spooling

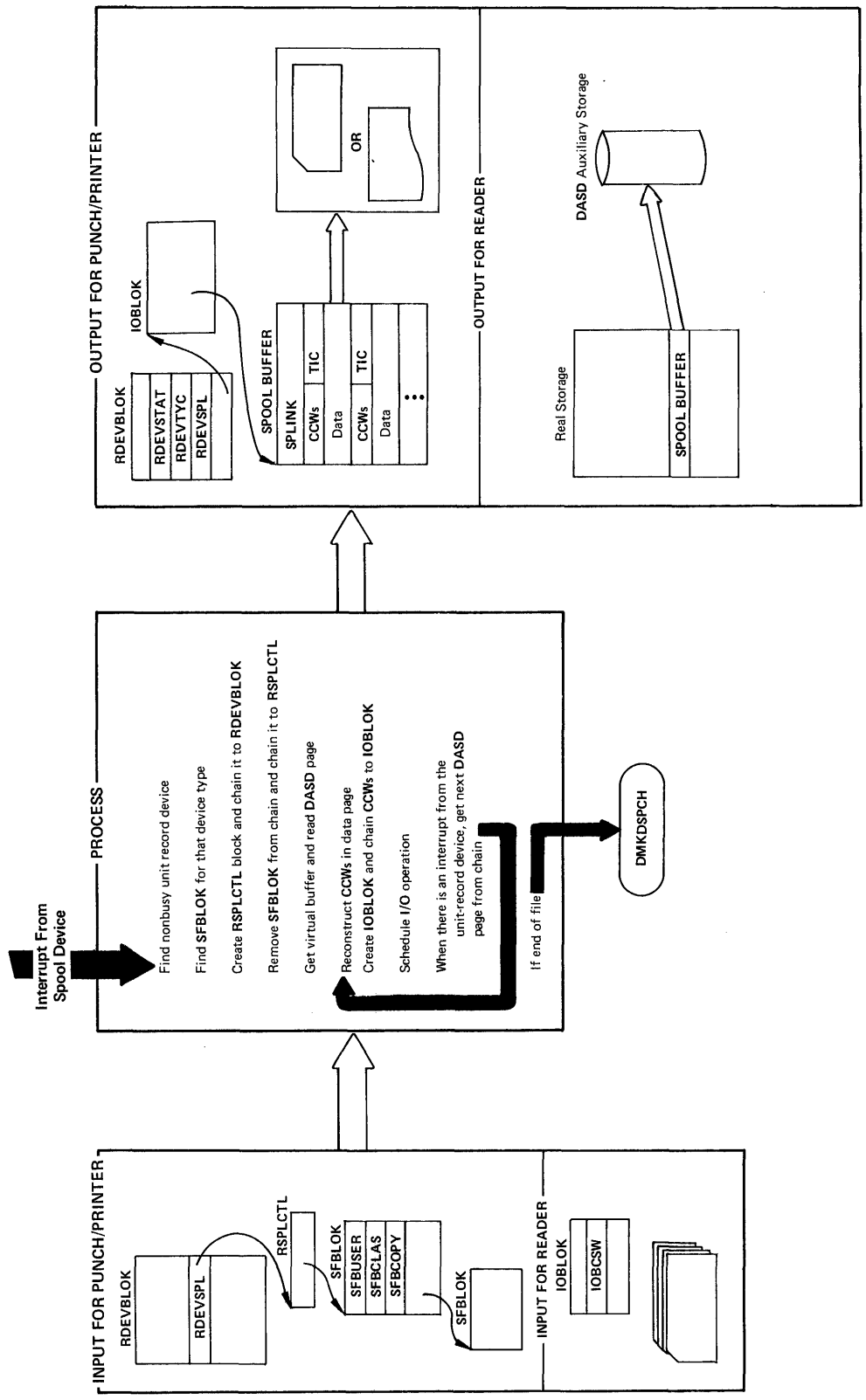
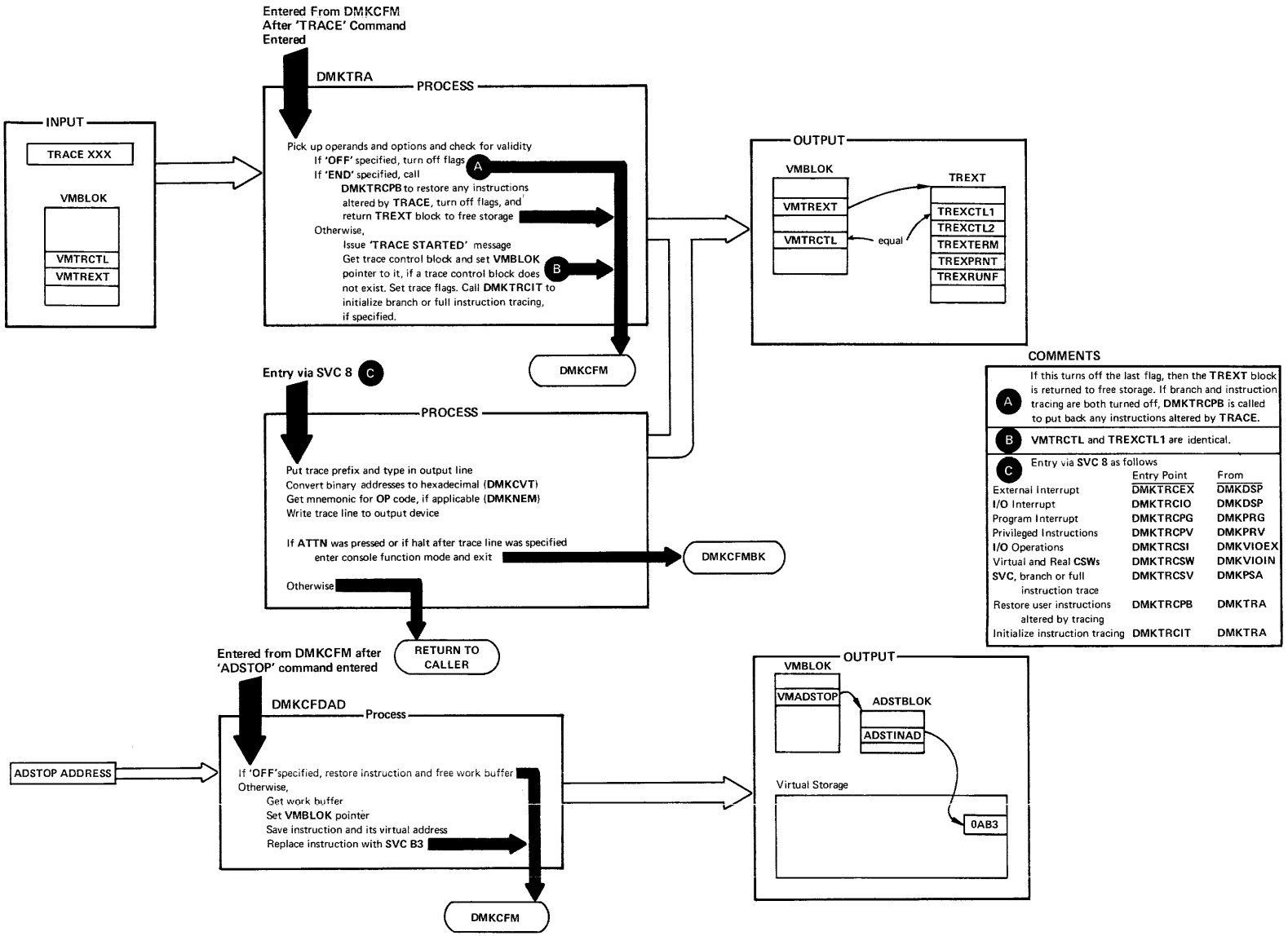



Figure 25. Real Spooling

Figure 26. Virtual Tracing



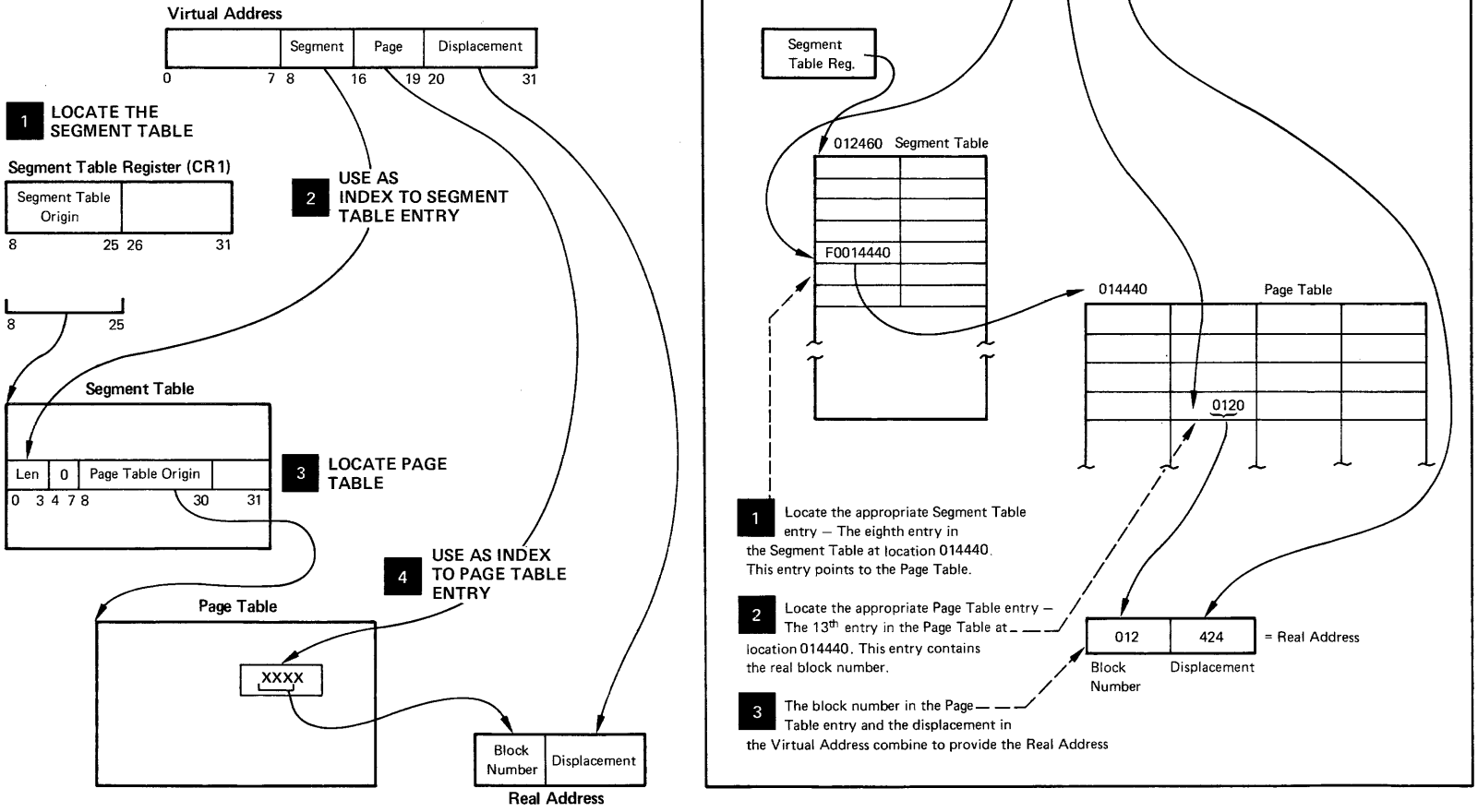



Figure 27. Virtual-to-Real Address Translation

## GENERAL INFORMATION

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

- The System/370 model used.
- The total number of virtual machines executing.
- The type of work being done by each virtual machine.
- The speed, capacity, and number of the paging devices.
- The amount of real storage available.
- The degree of channel and control unit contention, as well as arm contention, affecting the paging device.
- The type and number of VM/370 performance options in use by one or more virtual machines.

Performance of any virtual machine may be improved up to some limit by the choice of hardware, operating system, and VM/370 options. The topics discussed in this section address:

1. The performance options available in VM/370 to improve the performance of a particular virtual machine.
2. The system options and operational characteristics of operating systems running in virtual machines that will affect their execution in the virtual machine environment.

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

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

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

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

## VIRTUAL MACHINE I/O

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

1. Intercept the virtual machine SIO interrupt.
2. Allocate real storage space to hold the real CCW list to be created.
3. Translate the virtual device addresses referred to in the virtual CCWs to real addresses.
4. Page into real storage and lock for the duration of the I/O operation all virtual storage pages required to support the I/O operation.
5. Generate a new CCW sequence building a Channel Indirect Data Address list if the real storage locations cross page boundaries.
6. Schedule the I/O request.
7. Present the SIO condition code to the virtual machine.
8. Intercept, retranslate, and present the channel end and device end interrupts to the appropriate virtual machine, where they must then be processed by the virtual machine operating system.

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

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

- Use of large blocking factors (of up to 4096 bytes) for user data sets to reduce the total number of SIOs needed.
- Use of preallocated data sets.
- Use of virtual machine operating system options (such as chained scheduling in OS) that reduce the number of SIO instructions.
- Substitution of a faster resource (virtual storage) for I/O operations, by building small temporary data sets in virtual storage rather than using an I/O device.

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

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

1. The virtual=real option removes the need for CP to perform storage reference translation and paging before each I/O operation for a specific virtual machine.

2. The virtual machine assist feature reduces the real supervisor state time used by VM/370. It is available as a hardware feature on the System/370 Models 135, 145, and 158, and as an RPQ on the Model 168.

Assignment and use of these options is discussed in "Preferred Virtual Machines."

## PAGING CONSIDERATIONS

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

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

When an available page of virtual storage contains only reenterable code, paging activity can be reduced, since the page, although referred to, is never changed, and thus does not cause a write operation to the paging device. The first copy of that page is written on the paging device when that frame is needed for some other more active page. Only inactive pages that have changed must be paged out.

Virtual machines that reduce their paging activity by controlling their use of addressable space improve resource management for that virtual machine, the VM/370 system, and all other virtual machines. The total paging load that must be handled by CP is reduced, and more time is available for productive virtual machine use.

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

### Locked Pages Option

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

Since this facility reduces total real storage resources (real page frames) that are available to support other virtual machines, only frequently used pages should be locked into real storage. Since page zero (the first 4096 bytes) of a virtual machine storage is referred to and changed frequently (for example, whenever a virtual machine

interrupt occurs or when a CSW is stored), it should be the first page of a particular virtual machine that an installation considers locking. The virtual machine interrupt handler pages might also be considered good candidates for locking.

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

The normal CP paging mechanism selects unreferenced page frames in real storage for replacement by active pages. Unreferenced page frames are those whose contents have not been referred to during the last 50 milliseconds. Page frames belonging to inactive virtual machines will all eventually be selected and paged out if the real storage frames are needed to support active virtual machine pages.

When virtual machine activity is initiated on an infrequent or irregular basis, such as from a remote terminal in a teleprocessing inquiry system, some or all of its virtual storage may have been paged out before the time the virtual machine must begin processing. Some pages will then have to be paged in so that the virtual machine can respond to the teleprocessing request compared to running the same teleprocessing program on a real machine. This paging activity may cause an increase in the time required to respond to the request compared to running the teleprocessing program on a real machine. Further response time is variable, depending upon the number of paging operations that must occur.

Locking specific pages of the virtual machine's program into real storage may ease this problem, but it is not always easy or possible to identify which specific pages will always be required.

Once a page is locked, it remains locked until either the user logs off or the system operator (privilege class A) issues the UNLOCK command for that page. If the "locked pages" option is in effect and the user re-IPLs his system or loads another system, the locked pages will not be refreshed and a virtual system failure may occur. The SYSTEM CLEAR command will not clear the contents of any of that user's locked pages, even though it will clear virtual machine storage.

### Reserved Page Frames Option

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

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

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

### Virtual=Real Option

The VM/370 virtual=real option eliminates CP paging for the selected virtual machine. All pages of virtual machine storage, except page zero, are locked in the real storage locations they would use on a real computer. CP controls real page zero, but the remainder of the CP nucleus is relocated and placed beyond the virtual=real machine in real storage. This option is discussed in more detail in "Preferred Virtual Machines."

Since the entire address space required by the virtual machine is locked, these page frames are not available for use by other virtual machines except when the virtual=real machine is not logged on. This option often increases the paging activity for other virtual machine users, and in some cases for VM/370. (Paging activity on the system may increase substantially, since all other virtual machine storage requirements must be managed with fewer remaining real page frames.)

The virtual=real option may be desirable or mandatory in certain situations. The virtual=real option is desirable when running a virtual machine operating system (like DOS/VS or OS/VS) that performs paging of its own because the possibility of double paging is eliminated. The option must be used to allow programs that execute self-modifying channel programs or have a certain degree of hardware timing dependencies to run under VM/370.

## PREFERRED VIRTUAL MACHINES

VM/370 provides five functions that create a special virtual machine environment:

- Favored execution
- Priority
- Reserved page frames
- Virtual=real option
- Virtual machine assist

The first four functions are designed to improve the performance of a selected virtual machine; the last function improves the performance of VM/370. Although each of the first four functions could be applied to a different virtual machine, usually they are applied to only one if optimum performance is required for that one specific virtual machine. The fifth function can be applied to as many virtual machines as desired.

### Favored Execution

The favored execution options allow an installation to modify the normal scheduling algorithms and force the system to devote more of its CPU resources to a given virtual machine than would ordinarily be the case. The options provided are:

1. The basic favored execution option.

2. The favored execution percentage option.

The basic favored execution option means that the virtual machine so designated is not to be dropped from the active (in queue) subset by the scheduler, unless it becomes nonexecutable. When the virtual machine is executable, it is to be placed in the dispatchable list at its normal priority position. However, any active virtual machine represents either an explicit or implicit commitment of main storage. An explicit storage commitment can be specified by either the virtual=real option or the reserved page frames option. An implicit commitment exists if neither of these options is specified, and the scheduler recomputes the virtual machine's projected work-set at what it would normally have been at queue-drop time. Multiple virtual machines can have the basic favored execution option set. However, if their combined main storage requirements exceed the system's capacity, performance can suffer because of thrashing.

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, is guaranteed a specified minimum percentage of the total CPU time, if it can use it. The favored execution option can only be invoked by a system operator with command privilege class A. The format of the command is as follows:

```
                    ┌    ┐
SET FAVORED userid  |nn  |
                    |OFF |
                    └    ┘
```

where:

userid identifies the virtual machine to receive favored execution status.

nn is any value from 1 through 99 and specifies the percentage of the in-queue time slice that is guaranteed to this virtual machine.

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

The percentage option of the SET FAVORED command is administered as follows:

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

2. The favored virtual machine, when it is executable, is always placed at the top of the dispatchable list until it has obtained its guaranteed CPU time.

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

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

Whether or not a percentage is specified, a virtual machine with the favored execution option active is kept in the dispatching queues except under the following conditions:

- Entering CP console function mode
- Loading a disabled PSW
- Loading an enabled PSW with no active I/O in process
- Logging on or off

When the virtual machine becomes executable again, it is put back on the executable list in Q1. If dropped from Q1, the virtual machine is placed directly in Q2 and remains there even though it may exhaust its allotted amount of CPU usage. Virtual machines with this option are thus considered for dispatching more frequently than other virtual machines.

Note, however, that these options, can impact the response time of interactive users and that only one favored percentage user is allowed at any given time.

## Priority

The VM/370 operator can assign specific priority values to different virtual machines. In doing so, the virtual machine with a higher priority is considered for dispatching before a virtual machine with a lower priority. User priorities are set by the following class A command:

```
SET PRIORITY userid nn
```

where userid is the user's identification and nn is an integer value from 1 to 99. The value of nn affects the user's dispatching priority in relation to other users in the system. The priority value (nn) is one of the factors considered in VM/370's dispatching algorithm. Generally, the lower the value of nn, the more favorable the user's position in relation to other users in VM/370's dispatch queues.

## Reserved Page Frames

VM/370 uses chained lists of available and pageable pages. Pages for users are assigned from the available list, which is replenished from the pageable list.

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

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

```
SET RESERVE userid xxx
```

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

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

## Virtual=Real

For this option, the VM/370 nucleus must be reorganized to provide an area in real storage large enough to contain the entire virtual=real machine. In the virtual machine, each page from page 1 to the end is in its true real storage location; only its page zero is relocated. The virtual machine is still run in dynamic address translation mode, but since the virtual page address is the same as the real page address, no CCW translation is required. Since CCW translation is not performed, no check is made to ensure that I/O data transfer does not occur into page zero or any page beyond the end of the virtual=real machine's storage.

Systems that are generated with the virtual=real option use the system loader (DMKLD00E). See the VM/370: Planning and System Generation Guide for information about generating a virtual=real system.

Figure 28 is an example of a real storage layout with the virtual=real option.

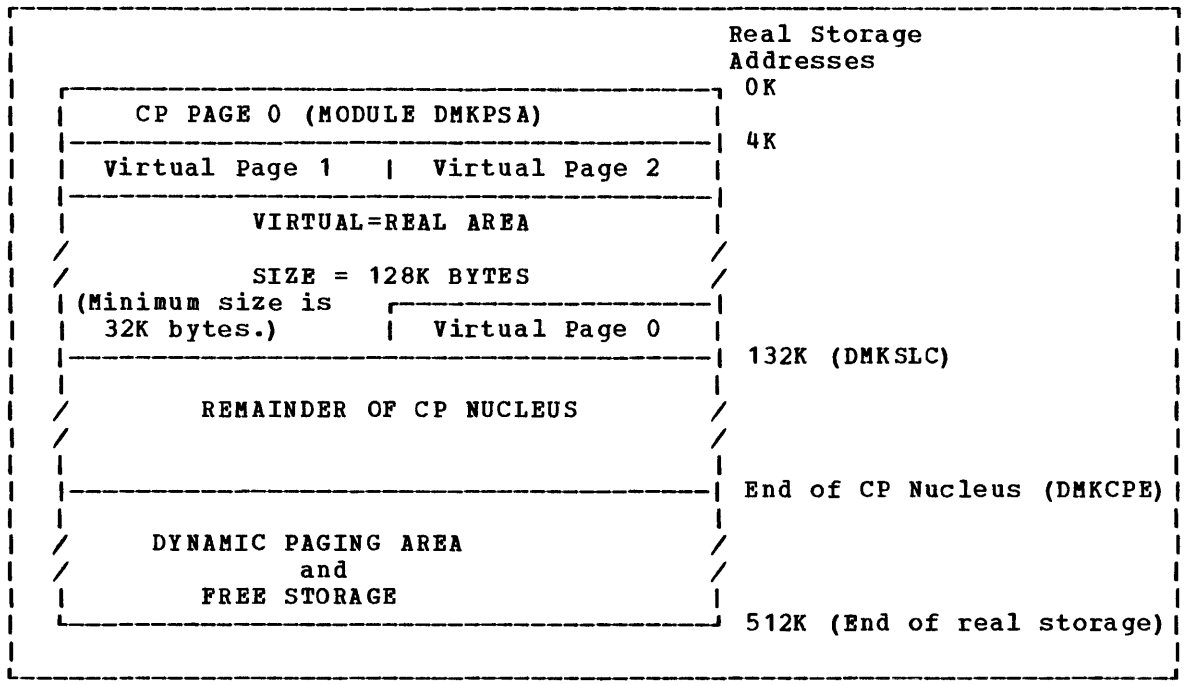



Figure 28. Storage in a Virtual=Real Machine

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

a. The area of contiguous storage built for the virtual=real machine must be large enough to contain the entire addressing space of the largest virtual=real machine. The virtual=real storage size that a VM/370 system allows is defined during system generation when the option is selected.

b. The storage reserved for the virtual=real machine can only be used by a virtual machine with that option specified in the VM/370 directory. It is not available to other users for paging space, nor for VM/370 usage until released from virtual=real status by a system operator via the CP UNLOCK command. Once released, VM/370 must be loaded again before the virtual=real option can become active again.

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

   - SIO tracing active
   - 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 nondedicated device
   - Pending device status

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

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

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

## Virtual Machine Assist Feature

The virtual machine assist feature is a combination of a CPU feature and VM/370 programming. It improves the performance of VM/370. Virtual storage operating systems which run in problem state under the control of VM/370 use many privileged instructions and SVCs that cause interrupts which VM/370 must handle. When the virtual machine assist feature is used, many of these interrupts are intercepted and handled by the CPU; and, consequently, VM/370 performance is improved.

The virtual machine assist feature is available with System/370 Models 135, 145, and 158. It intercepts and handles interruptions caused by SVCs (other than SVC 76), invalid page conditions, and several privileged instructions. An SVC 76 is never handled by the assist feature; it is always handled by CP. The processing of the following privileged instructions are handled by this feature:

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

Although the assist feature was designed to improve the performance of VM/370, virtual machines may see a performance improvement because more resources are available for virtual machine users.

USING THE VIRTUAL MACHINE ASSIST FEATURE: Whenever you IPL VM/370 on a CPU with the virtual machine assist feature, the feature is available for all VM/370 virtual machines. However, the system operator's SET command can make the feature unavailable to VM/370, and subsequently available again. The format of the system operator's SET command is:

```
SET ASSIST { ON  }
           { OFF }
```

If you do not know whether the virtual machine assist feature is available to VM/370, use the class A and E QUERY command. See the VM/370: Operator's Guide for a complete description of the Class A and E QUERY and SET commands.

If the virtual machine assist feature is available to VM/370 when you log on your virtual machine, it is also supported for your virtual machine. If your VM/370 directory entry has the SVCOFF option, the SVC handling portion of the assist feature is not available when you log on. The class G SET command can disable the assist feature (or only disable SVC handling). It can also enable the assist feature, or if the assist feature is available, enable the SVC handling. The format of the command is:

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

You can use the class G QUERY SET command line to find if you have full, partial, or none of the assist feature available. See the VM/370: CP Command Reference for General Users for a complete description of the Class G QUERY and SET commands.

RESTRICTED USE OF THE VIRTUAL MACHINE ASSIST FEATURE: Certain interrupts must be handled by VM/370. Consequently, the assist feature is not available under certain circumstances. VM/370 automatically turns off the assist feature in a virtual machine if it:

- Has an instruction address stop set.
- Traces SVC and program interrupts.

Since an address stop is recognized by an SVC interrupt, VM/370 must handle SVC interrupts while address stops are set. Whenever you issue the ADSTOP command, VM/370 automatically turns off the SVC handling portion of the assist feature for your virtual machine. The assist feature is turned on again after the instruction is encountered and the address stop removed. If you issue the QUERY SET command line while an address stop is in effect, the response will indicate that the SVC handling portion of the assist feature is off.

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

## THE VIRTUAL BLOCK MULTIPLEXER CHANNEL OPTION

Virtual machine SIO operations are simulated by CP in three ways: byte-multiplexer, selector, and block multiplexer channel mode.

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

Selector channel mode, the default mode, is the mode of operation for any channel that has an attached Channel-to-Channel Adapter (CTCA), regardless of the selected channel mode setting (the CTCA is treated as a shared control unit and therefore it must be connected to a selector channel). The user need not concern himself as to the location of the CTCA since CP interrogates the related channel linkage and marks the channel as being in selector mode. As in real selector channel operations, CP reflects a busy condition (condition code 2) to the virtual machine's operating system if the system attempts a second SIO to the same device, or another device on the same channel, before the first SIO is completed.

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

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

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

Two commands, INDICATE and MONITOR, provide a way to dynamically measure system performance.

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

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

## LOAD INDICATORS

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

The system analyst has a set of operands in the INDICATE command which enable him to understand the basic utilizations of and contentions for major system resources (possible bottleneck conditions) and to identify the userids and characteristics of the active users and the resources that they use.

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

## THE INDICATE COMMAND

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

The general user can display usage of and contention for the major system resources of CPU and storage. He can also display the total amount of resources he has used during his terminal session and the number of I/O requests. If he uses the INDICATE command before and after the execution of a program, he can determine the execution characteristics of that program in terms of resource usage.

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

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

The VM/370 scheduler maintains smoothed values of CPU usage and main storage contention. Specifically, every 30 seconds, the scheduler calculates the total wait time for the last interval and factors it into a smoothed wait value in the following way:

new smoothed wait value = (3 X old smoothed wait value + current interval wait)/4

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

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

new smoothed value = (15 X old smoothed value + last interval value)/16

Other operands of the command allow users to obtain other performance information that enables them to understand the reasons for the observed conditions.

## THE CLASS G INDICATE COMMAND

The format of the class G INDICATE command is:

where:

INDICATE LOAD
produces the following response, where n is a decimal number:

CPU nnn% Q1-nn Q2-nn STORAGE-nnn% RATIO-n.n

The CPU figure indicates the percentage of time that the system is running and is derived from the smoothed wait value maintained by the scheduler.

The contention for CPU is represented by smoothed values of the numbers of users in queue1 and queue2, maintained by the scheduler.

The next field, STORAGE, is a measure of the usage of real storage. It is a smoothed ratio of the sum of the estimated working sets of the users in queue1 and queue2, to the number of pageable pages in the system, expressed as a percentage.

Due to the algorithm used by the scheduler in determining entry to the active queues, the value of STORAGE can exceed 100%.

The scheduler contention ratio, RATIO, is a smoothed measure of the contention for real storage, and is defined as:

$$RATIO = \frac{E+M}{M}$$

where

M is the number of users in queue1 and queue2

E is the number of users waiting to be allocated real storage by the scheduler and therefore temporarily resident in the scheduler's eligible lists.

Thus, RATIO is the ratio of active users to users being serviced, and is 1.0 for optimum response. Optimum response occurs when enough real storage is available to accommodate all active users, assuming the CPU can process their commands. If E and M are both zero, the value of RATIO is set to 1.0.

Given the value of RATIO and M, (Q1+Q2) the number of users in the eligible list can be computed as:

$$E = M\ (RATIO-1)$$

INDICATE USER

allows a user to determine the resources used and occupied by his virtual machine, and the I/O events that have taken place.

The following two line response is returned:

```
PAGES: RES-nnnn WS-nnnn READS=nnnnnn WRITES=nnnnnn DISK-nnnn DRUM-nnnn
VTIME=nnn:nn TTIME=nnn:nn SIO=nnnnnn RDR-nnnnnn PRT-nnnnnn PCH-nnnnnn
```

The first line of the response displays the data from the user's VMBLOK that is relevant to his virtual machine's paging activity and resource occupancy.

RES is the current number of the user's virtual storage pages resident in real storage at the time the command is issued.

WS is the most recent system estimate of the user's working set size.

READS is the total number of page reads for this user since he logged on or since the last ACNT command was issued for his virtual machine.

WRITES is the total number of page writes for this user since he logged on or since the last ACNT command was issued for his virtual machine.

DISK is the current number of virtual pages allocated on the system paging disk for this user.

DRUM is the current number of virtual pages allocated on the system paging drum for this user.

The second line of the response gives the user his CPU usage and accumulated I/O activity counts since logon or since the last ACNT command was issued for his virtual machine.

VTIME is the total virtual CPU time for the user.

TTIME is the total virtual CPU and simulation time for the user.

SIO is the total number of non-spooled I/O requests issued by the user.

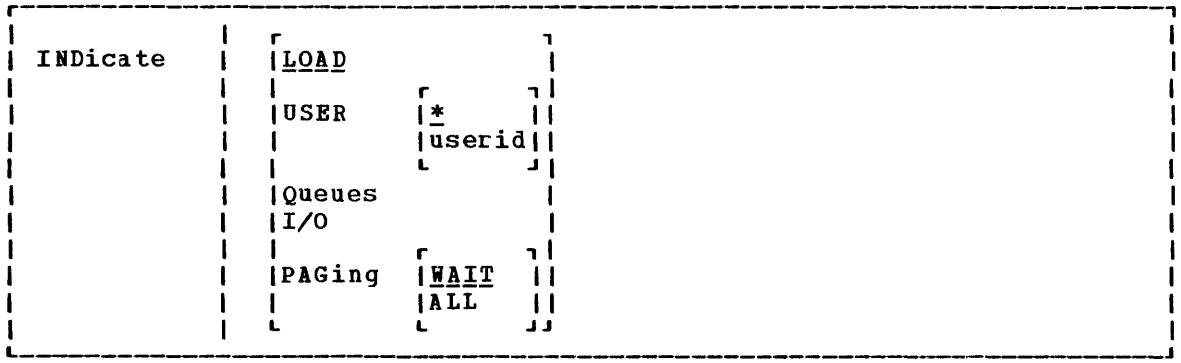
RDR is the total number of virtual cards read.

PRT is the total number of virtual lines printed.

PCH is the total number of virtual cards punched.

## THE CLASS E INDICATE COMMAND

The format of the class E INDICATE command is:




where:

INDICATE LOAD
provides the same output as the INDICATE LOAD option described under "The Class G Indicate Command."

INDICATE USER *
reflects activity of the system analyst's own virtual machine. The output of this option is the same as that of the INDICATE USER * option described under "The Class G Indicate Command."

INDICATE USER userid
allows the system analyst to determine the activity of other virtual machines in terms of the resources used and occupied and events that have taken place. Users with class E authority can access data from the VMBLOK of any user currently logged onto the system in their attempts to understand an overload or poor performance situation.

The output of this option is the same as that of the INDICATE USER * option described under "The Class G Indicate Command".

INDICATE QUEUES

displays the active users, the queues they are in, the storage they are occupying, and the status they are in. The display indicates those users currently dominating main storage. Users waiting in eligible lists are included in the response because they are contending for main storage and it is only by chance that they were not occupying main storage at the time of the command.

The response to the INDICATE QUEUES command is as follows:

userid1 aa bb sss/ttt userid2 ... (up to 3 userids per line)

where:

useridn is the user identification.

aa is the eligible list or queue that the user occupies.

bb is one of the following status indicators:

- RU the user is currently running.
- PG the user is not running because CP is attempting to bring in a page from a paging device.
- IO the user is in I/O wait because access to the device is not available at the moment.
- EX the user is waiting for the completion of an instruction simulation.
- PS the user is in an enabled wait state for high speed I/O devices.
- -- waiting to be redispatched.

Note: In cases where a virtual machine may be in more than one of the above states, only one state is displayed. The state displayed is the first one encountered in the order of priority indicated above.

sss is a hexadecimal number indicating the number of pages resident in real storage

ttt is a hexadecimal number indicating the working set size

INDICATE I/O

provides information about conditions leading to possible I/O contention within the system. The response gives the userids of all the users in I/O wait state at that instant in time, and the address of the real device to which the most recent virtual SIO was mapped. Because the response indicates only an instantaneous sample, use the command several times before assuming a condition to be persistent. If it is persistent, run the SEEKS option of the MONITOR command to conduct a thorough investigation of the suggested condition.

The response to the INDICATE I/O option is as follows:

userid1 cuu userid2 cuu ... (up to 5 userids per line)

where:

useridn is the user identification.

cuu indicates the real device address.

In the case where a virtual machine may have issued multiple SIOs, the response indicates the real device address corresponding to the most recent one issued.

INDICATE PAGING WAIT
is provided for installations that have 2305s as primary paging devices and other direct access devices as secondary paging device. A full primary device and subsequent allocation of paging space on the slower device may be responsible for degradation in system performance. Use the INDICATE PAGING WAIT option when the INDICATE QUEUES option shows that a significant proportion of the users in queue1 and queue2 are persistently in page wait. The response to the command gives the userids of those users currently in page wait and the numbers of page frames allocated on drum and on disk.

The response to the INDICATE PAGING WAIT option is as follows:

userid1 nnn:mmm userid2 nnn:mmm ... (up to 4 userids per line)

where:

useridn is the user identification.

nnn is the hexadecimal number of pages allocated on drum for these users.

mmm is the hexadecimal number of pages allocated on disk for these users.

Note: Consider, for example, the following response:

usera 010:054 userb 127:000

If the two users were to execute programs of similar characteristics, then usera would be expected to experience more pagewait than userb. Also, if the level of multiprogramming were to be low during the execution of usera's program, then more system page wait would occur than during the execution of userb's program.

If users appear to have most of their pages allocated on disk, it would be useful to know which users are occupying most of the primary paging device space, and whether or not they are still active. (That is, a virtual machine that is running a large operating system may have been allocated large amounts of primary paging device space at IPL time but then may have become inactive. Consequently, the machine is occupying a critical resource that could be put to better use.

INDICATE PAGING ALL
displays the page residency data of all users of the system (including the system nucleus and pageable routines). The response is identical to that of the INDICATE PAGING WAIT option.

Other Responses:

The following response is issued for the INDICATE QUEUES option when appropriate:

NO USERS IN QUEUE

The following response is issued for the INDICATE I/O option when appropriate:

```
NO USERS IN I/O WAIT
```

The following response is issued for the INDICATE PAGING WAIT option when appropriate:

```
NO USERS IN PAGEWAIT
```

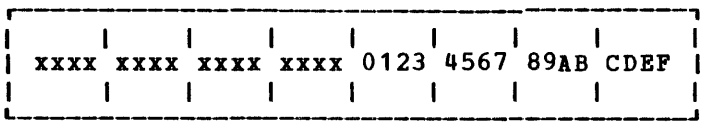
## THE MONITOR COMMAND

VM Monitor collects data in two ways:

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

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

MONITOR CALL instructions with appropriate classes and codes are presently embedded in strategic places throughout the main body of VM/370 code (CP). When a MONITOR CALL instruction executes, a program interruption occurs if the particular class of MONITOR CALL is enabled. The classes of MONITOR CALL that are enabled are determined by the mask in control register 8. For the format and function of the MONITOR CALL instruction, refer to the System/370 Principles of Operation Manual, Order No. GA22-7000. The format of control register 8 is as follows:




where:

x — indicates unassigned bits.

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

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

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

Monitor output consists of event data and sampled data. Event data is obtained via MONITOR CALL instructions placed within the VM/370 code. Sampled data is collected following timer interruptions. All data is recorded on the output tape as though it were obtained through a MONITOR CALL instruction. This simplifies the identification of the tape records.

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

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

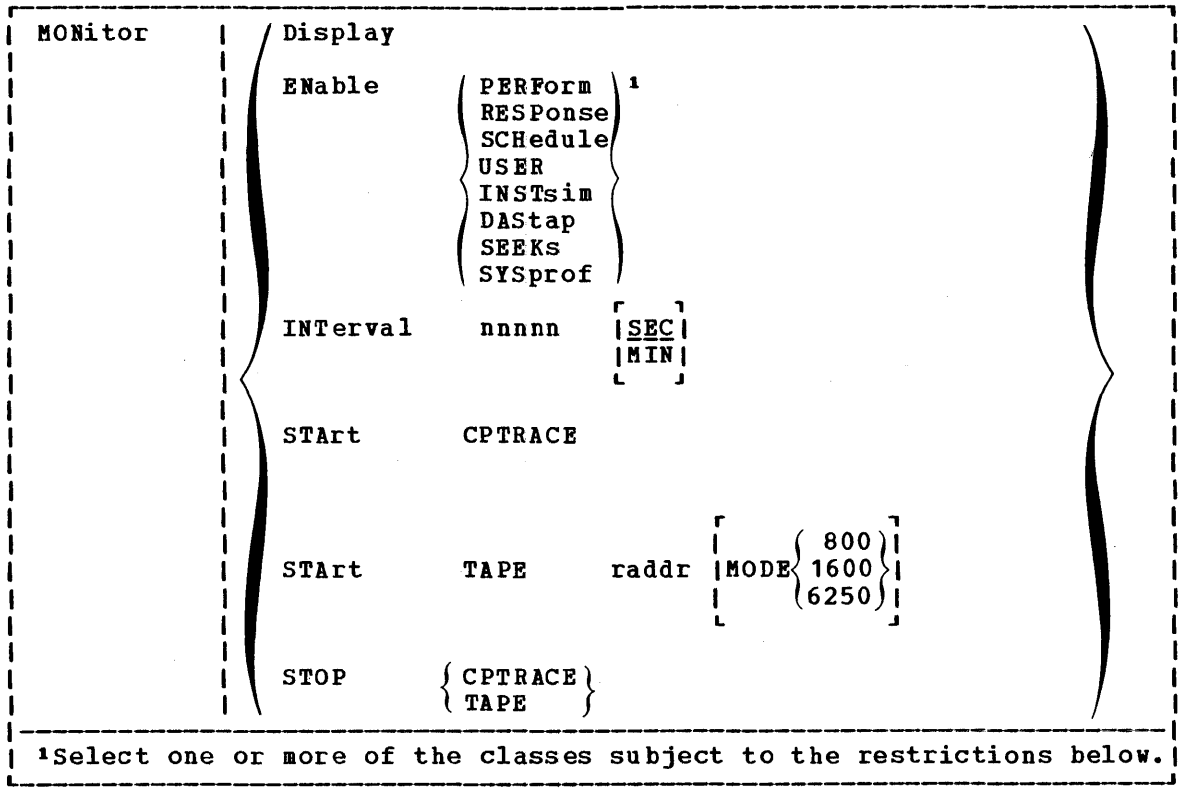
Another function, separate from the VM Monitor, is also handled by the MONITOR command. The MONITOR command can stop and start CP internal trace table data collection, which is not initiated by MONITOR CALLs.

Note: The VM Monitor tape record format and the contents of the tape record are shown in Appendix B in this manual.

The MONITOR command:

- Stops and starts CP internal trace table data collection.
- Displays the status of the internal trace table and each implemented class of VM Monitor data collection. Enables one or more classes of MONITOR CALL.
- Starts and stops data collection recording by VM Monitor onto tape.
- Specifies what interval is to be used for timer driven data collection.

The format of the class A and E MONITOR command is:




where:

DISPLAY
displays the status of the internal trace table and the implemented classes. A separate line of output for the internal trace table and each class of MONITOR CALL indicates the class and its status (enabled or disabled).

```
ENABLE ( PERForm  )
       ( RESPonse )
       ( SCHedule )
       ( USER     )
       ( INSTsim  )
       ( DAStap   )
       ( SEEKs    )
       ( SYSprof  )
```

enables the specified classes of MONITOR CALL. Each successful completion of this command creates a new mask for control register 8. The function of each class is described in the section "Implemented Classes."

The effect of the MONITOR ENABLE command depends on whether data collection is active or inactive when the command is issued. If data collection is active (MONITOR START TAPE has been issued), the new mask is moved directly into control register 8, replacing the previous mask, and the new mask takes effect immediately. Collection then continues with the classes just entered. If data collection is not active at the time the command is issued, then the mask is saved until the MONITOR START TAPE command is issued.

MONITOR ENABLE Restrictions:

Restrictions exist on issuing the MONITOR ENABLE command while the VM Monitor is collecting and recording data on tape.

Every MONITOR ENABLE command yields a new mask. Thus, for example, if PERFORM and USER classes are currently being collected, and you enter MONITOR ENABLE INSTSIM, then PERFORM and USER classes are stopped and INSTSIM is started.

The DASTAP operand in the MONITOR ENABLE command must be specified prior to the MONITOR START TAPE command. DASTAP may be disabled at any time by respecifying the MONITOR ENABLE command with DASTAP absent from the class list.

The SYSPROF class cannot be activated unless both the DASTAP and SCHEDULE classes are also active.

If data collection is in progress when you issue a MONITOR ENABLE command and an error occurs in the command line during processing, no change is made to the monitoring status. Unrecognizable keywords, conflicting or missing operands generate appropriately different error messages.

Due to the security exposure which potentially exists with collecting terminal input and output data, the RESPONSE class of data collection does not occur unless the system programmer sets the TRACE (1) bit in the LOCAL COPY to 1 and reassembles the CP module DMKMCC. If this is not done, the RESPONSE class is considered an invalid operand of the MONITOR ENABLE command.

```
INTERVAL  nnnnn [SEC]
                [MIN]
```

specifies the time interval to be used for the three timer driven data collection classes: PERFORM, USER, and DASTAP. The value specified by nnnnn is the number of seconds or minutes between data collections. If no interval is specified on the MONITOR INTERVAL command, an error message occurs. If

you give an interval but enter neither SEC nor MIN, the default is SEC. The maximum allowable interval is 9 hours (540 minutes or 32,400 seconds). The minimum is 30 seconds.

If the MONITOR INTERVAL command is not issued, the default interval is 60 seconds. The MONITOR INTERVAL command can be issued at any time; however, if data collection is already in progress, the new interval does not take effect until the current interval has elapsed.

The MONITOR interval is reset to the default of 60 seconds whenever any of the following occurs:

- The user issues MONITOR STOP TAPE
- The system stops the MONITOR because of an unrecoverable I/O error
- The end of tape is reached

START CPTRACE

starts the tracing of events that occur on the real machine. The events are recorded in 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.

```
                  ┌      ┌ 800 ┐ ┐
START TAPE raddr  │MODE{ 1600 } │
                  │      └ 6250 ┘ │
                  └              ┘
```

starts the data collection by VM Monitor on to a tape. Specify "raddr" as the real hexadecimal address of the tape drive that you want to use. It activates data collection for those classes of MONITOR CALL previously specified in a MONITOR ENABLE command. The mask that was saved by the MONITOR ENABLE command is moved into control register 8. The data is collected in two buffer pages in real storage. These pages are separate from the internal trace table pages. As each data page is filled, it is written onto the tape.

When the VM Monitor is started, CP issues a REWIND command followed by a Set Mode command for the reset value of tape density.

The user can request a different mode setting by specifying the MODE option in the MONITOR START TAPE command. Mode values of 800, 1600, or 6250 BPI may be specified.

Note: If a user specifies density mode that the tape cannot handle, the control unit may not return an error conditon; in this case, the mode setting is ignored and the default control unit setting is used.

STOP CPTRACE

terminates the tracing of events occurring on the real machine. 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 MONITOR START CPTRACE.

STOP TAPE

stops data collection by VM Monitor on to tape. A zero mask is immediately stored in control register 8, thus disabling

MONITOR CALL interruptions. The last partially filled page is written out, two tape marks are written, and the tape is rewound and unloaded. The two buffer pages, which were obtained at the time the MONITOR START TAPE command was issued, are released.

Note: The CPTRACE and TAPE operands of the MONITOR command are completely separate functions. Commands affecting the status of one function have no effect on the other.

Responses:

The following response occurs if you issue the MONITOR DISPLAY command:

| CLS | KEYWORD | STATUS |
|---|---|---|
| 0 | PERFORM | |
| 1 | RESPONSE | (ENABLED |
| 2 | SCHEDULE | |
| 4 | USER | or |
| 5 | INSTSIM | |
| 6 | DASTAP | DISABLED) |
| 7 | SEEKS | |
| 8 | SYSPROF | |
| -- | CPTRACE | |

The following response occurs for MONITOR commands, except MONITOR DISPLAY, that successfully execute:

```
COMMAND COMPLETE
```

## IMPLEMENTED CLASSES

The following MONITOR CALL classes correlate with the corresponding classes in control register 8. Refer to the System/370 Principles of Operation, Order No. GA22-7000 for details of the MC instruction and the bits in control register 8.

| Monitor Class | Keyword | Data Collection Function |
|---|---|---|
| Zero | PERFORM | Samples system resource usage data by accessing system counters of interest to system performance analysts. |
| One | RESPONSE | Collects data on terminal I/O. Simplifies analyses of command usage, user and system response times. It can relate user activity to system performance. This class is invalid and no data can be collected for it unless the system programmer changes the LOCAL COPY file and reassembles DMKMCC. |
| Two | SCHEDULE | Collects data about scheduler queue manipulation. Monitors flow of work through the system, and indicates the resource allocation strategies of the scheduler. |
| Three | ------- | Reserved. |

| | | |
|---|---|---|
| Four | USER | Periodically scans the chain of VMBLOKs in the system, and extracts user resource utilization and status data. |
| Five | INSTSIM | Records every virtual machine privileged instruction handled by the control program (CP). Because simulation of privileged instructions is a major source of overhead, this data may lead to methods of improving performance.<br><br>If the VMA feature is active, the number of privileged instructions that are handled by the control program is reduced for those virtual machines that are running with the feature activated. |
| Six | DASTAP | Periodically samples device I/O activity counts (SIOs), for tape and DASD devices only.<br><br>It is possible that the number of DASD and tape devices defined in DMKRIO exceeds 291 (the maximum number of MONITOR DASTAP records that fit in a MONITOR buffer). The following algorithm determines which devices are monitored:<br><br>1. If the total number of DASD and tape devices that are online is less than or equal to 291, all online DASD and tape devices are monitored.<br><br>2. If the online DASD devices total less than or equal to 291, all online DASD devices are monitored.<br><br>3. Otherwise, the first 291 online DASD device are monitored. |
| Seven | SEEKS | Collects data for every I/O request to DASD devices. Reveals channel, control unit, or device contention and arm movement interference problems.<br><br>No data is collected for TIO or HIO operations. For SIO operations, data is collected when the request for the I/O operation is initially handled and again when the request is satisfied.<br><br>This means that a single SIO request could result in two Monitor Calls. For example, if the request gets queued because the device is already busy, then a Monitor Call would be issued as the request is queued. Later, when the device becomes free and is restarted, a second Monitor Call is issued.<br><br>In general, the data collected is the same except that in the first case there will be non-zero counts associated with queued requests. |

| | | |
|---|---|---|
| | | If the request for I/O is satisfied when it is initially handled, without being queued, only one Monitor Call results. In both this case and the second of the two data collections mentioned above, the count of I/O requests queued for the device is zero. |
| Eight | SYSPROF | Collects data complimentary to the DASTAP and SCHEDULE classes in order to provide a more detailed "profile" of system performance through a closer examination of DASD utilization. |

## VM MONITOR RESPONSE TO UNUSUAL TAPE CONDITIONS

### Suspension

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

### Unrecoverable Tape Error

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

### End-of-Tape Condition

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

## VM MONITOR CONSIDERATIONS

### System Generation

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

### Initial Program Load

MONITOR START CPTRACE is active after real system IPL (manual or automatic). The VM Monitor tape data collection is off after IPL.

### System Shutdown

System shutdown implies a MONITOR STOP TAPE command. Normal command processing for the MONITOR STOP TAPE function is performed by the system.

### System Failure

If the VM/370 system fails and data collection is active, an attempt is made to write two tape marks, rewind and unload the tape. If the tape drive fails to rewind and unload, be sure to write a tape mark before rewinding and unloading the tape. VM Monitor data collection is terminated by the system failure.

### I/O Devices

A supported tape drive must be dedicated to the system for the duration of the monitoring. For accounting purposes, all I/O is charged to the system.

## VM MONITOR DATA VOLUME AND OVERHEAD

Use of the VM Monitor requires that three pages be locked in storage for the entire time the VM Monitor is active; this reduces by three the number of page frames available for paging. This significantly affects the performance of the rest of the system when there is a limited number of page frames available for paging.

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

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

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

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

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

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

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

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

## Monitoring Recommendations

For daily monitoring, to generate a data tape suitable for analyzing utilization and performance trends, use the PERFORM class only. If performance bottlenecks are suspected, run the RESPONSE and SCHEDULE classes to relate user activity to system scheduling decisions in terms of demands on system resources. If particular users are suspected of dominating the system, or if I/O activity is suspected of being concentrated on particular devices, then run the USER and DASTAP classes. If the DASTAP class does not give enough information to resolve possible I/O contention questions, activate the SEEKS class for a short period of time, for instance, ten minutes.

The SYSPROF class can be enabled along with SCHEDULE and DASTAP to give an additional breakdown of I/O device activity as it relates to queue manipulation by the scheduler. The INSTSIM class can be enabled to determine which users are incurring high amounts of system overhead due to instruction simulation.

## LOAD ENVIRONMENTS OF VM/370

Two distinct uses of VM/370 can be readily identified, and consequently some differences in criteria for acceptable performance may occur. The system may be required to time share multiple batch-type virtual machines with interactive machines performing minor support roles; or, the system may be primarily required to provide good interactive time-sharing services in the foreground, with a batch background absorbing spare resources of real storage and CPU.

## Performance for Time-Shared Multi-Batch Virtual Machines

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

After determining this, you can perform external observations of turn-around time on benchmarks and specify a point beyond which the addition of more users would be unacceptable. However, when that point is reached, more sophisticated internal measurement is required to determine the most scarce resource and how the bottleneck can be relieved by additional hardware.

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

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

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

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

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

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

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

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

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

A number of other conditions are more obvious and straightforward. You need to measure response and determine at what point it becomes unacceptable and why. Studies of time-sharing systems have shown that a user's rate of working is closely correlated with the system response. When the system responds quickly, the user is alert, ready for the next interaction, and thought processes are uninterrupted. When the system response is poor, the user becomes sluggish.

For interactive environments, a need exists to analyze command usage. Average execution time of the truly interactive commands can provide data for validation of the queue1 execution time.

The accounting data gathered by VM/370 can help in analysis of overall system operation. Also, accounting data can be used to bill VM/370 users for time and other system resources they use.

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

If the amount of free storage (available page frames) is relatively small and the card punch is not periodically assigned to punch CP's accounting cards, it is possible for CP's accounting routine to progressively use a significant percentage of the available page frames and cause a page thrashing condition to occur in VM/370. This occurs because the accounting routine creates and maintains accounting records in real storage, and does not free that storage space until the accounting records are punched on the real system card punch.

To eliminate this problem, it is recommended that one punch pocket be permanently dedicated to this accounting function, or if that is not feasible, to punch all the accumulated records every 1 to 2 hours.

Accounting cards are punched and selected to pocket 2 of any class C card punch when a user logs off of the system, detaches a dedicated device or T-disk or issues a DIAGNOSE code X'4C' instruction. (If the real punch is a 2540, the accounting cards are put in pocket 3.) These records should be kept for system accounting purposes.

## ACCOUNTING RECORD FOR VIRTUAL MACHINE RESOURCE USAGE

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

| Column | Contents |
|---|---|
| 1- 8 | Userid |
| 9-16 | Account number |
| 17-28 | Date and Time of Accounting (mmddyyhhmmss) |
| 29-32 | Number of seconds connected to VM/370 System |
| 33-36 | Milliseconds of CPU time used, including time for VM/370 supervisor functions |
| 37-40 | Milliseconds of virtual CPU time used |
| 41-44 | Number of page reads |
| 45-48 | Number of page writes |
| 49-52 | Number of virtual machine SIO instructions for nonspooled I/O |
| 53-56 | Number of spool cards to virtual punch |
| 57-60 | Number of spool lines to virtual printer (this includes one line for each carriage control command) |
| 61-64 | Number of spool cards from virtual reader |

| Column | Contents |
|---|---|
| 65-78 | Reserved |
| 79-80 | Accounting card identification code[1] (01) |

## ACCOUNTING RECORDS FOR DEDICATED DEVICES AND TEMPORARY DISK SPACE

Accounting cards are punched and selected to pocket 2 of any class C card punch when a previously dedicated device and temporary disk space is released by a user via DETACH, LOGOFF, or releasing from DIAL (dedicated device only). A dedicated device is any device assigned to a virtual machine for that machine's exclusive use. These include devices dedicated by the ATTACH command, those being assigned at logon by directory entries, or by a user establishing a connection (via DIAL) with a system that has virtual 2702 or 2703 lines. The information on the accounting card is as follows (columns 1-28 contain character data; all other data is in hexadecimal form, except as noted):

| Column | Contents |
|---|---|
| 1- 8 | Userid |
| 9-16 | Account number |
| 17-28 | Date and Time of Accounting (mmddyyhhmmss) |
| 29-32 | Number of seconds connected to VM/370 system |
| 33 | Device class |
| 34 | Device type |
| 35 | Model (if any) |
| 36 | Feature (if any) |
| 37-38 | Number of cylinders of temporary disk space used (if any). This information appears only in a code 03 accounting card. |
| 39-78 | Unused |
| 79-80 | Accounting card identification code. (02, 03) |

The device class, device type, model, and feature codes in columns 33-36 are shown in Figure 12.

## ACCOUNTING RECORDS CREATED BY THE USER

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

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

---

[1] The accounting card identification code is one of the following:

| | |
|---|---|
| C0 | User formatted accounting card (formatted by user and punched via DIAGNOSE code 4C) |
| 01 | User virtual machine accounting card |
| 02 | User dedicated device accounting card |
| 03 | User temporary disk space accounting card. |

The information on the accounting card is as follows:

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

A complete description of the DIAGNOSE code X'4C' instruction can be found under "DIAGNOSE Instruction in a Virtual Machine" in this section.

## OPERATIONAL NOTES

If a punch is started for two classes with NOSEP specified, accounting cards are not uniquely separated from data decks. If started with NOSEP specified, the operator is prompted when a user has a deck to be punched. The operator can thus remove any accounting cards before starting the punch. After data is through punching, accounting cards may be punched.

If the amount of free storage (available page frames) is relatively small and the card punch is not periodically assigned to punch out CP's accounting cards, it is possible for CP's accounting routine to progressively use up a significant percentage of the available page frames and cause a page thrashing condition to occur in VM/370. This is because the accounting routine creates and updates accounting records in real storage, and does not free that storage space until the accounting records are punched out on the real system card punch. This situation is further aggravated when the accounting option for a batch virtual machine is in effect, due to the increased number of accounting records generated.

To eliminate this problem, it is recommended that one punch pocket be permanently dedicated to this accounting function, or, if that is not feasible, to punch out all the accumulated accounting records every 1 to 2 hours.

## USER ACCOUNTING OPTIONS

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

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

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

   Note: The ACCTON COPY file distributed with VM/370 contains the basic logic required to enhance system security based on the 3277

Operator Identification Card Reader feature. Additional checking may be added to examine or validate the data read from the identification card.

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

2. CP has no provision for writing the accounting records to disk.

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

4. If no punch is generated in the VM/370 system, accounting records are not queued for punching. The ACCTON and ACCTOFF copy files are still called, however.

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

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

The procedure for generating a saved system consists of two steps:

1. Configuring and assembling the NAMESYS macro (DMKSNT).

2. Loading the system to be saved and then invoking the SAVESYS command.

When allocating DASD space for named systems, provide an extra page for information purposes; do not overlay this area with subsequent named systems.

## THE NAMESYS MACRO FOR SAVED SYSTEMS

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

When making additions, changes, or deletions to the system name table, the DMKSNT module must be reassembled. The GENERATE EXEC procedure has the facility to reassemble only the DMKSNT module. See the description of the GENERATE EXEC procedure in the VM/370: Planning and System Generation Guide.

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

The format of the NAMESYS macro is:

```
label | NAMESYS | SYSSIZE=nnnnnK,SYSNAME=name,VSYSRES=cccccc,
      |         | VSYSADR={cuu    },SYSVOL=cccccc,SYSCYL=nnn,
      |         | SYSSTRT=(cc,p),SYSPGCT=pp,
      |         | SYSPGNM=(nn,nn,nn-nn,...),
      |         | SYSHRSG=(s,s,...)
```

where:

label is any desired user label.

SYSSIZE=nnnnnK
: is the minimum amount of storage you must have available in order to load the saved system. K must be specified.

SYSNAME=name
: is the name (one-to-eight alphameric characters) given to the system to be used for identification by the SAVESYS command.

  The name selected must never be one that could be interpreted as a hexadecimal device address (for example, 'A' or 'E').

VSYSRES=cccccc
: is the real volume serial number of the DASD volume containing the virtual disk that is the system residence volume for the system to be saved.

VSYSADR=cuu
: is the virtual address of the virtual disk that is the system residence volume for the system to be saved.

SYSVOL=cccccc
: is the volume serial number (up to six alphameric characters) of the DASD volume designated to receive the saved system. This must be a CP-owned volume.

SYSCYL=nnn
: is the real starting cylinder of the virtual disk (specified by VSYSRES and VSYSADR) that is the system residence volume for the system to be saved.

SYSSTRT=(cc,p)
: designates the starting cylinder (cc) and page address (p) on SYSVOL at which this named system is to be saved. During the SAVESYS and IPL processing, this is used to generate the "cylinder page and device" address for the DASD operations. These numbers are specified in decimal.

  The number of pages written to this area is the total number specified via the SYSPGNM operand, plus one information page.

SYSPGCT=pp
: is the total number of pages (pp) you specify to be saved (that is, the total number of pages you indicate via the SYSPGNM operand). This is a decimal number, up to two digits.

SYSPGNM=(nn,nn,nn-nn,...)
: are the numbers of the pages to be saved. Pages may be specified singly or in groups. For example: if pages 0, 4, and 10 through 13 are to be saved, use the format: SYSPGNM=(0,4,10-13).

SYSHRSG=(s,s,...)
: are the segment numbers designated as shared. The pages in these segments are set up at load time to be used by any user loading by this name. All segments to be shared must be reentrant.

For example, a DMKSNT module to create a named CMS system could be coded as follows:

```
DMKSNTBL CSECT
FSTNAME NAMESYS  SYSSIZE=384K,SYSNAME=CMS,VSYSRES=CPDSK1,              X
                 VSYSADR=190,SYSCYL=100,SYSVOL=CPDSK2,                 X
                 SYSSTRT=(400,1),SYSPGCT=35,                           X
                 SYSPGNM=(0-34),SYSHRSG=(1)
         END
```

## USING THE SAVESYS COMMAND

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

where:

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

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

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

CMS was designed to run under CP and it was also designed so that it could easily be saved by CP. See "Saving the CMS System" in "Part 3. Conversational Monitor System (CMS)" of this publication.

## SHARED SEGMENTS

If one or more segments of a saved system are designated as being "shared", a single copy of these segments in real storage can be used by any virtual machine that loads the saved system by name. A shared segment must be reentrant and the segment number must be included in the SYSHRSG operand of the NAMESYS macro for the saved system.

In the previous example of a DMKSNT module to create a named CMS system, the NAMESYS macro labeled FSTNAME contains the operand:

```
SYSHRSG=(1)
```

This indicates that segment 1 of CMS is to be shared. When CMS is saved, via the SAVESYS command, the pages in segment 1 are set up so that any user loading CMS by name will share the same set of these pages in real storage. This results in a saving of both real and external page storage. Also, the more virtual machines using the shared segment,

| the more likely it is that these pages will be frequently referenced
| and, thereby, kept in real storage. As a result, the number of page
| faults and the corresponding time and resources expended in page
| swapping will be reduced.

## SPECIAL CONSIDERATIONS FOR SHARED SEGMENTS

When a saved system containing one or more shared segments is again saved, a problem can occur if the following conditions are present:

1. The previous system has been loaded by name before the new system was saved, and is still in use.

2. The unshared segments contain at least one address pointing to data in a shared segment that has moved as a result of a change.

3. The new system has been loaded by name.

The problem is that when the new system is loaded, it will use the old system's shared segments if one or more users of the old system are still logged on. Therefore, new versions of named systems containing shared segments (for example, CMS) should not be saved as long as the above conditions exist.

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

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

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

## DISCONTIGUOUS SAVED SEGMENTS

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

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

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

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

| An example of a discontiguous saved segment is the segment of CMS
| that supports DOS program development and testing under CMS. This

segment is reentrant and is named CMSDOS. The VM/370 starter system includes an EXEC procedure that helps you load and then save this segment. CMS contains all the necessary linkage to load the CMSDOS segment when it is needed.

## USER REQUIREMENTS

To use discontiguous saved segments you must:

- Allocate permanent space on a CP-owned volume to contain the saved segment.
- Assign a name to the segment and specify where it is to be stored on disk. To do this, define an entry in the system name table (DMKSNTBL) with the NAMESYS macro.
- Load and save the segment. The VM/370 starter system has EXEC procedures to help you load and save the discontiguous saved segments for CMS (one EXEC procedure to load and save CMS/DOS, one for CMS/VSAM and AMSERV, and one for the CMS Editor, EXEC processor, and OS simulation routines).
- Be sure that the proper linkage for attaching and detaching discontiguous saved segments is in the operating system that needs the segment. CMS contains the linkage necessary to attach and detach the discontiguous saved segments it supports.

Usually, the direct access storage space is allocated and the system name table entries are created during system generation. You allocate DASD space as permanent (PERM) by executing the Format/Allocate program. This program is executed during system generation, but it is a standalone program that can be executed at any time. During system generation, you designate the CP-owned volumes by coding the SYSOWN macro of the DMKSYS file. The system name table (DMKSNT) is also created during system generation. If, at some time after system generation, you wish to change the DMKSYS or DMKSNT files, you can do a partial system generation and reassemble those files using the GENERATE EXEC procedure. GENERATE is described in the VM/370: Planning and System Generation Guide. You can also load and save a discontiguous saved segment any time after system generation.

## THE NAMESYS MACRO FOR DISCONTIGUOUS SAVED SEGMENTS

Use the NAMESYS macro to define the name and location of discontiguous saved segments. The NAMESYS macro is the same one that is used to define the name and location of saved systems except that two of the operands are ignored and another has a mandatory set value. VSYSADR=IGNORE should be coded when the NAMESYS macro is describing a discontiguous saved system. For discontiguous saved segments, the format of the NAMESYS macro is:

```
| label | NAMESYS | SYSSIZE=nnnnnK,SYSNAME=name,VSYSRES=cccccc,
|       |         | VSYSADR={IGNORE},SYSVOL=cccccc,SYSCYL=nnn,
|       |         | SYSSTRT=(cc,p),SYSPGCT=pp,
|       |         | SYSPGNM=(nn,nn,nn-nn,...),
|       |         | SYSHRSG=(s,s,...)
```

where:

label is any desired user label.

SYSSIZE=nnnnnK
is the minimum amount of storage you must have available in order to load the saved system. K must be specified. Although you must code this operand, it is not used for discontiguous saved segments.

SYSNAME=name
is the name (one-to-eight alphameric characters) given to the discontiguous segment to be used for identification by the SAVESYS command and FINDSYS/LOADSYS DIAGNOSE instruction.

The name selected must never be one that could be interpreted as a hexadecimal device address (for example, 'A' or 'E').

VSYSRES=cccccc
This operand is ignored if VSYSADR=IGNORE.

VSYSADR=IGNORE
indicates that the NAMESYS macro is describing a system or segment that does not require a virtual system residence volume. Code VSYSADR=IGNORE when you are defining a discontiguous saved segment.

SYSVOL=cccccc
is the volume serial number (up to six alphameric characters) of the DASD volume designated to receive the saved system. This must be a CP-owned volume.

SYSCYL=nnn
This operand is ignored if VSYSADR=IGNORE.

SYSSTRT=(cc,p)
designates the starting cylinder (cc) and page address (p) on SYSVOL at which this named system is to be saved. During the saving and loading processing, this is used to generate the "cylinder page and device" address for the DASD operations. These numbers are specified in decimal.

The number of pages written to this area is the total number specified via the SYSPGNM operand, plus one information page.

SYSPGCT=pp
is the total number of pages (pp) you specify to be saved (that is, the total number of pages you indicate via the SYSPGNM operand). This is a decimal number, up to two digits.

SYSPGNM=(nn,nn,nn-nn,...)
are the numbers of the pages to be saved. Pages may be specified singly or in groups. For example: if pages 0, 4, and 10 through 13 are to be saved, use the format: SYSPGNM=(0,4,10-13).

SYSHRSG=(s,s,...)
are the segment numbers designated as shared. The pages in these segments are set up at load time to be used by any user loading by this name. All segments to be shared must be reentrant.

## LOADING AND SAVING DISCONTIGUOUS SHARED SEGMENTS

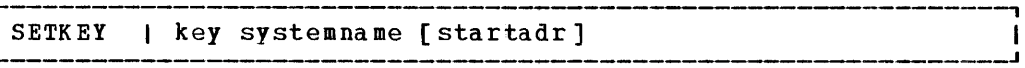
Before a discontiguous saved segment can be attached and detached by name, it must be loaded and saved. The discontiguous saved segment must be loaded at an address that is beyond the highest address of any virtual machine that will attach it. It is the system programmer's responsibility to make sure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

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

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

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

The format of the CMS SETKEY command is:




where:

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

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

startadr is the starting address (in hexadecimal) at which the keys are to be assigned. The address must be within the address range defined for the saved system or discontiguous saved segments. Using the startadr operand, you can issue the SETKEY command several times and thus assign different keys to various portions of the saved system or segment.

## HOW THE INTERFACE WORKS

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

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

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

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

```
SET NONSHARE segmentname
```

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

When CMS loads a named segment in shared mode it issues the CP DIAGNOSE Code X'64' with subcode X'00'. CMS also issues the same code with subcode X'04' to load the named segment in nonshared mode.

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

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

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

When the named segment is no longer needed, it can be detached. The CP DIAGNOSE Code X'64' subcode X'08', is called the PURGESYS function; it detaches named segments. When a named segment is detached, its storage is no longer addressable by the virtual machine and CP updates its segment tables. The entries for segments beyond the original virtual machine size are deleted and the associated real storage is released.

## SHARED SEGMENT PROTECTION

Before CP selects a new user to be dispatched, it checks that the current virtual machine has not altered any pages that are in shared

segments. If the current virtual machine has altered a shared page, it is placed in nonshared mode; the system is assigned to the virtual machine that altered it. Then, a fresh copy of the shared system is loaded and new tables are created for the remaining shared system users.

The paging supervisor checks the keys of any shared pages that are stolen. Whenever a modified page is detected, the user responsible is assigned a nonshared copy of the system.

Shared segment protection supports:

- The Virtual Machine Assist feature for named shared systems.
- The execution of all options of the CP STORE command in shared segments, including branch and instruction tracing.
- The execution of the CP STORE and ADSTOP commands in shared segments.
- The execution of the STORE and BREAK subcommands of the CMS DEBUG command.

CP's handling of storage keys includes the following:

- No distinction is made between shared and nonshared systems for storage key fetch instruction simulation, DISPLAY command execution, and page key handling.
- The virtual PSW key is not changed from X'0' to X'F' for a virtual machine that uses shared systems.
- A mask in control register 6 prevents the ISK (insert storage key) and SSK (set storage key) instructions from being handled by the VMA feature. This is necessary because VMA updates the key on SSK instructions (including the SWPTABLE fields), but the new value is not detected by the hardware change bit monitoring.

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

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

## VIRTUAL MACHINE OPERATION

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

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

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

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

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

You must enter the BEGIN command to continue execution. Again, your virtual machine is assigned the altered system in nonshared mode and other users continue as before with an unaltered system in shared mode.

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

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

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

# VM/VS Handshaking

VM/VS Handshaking is a communication path between the VM/370 Control Program and a virtual machine operating system (OS/VS1) that makes each system control program aware of any capabilities or requirements of the other. VM/VS Handshaking consists of:

- Closing CP spool files when VS1 job output from its DSO, terminator, and output writer is completed
- Processing VS1 pseudo page faults
- Providing an optional nonpaging mode for VS1 when it is run under the control of VM/370
- Providing miscellaneous enhancements for VS1 when it is run under the control of VM/370

The handshaking feature improves the operational characteristics of VS1 with VM/370 and yet allows the same VS1 operating system to run without change in either (1) a real machine or (2) a virtual machine under the control of VM/370. When the VM/VS Handshaking feature is active, the operation of VS1 with VM/370 more closely resembles the standalone operation of VS1. There is less need for virtual machine operator intervention because VS1 closes its CP spool files so they can be processed by VM/370 when the job output from the VS1 DSO, terminator, and output writer is complete. Also, one VS1 task can be dispatched while another is waiting for a page to be brought into real storage if the pseudo page fault handling portion of handshaking is active. With nonpaging mode, duplicate paging can be eliminated.

Although handshaking is a system generation feature for VS1, it is active only when VS1 is run under the control of VM/370; it is disabled when that same VS1 operating system is run on a real machine. Storage for the VS/1 virtual machine should be no larger than 4M. The VM/VS Handshaking feature is not active unless:

- VS1 is generated with the VM/370 option.
- VS1 is run under the control of a version of VM/370 that supports the feature (VM/370 supports handshaking with Release 2 PLC 13.)

The pseudo page fault portion of the handshaking feature is not active unless it is set on. It can be set on, and later set off, with the CP SET PAGEX command line.

When a VS1 virtual machine with the handshaking feature is loaded, its initialization routines determine whether the handshaking feature should be enabled or not. First, VS1 checks to see if it is running under the control of VM/370 by issuing an STIDP (Store Processor ID) instruction. STIDP returns a version code; a version code of X'FF' indicates VS1 is running under VM/370. If VS1 finds a version code of X'FF', it then issues a DIAGNOSE code 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 point in the VS1 initialization process, VM/VS Handshaking support is available. If VS1 is running in the nonpaging mode and if the virtual machine operator issues the CP SET PAGEX ON command, full VM/VS Handshaking support is available.

## CLOSING CP SPOOL FILES

When the handshaking feature is active, VS1 closes the CP spool files when the job output from the VS1 DSO, terminator, and output writer is complete. Once the spool files are closed, they are processed by VM/370 and sent to the real printer or punch. With the VM/VS Handshaking feature, virtual machine operator intervention is not required to close CP spool files.

During its job output termination processing, VS1 issues DIAGNOSE code X'08' instructions to pass the CP CLOSE command to VM/370 for each CP spool file.

## PSEUDO PAGE FAULTS

A page fault is a program interrupt that occurs when a page that is marked "not in storage" is referred to by an instruction within an active page. The virtual machine operating system 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 any other VS1 task. Thus, when VS1 uses pseudo page faults, its execution under the control of VM/370 more closely resembles its execution on a real machine.

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 interrupts. Then, VM/370 reflects the page faults to VS1 by:

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

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

When the requested page is available in real storage, VM/370 reflects the same program interrupt to VS1, except that 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 is run under the control of VM/370, it executes in nonpaging mode if:

- Its virtual address space is equal to the size of the VM/370 virtual machine
- Its virtual machine size is at least one megabyte
- 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, that the working set size may be larger for a VS1 virtual machine in nonpaging mode than for one not in nonpaging mode.

## MISCELLANEOUS ENHANCEMENTS

When OS/VS1 is run in the VM/370 environment without the handshaking feature, some duplication results. VS1 must perform certain functions when it is run on a real machine; it continues to perform all those functions in a VM/370 virtual machine even though VM/370 also provides services. However, with the handshaking feature, VS1 avoids many of the instructions and procedures that are redundant or less efficient in the VM/370 environment. For example, VS1 avoids:

- ISK (Insert Storage Key) instructions and instead uses a key table
- Seek separation for 2314 direct access devices
- The ENABLE/DISABLE sequence in the VS1 I/O Supervisor (IOS)
- TCH (Test Channel) instructions preceding SIO (Start I/O) instructions

## Timers in a Virtual Machine

This section describes the results obtained in using timers in a virtual machine created by CP.

### INTERVAL TIMER

Virtual location 80 (X'50'), the interval timer, contains different values than would be expected when operating in a real machine. On a real machine, the interval timer is updated 60 times per second when enabled and when the real machine is not in manual state. The interval timer on a real machine thus reflects system time and wait state time. In a virtual machine, the interval timer reflects only virtual CPU time, and not wait time. It is updated by CP whenever a virtual machine passes control to CP, and this one updating reflects the entire time the virtual machine had control. Note that during the time a virtual machine has control, the virtual interval timer does not change; the virtual CPU time used is added to the virtual interval timer when CP regains control. For some privileged instructions, CP may be able to simulate the instruction and still return control to the virtual machine before the end of that virtual machine's time slice. In such cases, the virtual interval timer is updated but only for those privileged instructions which require normal or fast reflect entry into the dispatcher. For those privileged instructions which do not require entry into the dispatcher, the virtual interval timer is not updated until CP gets control at the end of the time slice.

If the virtual machine assist feature is ON, more time is charged to the virtual interval timer than if the feature is OFF. When the virtual machine assist feature is OFF, the time spent by CP to simulate privileged instructions is not charged to the virtual interval timer; whereas, with the feature ON, the time spent by virtual machine assist to execute privileged instructions is charged to the virtual interval timer.

VM/370 provides an option, called the REALTIMER option, which causes the virtual interval timer to be updated during virtual wait state as well. With the REALTIMER option in effect, a virtual interval timer reflects virtual CPU time and virtual wait time, but not CP time used for services for that virtual machine, such as privileged instruction execution. The more services a virtual machine requires from CP, the greater the difference between the time represented by the interval timer and the actual time used by and for the virtual machine. The larger the number of active virtual machines contending for system resources, the greater the difference between virtual machine time and actual elapsed (wall clock) time.

### CPU TIMER

A virtual machine must have the ECMODE directory option to use the System/370 CPU timer.

The CPU timer is supported in a virtual machine in much the same way as is the interval timer. That is, the CPU timer in a virtual machine

records only virtual CPU time, and it is updated when the virtual machine passes control back to CP.

If the real timer option is specified, the CPU timer reflects all actual elapsed time except CP time used for services, such as privileged instruction execution, for that virtual machine.

The method of sampling the value in the CPU timer causes it to appear to a virtual machine to be updated more often than an interval timer. The privileged instructions Set CPU Timer (SPT) and Store CPU Timer (STPT) are used to set a doubleword value in the CPU timer and to store it in a doubleword location of virtual storage. When a virtual machine samples the value in the CPU timer by issuing a STPT instruction, CP regains control to execute the privileged instruction, and updates the time. The act of sampling the CPU timer from a virtual machine causes it to be brought up to date.

## TOD CLOCK

The System/370 time-of-day (TOD) clock does not require simulation in a virtual machine. The System/370 in which CP is operating has one real TOD clock, and all virtual machines can interrogate that real TOD clock. The Store Clock (STCK) instruction is nonprivileged; any virtual machine can execute it to store the current value of the TOD clock in its virtual storage. The Set Clock (SCK) instruction, which is used to set the TOD Clock value can be issued from a virtual machine, but CP always returns a condition code of zero, and does not actually set the clock. Note that the TOD clock is the only true source of actual elapsed time information for a virtual machine. The base value for the TOD clock in VM/370 is 00:00:00 GMT January 1, 1900.

## CLOCK COMPARATOR

The clock comparator associated with the TOD clock is used in virtual machines for generating interrupts based on actual elapsed time. The ECMODE option must be specified for a virtual machine to use the clock comparator feature. The Set Clock Comparator (SCKC) instruction specifies a doubleword value which is placed in the clock comparator. When the TOD clock passes that value, an interrupt is generated.

## PSEUDO TIMER

The pseudo timer is a special VM/370 timing facility. It provides 24 or 32 bytes of time and date information in the format shown in Figure 29.

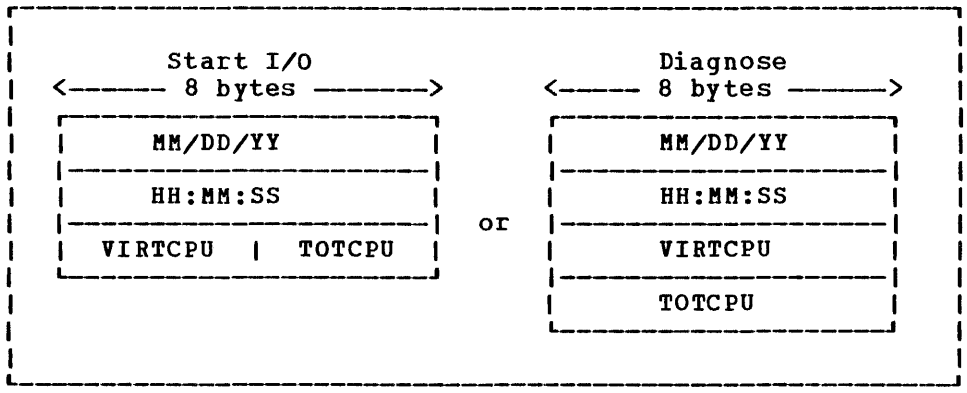



Figure 29. Formats of Pseudo Timer Information

The first eight-byte field is the date, in EBCDIC, in the form Month/Day-of-Month/Year. The next eight-byte field is the Time of Day in Hours:Minutes:Seconds. The VIRTCPU and TOTCPU fields contain virtual CPU and total CPU time used. The units in which the CPU times are expressed and the length of the fields depend upon which of two methods is used for interrogating the pseudo timer.

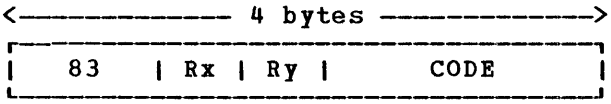
## PSEUDO TIMER START I/O

The pseudo timer can be interrogated by issuing a START I/O to the pseudo timer device, which is device type TIMER, and is usually at device address 0FF. No I/O interrupt is returned from the SIO. The address in virtual storage where the timer information is to be placed is specified in the data address portion of the CCW associated with the SIO. This address must not cross a page boundary in the user's address space. If this method is used, the virtual CPU and the total CPU times are expressed as fullwords in high resolution interval timer units. One unit is 13 microseconds.

## PSEUDO TIMER DIAGNOSE

The pseudo timer can also be interrogated by issuing DIAGNOSE with an operation code of C, as described under "DIAGNOSE Instruction in a Virtual Machine." If this method is used, the virtual and total CPU times are expressed as doublewords in microseconds.

The DIAGNOSE instruction cannot be used in a virtual machine for its normal function. If a virtual machine attempts to execute a DIAGNOSE instruction, a program interrupt returns control to CP. Since a DIAGNOSE instruction issued in a virtual machine results only in returning control to CP, and not in performing normal DIAGNOSE functions, the instruction is used for communication between a virtual machine and CP. The machine language format of DIAGNOSE is:




(There is no Assembler language mnemonic for X'83')

The operand storage addresses, passed to the DIAGNOSE interface in Rx and Ry, must be real addresses to the virtual machine issuing the DIAGNOSE.

The Code is a two-byte hexadecimal value that CP uses to determine what function to perform. The codes defined for the general VM/370 user are described in this section. The code must be a multiple of 4. Codes X'00' through X'FC' are reserved for IBM use, and codes X'100' through X'1FC' are reserved for users.

Because DIAGNOSE operates differently in a virtual machine than in a real machine, a program should determine that it is operating in a virtual machine before issuing a DIAGNOSE, and prevent execution of a DIAGNOSE when in a real machine. The Store CPU ID (STIDP) instruction provides a program with information about the CPU in which it is executing, including the CPU version number. If STIDP is issued from a virtual machine the version number will be 'FF', in the first byte of the CPUID field.

A virtual machine issuing a Diagnose instruction should run with interrupts disabled. This prevents loss of status information pertaining to the Diagnose operation such as condition codes and sense data.

## DIAGNOSE CODE 0 -- STORE EXTENDED-IDENTIFICATION CODE

Execution of 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 with supports the VM/VS Handshaking feature. If the extended-identification code is returned to VS1, VM/370 supports handshaking; otherwise, it does not.

The register specified as Rx contains the doubleword aligned virtual storage address where the VM/370 extended-identification code is to be stored. The Ry register contains the number of bytes to be stored entered as 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 | Characteristics |
|---|---|---|
| System Name | "VM/370" | 8 bytes, EBCDIC |
| Version Number | The first byte is the version number, the second byte is the level, and the third byte is the PLC (Program Level Change) number. | 3 bytes, hexadecimal |
| Version Code | VM/370 executes the STIDP (Store CPU ID) instruction to determine the version code. | 1 byte, hexadecimal |
| MCEL | VM/370 executes the STIDP instruction to determine the maximum length of the MCEL (Machine Check Extended Logout) area. | 2 bytes, hexadecimal |
| Processor Address | VM/370 executes the STAP (Store CPU Address) instruction to determine the processor address. | 2 bytes, hexadecimal |
| Userid | The userid of the virtual machine issuing the DIAGNOSE. | 8 bytes, EBCDIC |

If VM/370 is executing in a virtual machine, another 24 bytes, or less, of extended identification data is appended to the first 24 bytes described above. Up to five nested levels of VM/370 virtual machines are supported by this Diagnose instruction resulting in a maximum of 120 bytes of data that can be returned to the virtual machine that initially issued the Diagnose instruction.

Upon return, Ry contains its original value less the number of bytes that were stored.

No completion code is returned, and the condition code remains unchanged.

## DIAGNOSE CODE 4 -- EXAMINE REAL STORAGE

Execution of a DIAGNOSE Code 4 allows a user with command privilege class C or E to examine real storage. The register specified as Rx contains the virtual address of a list of CP (real) addresses to be examined. The Ry register contains the count of entries in the list. Ry+1 contains the virtual address of the result field. The result field contains the values retrieved from the specified real locations.

## DIAGNOSE CODE 8 -- VIRTUAL CONSOLE FUNCTION

The execution of DIAGNOSE with code 8 allows a program executing in supervisor mode in a virtual machine to perform a CP command. The register specified as Rx contains the address, in virtual storage, of

the data area defining the CP command and parameters. The Ry register contains the length of the associated command input, which may be up to 132 characters. The following example illustrates how DIAGNOSE Code 8 would be issued to perform the CP command, QUERY, to determine the number of input and output spool files:

```
LA       6,CMMD
LA      10,CMMDL
DC      X'83',X'6A',XL2'0008'
.
.
.
CMMD   DC       C'QUERY FILES'
CMMDL  EQU      *-CMMD
.
.
.
```

The output of the command is at the user's terminal. A completion code is returned to the user as a value in the register specified as Ry. In the example above, it would be register 10. A completion code of 0 signifies normal completion. If there is an error, the completion code is the binary value of the numeric portion of the error message. For instance, the error message

```
DMKCFM045E userid NOT LOGGED ON
```

returns '045' in the Ry register. The condition code remains unchanged.

## DIAGNOSE CODE C -- PSEUDO TIMER

Execution of DIAGNOSE with Code C causes CP to store four doublewords of time information in the user's virtual storage. The register specified as Rx contains the address of the 32 byte area where the time information is to be stored. The address must be a doubleword boundary. The information returned is as shown in Figure 29.

The first eight bytes contain the Month/Day-of-Month/Year. The next eight bytes contain the time of day in Hours:Minutes:Seconds. The last 16 bytes contain the virtual and total CPU time used by the virtual machine that issued the DIAGNOSE. These times are expressed as doubleword, unsigned integers, in microseconds. One-hundredths of seconds are not returned. No completion code is returned, and the condition code remains unchanged.

## DIAGNOSE CODE 10 -- RELEASE PAGES

Pages of virtual storage can be released by issuing a DIAGNOSE with Code 10. When a page is released it is considered all zero. The register specified by Rx contains the address of the first page to be released, and the Ry register contains the address of the last page to be released. Both addresses must be page boundaries. A page boundary is a storage address whose low order three digits, expressed in hexadecimal, are zero. No completion code is returned, and the condition code remains unchanged.

## DIAGNOSE CODE 14 -- INPUT SPOOL FILE MANIPULATION

Execution of DIAGNOSE Code 14 causes DMKDRDER to perform input spool file manipulation. Depending on the value of the function subcode, the register specified as Rx contains a buffer address, a copy count, or a spool file identifier. The Ry register contains either the virtual address of a spool input card reader or, if Ry+1 contains X'0FFF', a spool file ID number. Ry+1 contains a hexadecimal code indicating the file manipulation to be performed. The codes are:

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

On return Ry+1 may contain error codes which further define a returned condition code of 3.

| Condition Code | Ry+1 | Error |
|---|---|---|
| 0 | | Data transfer successful |
| 1 | | End of file |
| 2 | | File not found |
| 3 | 4 | Device address invalid |
| 3 | 8 | Device type invalid |
| 3 | 12 | Device busy |
| 3 | 16 | Fatal paging I/O error |

### SUBCODE X'000'

Rx = start address of full-page virtual buffer
Ry = virtual spool reader address

The specified device is checked for an already active file, and if there is one, the next full-page buffer is made available to the virtual machine via a call to DMKRPAGT. If there is no active file, the chain of reader files is searched for a file for the calling user and connected to the virtual device for further reading. If no file is found, virtual condition code 2 is set. When the end of an active file is reached, the device status settings are tested for 'spool continuous'. If not set, virtual condition code 1 is set, indicating end of file. If the device is set for continuous input, the active file is examined to determine if it is a multiple-copy file. If so, reading is restarted at the beginning of the file. If not, the file is closed via DMKVSPCR and the reader chain is searched for another input file. If no other file is found, virtual condition code 1 is set.

### SUBCODE X'004'

Rx = virtual address of a 12-doubleword buffer
Ry = virtual spool reader address

If the specified device is in use via diagnose, the VSPLCTL block is checked to see if this is a repeated call for printer SFBLOKs. If yes, then the chain search continues from the point where the last SFBLOK was given to the VM. In this case, CC = 1 is set when there are no more print files. If this is the first call for an SFBLOK, or if there have been intervening calls for file reading, the spool input chain is searched from the beginning, and cc = is set if no files are found.

Note: The virtual buffer specified via Rx must not cross a page boundary or a specification exception will result.

SUBCODE X'008'

Rx = virtual address of a 12-doubleword buffer
Ry = virtual spool reader address

Processing for code 8 is the same as for code 4, except that only card-image input files are processed.

Note: For both codes 4 and 8, the format definition for a VM/370 SFBLOK can be found in the system macro library.

SUBCODE X'00C'

Rx = file identifier of requested file

The spool input chain is searched for the file specified. If it is not found, cc=2 is set. If it is found, the file is moved to the head of the chain such that it will be the next file processed by any of the other functions.

SUBCODE X'010'

Rx = new copy count for the active file
Ry = virtual spool reader address

The specified device is checked for an active file. If no file is active, cc=2 is set. Otherwise, the copy COUNT for the file is set to the specified value, with a maximum of 255. If the specified count is not positive, a specification exception is generated. If the count is greater than 255, it is adjusted to module 256.

SUBCODE X'014'

Rx = start address of virtual full-page buffer
Ry = virtual spool reader address

The specified device is checked for an active file. If no active file is found, cc=2 is set. Otherwise, the VSPLCTL pointers are reset to the beginning of the file and the first page record is given to the user as for code 0.

SUBCODE X'018'

Rx = start address of virtual full-page buffer
Ry = virtual spool reader address

The specified device is checked for an active file. If no active file is found, cc=2 is set. Otherwise, the file is backspaced one record and the record is given to the user as in code 0. If the file is already positioned at the first record, the first record is given to the user.

SUBCODE X'FFF'

Rx = Virtual address of a 244-byte buffer
Ry = Spool file ID number

If Ry is non-zero the spool input chain is searched for a file with a matching ID number; if none is found, or if one is found which is owned by a different virtual machine, cc=2 is set. The chain search is continued from the file which was found, or from the anchor if Ry is zero, for the next file owned by the caller, independent of file type, class, INUSE flag, etc. If none is found, cc=1 is set. Otherwise, the SFBLOK and the first record of the file (generally, the TAG) are copied to the caller's virtual storage buffer.

## DIAGNOSE CODE 18 -- STANDARD DASD I/O

Input/output operations to a direct access device of the type used by CMS, can be performed from a virtual machine using DIAGNOSE with Code 18. No I/O interrupts are returned by CP to the virtual machine; the DIAGNOSE instruction is complete only when the Read or Write commands associated with the DIAGNOSE are completed. The Rx register contains the virtual device address of the direct access device. The Ry register contains the address of a chain of CCWs. The CCW chain must be in a standard format that CP expects when DIAGNOSE Code 18 is used, as shown below. Register 15 must be loaded by the user with the number of reads or writes in the CCW chain.

A typical CCW string to read or write two 800-byte records is as follows:

```
      SEEK,A,CC,6
      SET SECTOR (not used for 2314/2319)
      SRCH,A+2,CC,5
      TIC,*-8,0,0
      RD or WRT,DATA,CC+SILI,800
      SEEK HEAD,B,CC,6 (omitted if HEAD number unchanged)
      SET SECTOR
      SRCH,B+2,CC,5
      TIC,*-8,0,0
      RD or WRT,DATA+800,SILI,800

A     SEEK and SRCH arguments for first RD/WRT
B     SEEK and SRCH arguments for second RD/WRT
```

The Condition Codes and Completion Codes returned are as follows:

cc=0 I/O complete with no errors

cc=1 Error condition. Register 15 contains one of the following:
R15=1 Device not attached
R15=2 Device not 2319, 2314, 3330, 3340, or 3350
R15=3 Attempt to write on a Read-only disk
R15=4 Cylinder number not in range of user's disk
R15=5 Virtual device is busy or has an interrupt pending

cc=2 Error condition. Register 15 contains one of the following:
R15=5 Pointer to CCW string not doubleword aligned.
R15=6 SEEK/SEARCH arguments not within range of user's storage
R15=7 READ/WRITE CCW is neither Read (06) nor Write (05)
R15=8 READ/WRITE Byte Count=0
R15=9 READ/WRITE Byte Count greater than 2048
R15=10 READ/WRITE buffer not within user's storage
R15=11 The value in R15, at entry, was not a positive number from 1 through 15; or, was not large enough for the given CCW string.
R15=12 Cylinder number on seek head was not the same number as on the first seek.

cc=3 Uncorrectable I/O error:
R15=13
CSW (8 bytes) returned to user
Sense bytes are available if user issues a SENSE command

## DIAGNOSE CODE 1C -- CLEAR I/O RECORDING

Execution of DIAGNOSE Code 1C allows a user with privilege class F to clear the I/O error recording data on disk. The DMKIOEFM routine performs the clear operation. The register specified as Rx contains a code value:

| Code | Function |
|---|---|
| 1 | Clear and reformat all I/O error recording |
| 2 | Clear and reformat all machine check error recording |
| 3 | Clear and reformat all error recording (I/O and machine check) |

## DIAGNOSE CODE 20 -- GENERAL I/O

With DIAGNOSE Code 20, a virtual machine user can specify any valid CCW chain to be performed on a tape or disk device. No I/O interrupts are reflected to the virtual machine; the DIAGNOSE instruction is complete only when all I/O commands in the specified CCW chain are finished. The register specified as Rx contains the virtual device address. The Ry register contains the address of the CCW chain.

The CCW string is processed via DMKCCWTR through DMKGIOEX, providing full virtual I/O in a synchronous fashion (self-modifying CCW strings are not permitted, however) to any virtual machine specified. Control returns to the virtual machine only after completion of the operation or detection of a fatal error condition. EREP support is provided for tape and DASD devices only; all other devices will present an error condition in the PSW to the virtual user. Condition codes and error codes are returned to the virtual system.

The Condition Codes and Completion Codes returned are as follows:

cc=0 I/O complete with no errors

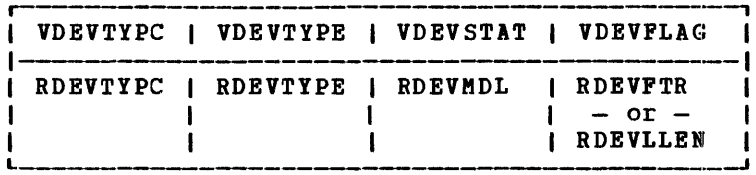
cc=1 Error condition. Register 15 contains the following:
R15=1 Device is either not attached or the virtual channel is dedicated.
R15=5 Virtual device is busy or has an interrupt pending.

cc=2 Exception conditions. Register 15 contains one of the following:
R15=2 Unit Exception bit in device status byte=1
R15=3 Wrong Length Record detected.

cc=3 Error Condition:
R15=13 A permanent I/O error occurred or an unsupported device was specified. The two low-order positions of the user's Ry register contain the first two sense bytes.

## DIAGNOSE CODE 24 -- DEVICE TYPE AND FEATURES

CP maintains control blocks describing each virtual device and each real device. DIAGNOSE Code 24 causes CP to return to the virtual machine certain information from the virtual device block (VDEVBLOK) and the real device block (RDEVBLOK) associated with a given virtual device address. The Rx register from the caller contains a virtual device address or a value of -1, indicating that the device is a virtual console and its address is not known. If the console is found, control returns to the caller with the virtual device address in the low order 2 bytes of the Rx register. The Ry register and the Ry+1 register, on return, contain the following one-byte fields:




The meanings of these fields are as follows:

| Ry Register | Virtual Device Information |
|---|---|
| VDEVTYPC | Virtual device type class |
| VDEVTYPE | Virtual device type |
| VDEVSTAT | Virtual device status |
| VDEVFLAG | Virtual device flags |

| Ry+1 Register | Real Device Information |
|---|---|
| RDEVTYPC | Real device type class |
| RDEVTYPE | Real device type |
| RDEVMDL | Real device model number |
| RDEVFTR | Real device feature code, for a device other than a virtual console |
| RDEVLLEN | Current device line length, for a virtual console |

Note: If Ry is register 15, only the virtual device information is returned to the caller.

A condition code of 3 indicates that the virtual device address specified was invalid, or that the virtual device does not exist. A condition code of 2 indicates the virtual device exists, but there is no real device associated with it, and therefore no real device information was provided. Spooling devices and Pseudo Timers are examples of such devices. A condition code of 0 indicates normal completion.

## DIAGNOSE CODE 28 -- CHANNEL PROGRAM MODIFICATION

DIAGNOSE Code 28 allows a virtual machine to correctly execute some channel programs modified after the Start I/O (SIO) instruction is issued and before the input/output operation is completed. The channel command word (CCW) modifications allowed are:

- A Transfer in Channel (TIC) CCW modified to a No Operation (NOP) CCW
- A TIC CCW modified to point to a new list of CCWs
- A NOP modified to a TIC CCW

When a virtual machine modifies a TIC CCW, it is modifying a virtual channel program. CP has already translated that channel program and is waiting to execute the real CCWs. The DIAGNOSE instruction, with Code 28, must be issued to inform CP of the change in the virtual channel program, so CP can make the corresponding change to the real CCW before it is executed. In addition, when a NOP CCW is modified to point to a new list of CCWs, CP translates the new CCWs.

To be sure that the DIAGNOSE instruction is recognized in time to update the real CCW chain, the virtual machine issuing the DIAGNOSE instruction should have a high favored execution value and a low dispatching priority value. The CP SET command should be issued:

```
SET FAVORED xx

SET PRIORITY nn
```

where xx has a high numeric value and nn has a low numeric value. The virtual machine issuing the DIAGNOSE Code 28 must be in the supervisor mode at the time it issues the DIAGNOSE instruction.

When DIAGNOSE Code 28 is issued, the Rx register contains the address of the TIC or NOP CCW that was modified by the virtual machine. The Ry register contains the device address in bits 16 through 31. Rx and Ry cannot be the same register. The addresses specified in the Rx register, the new address in the modified TIC CCW, and the new CCW list that the modified TIC CCW points to must all be addresses that appear real to the virtual machine: CP knows these addresses are virtual, but the virtual machine thinks they are real.

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

cc=0 The real channel program was successfully modified; register 15 contains a zero.

cc=1 There was probably an error in issuing the DIAGNOSE instruction. Register 15 (R15) contains one of the following completion codes:
R15=1 The same register was specified for Rx and Ry.
R15=2 The device specified by the Ry register was not found.
R15=3 The address specified by the Rx register was not within the user's storage space.

R15=4 The address specified by the Rx register was not doubleword aligned.
R15=5 A CCW string corresponding to the device (Ry) and address (Rx) specified was not found.
R15=6 The CCW at the address specified by the Rx register is not a TIC or a NOP, or the CCW in the channel program is not a TIC or a NOP.
R15=7 The new address in the modified TIC CCW is not within the user's storage space.
R15=8 The new address in the modified TIC CCW is not doubleword aligned.

cc=2 The real channel program cannot be modified because a channel end or device end already occurred. Register 15 contains a 9. The virtual machine should restart the modified channel program.

## DIAGNOSE CODE 2C -- RETURN DASD START OF LOGREC

Execution of DIAGNOSE Code 2C allows a user with privilege class C, E, or F to find the location on disk of the error recording area. The register specified as Rx, on return contains the DASD location (in VM/370 control program internal format) of the first record of the system I/O and machine check error recording area.

## DIAGNOSE CODE 30 -- READ ONE PAGE OF LOGREC DATA

Execution of DIAGNOSE Code 30 allows a user with privilege class C, E, or F to read one page of the system error recording area. The register specified as Rx contains the DASD location (in VM/370 control program internal format) of the desired record. The Ry register contains the virtual address of a page-size buffer to receive the data. The DMKRPAGT routine supplies the page of data. The condition codes returned are:

| Condition Code | Meaning |
|---|---|
| 0 | Successful read, data available |
| 1 | End of cylinder, no data |
| 2 | Invalid cylinder, outside recording area |

## DIAGNOSE CODE 34 -- READ SYSTEM DUMP SPOOL FILE

A user with privilege class C or E can read the system spool file by issuing a DIAGNOSE Code 34 instruction. The register specified as Rx contains the virtual address of a page-size buffer to receive the data. The Ry register, which cannot be register 15, contains the virtual address of the spool input card reader. Ry+1, on return, may contain error codes:

| Condition Code | Ry+1 Error Code | Meaning |
|---|---|---|
| 0 | | Data transfer successful |
| 1 | | End of file |
| 2 | | File not found |
| 3 | 4 | Device address invalid |
| 3 | 8 | Device type invalid |
| 3 | 12 | Device busy |
| 3 | 16 | Fatal paging I/O error |

The DMKDRDMP routine searches the system chain of spool input files for the dump file belonging to the user issuing the DIAGNOSE instruction. The first (or next) record from the dump file is provided to the virtual machine via DMKRPAGT and the condition code is set to zero. The dump file is closed via VM/370 console function CLOSE.

## DIAGNOSE CODE 38 -- READ SYSTEM SYMBOL TABLE

Execution of DIAGNOSE Code 38 causes the routine DMKDRDSY to read the system table into storage. The register specified as Rx contains the address of the page buffer to contain the symbol table.

## DIAGNOSE CODE 3C -- VM/370 DIRECTORY

Execution of DIAGNOSE Code 3C allows a user to dynamically update the VM/370 directory. The register specified as Rx contains the first 4 bytes of the volume serial label. The first two bytes of Ry contain the last 2 bytes of the volume serial label. The routine DMKUDRDS dynamically updates the directory.

## DIAGNOSE CODE 4C -- GENERATE ACCOUNTING CARDS FOR THE VIRTUAL USER

This code can be issued only by a user with the account option (ACCT) in his directory.

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 zeroes, the accounting card is punched with the identification of the user issuing the DIAGNOSE instruction.

Ry contains a hexadecimal code interpreted by DMKHVC as follows:

| Code | Rx points to: |
|---|---|
| 0000 | a parameter list containing only a userid. |
| 0004 | a parameter list containing a userid and account number. |
| 0008 | a parameter list containing a userid and distribution number. |
| 000C | a parameter list containing a userid, account number, and distribution number. |
| 0010 | a data area containing up to 70 bytes of user information to be transferred to the accounting card starting in column 9. |

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.

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

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

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

If both the virtual address and the length are valid, DMFREE is called to obtain storage for an account buffer (ACNTBLOK) which is then initialized to blanks. The userid of the user issuing the DIAGNOSE instruction is placed in columns 1 through 8 and an accounting 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 zero.

If Ry contains other than a 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 zero.
- If invalid, an addressing exception is generated.
- If not aligned on a doubleword boundary, a specification exception is generated.

For a parameter list address that is nonzero and valid, the userid in the parameter list is checked against the directory list and if not found, control is returned to the user with a condition code of two. If the function hexadecimal code is invalid, control is returned to the user with a condition code of three. If both userid and function hexadecimal code are valid, the User Accounting Block (ACCTBLOK) is built and the userid, account number, and distribution number are moved to the block from the parameter list or the User Machine Block belonging to the userid in the parameter list. Control is then passed to the user with a condition code of zero.

## DIAGNOSE CODE 50 -- SAVE THE 3704/3705 CONTROL PROGRAM IMAGE

(Privilege class A, B, or C only) DIAGNOSE Code 50 invokes the CP module DMKSNC to (1) validate the parameter list and (2) write the page-format image of the 3704/3705 control program to the appropriate system volume.

When a 3704/3705 control program load module is created, the CMS service program SAVENCP builds a communications controller list (CCPARM) of control information. It passes this information to CP via a DIAGNOSE Code X'0050'.

The register specified as Rx contains the virtual address of the parameter list (CCPARM). The Ry register is ignored on entry.

Upon return, the Ry register contains the following error codes:

| Code | Meaning |
|---|---|
| 044 | 'ncpname' was not found in system name table. |
| 171 | System volume specified not currently available. |
| 178 | Insufficient space reserved for program and system control information. |
| 179 | System volume specified is not a CP-owned volume. |
| 435 | Paging error while writing saved system. |

## DIAGNOSE CODE 54 -- CONTROL THE FUNCTION OF THE PA2 FUNCTION KEY

DIAGNOSE Code 54 controls the function of the PA2 function key. The PA2 function key can be used either to simulate an external interrupt to a virtual machine or to clear the output area of a display screen.

The function performed depends on how Rx is specified when DIAGNOSE Code 54 is issued. If Rx contains a nonzero value, the PA2 key simulates an external interrupt to the virtual machine. If Rx contains a value of zero, the PA2 key clears the output area of the display screen.

The external interrupt is simulated only when the display screen is in the VM READ, HOLD, or MORE status and the TERMINAL APL ON command has been issued.

## DIAGNOSE CODE 58 -- 3270 VIRTUAL CONSOLE INTERFACE

Execution of DIAGNOSE Code 58 allows a virtual machine to display large amounts of data on a 3270 in a very rapid fashion. The interface can display the entire 3270 screen with one write operation instead of 22 writes (one for each line in the output area of a 3270 screen).

The register specified as Rx contains the address of the console CCW string. The Ry register contains (in bits 16-31) the device address of the virtual console.

The format of the special display CCW is:

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

where:

dataddr is the beginning address of the data to be displayed.

flags is the standard CCW flag field with the SILI bit on.

ctl is a control byte that indicates the starting output display line. 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. The maximum number of bytes is 1760.

When the DIAGNOSE is executed with a valid CCW string, a buffer (whose length is the number of bytes specified by count) is built in free storage. The data pointed to by dataddr is loaded into the buffer. 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.

Using the starting output line (ctl) and the number of bytes of output (count), CP checks that the data will fit on the screen. CP then does the display. A zero condition code indicates the I/O operation completed successfully; a nonzero condition code indicates an I/O error occurred.

## DIAGNOSE CODE 5C: ERROR MESSAGE EDITING

Execution of DIAGNOSE Code 5C causes the editing of an error message according to the user's setting of the EMSG function:

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 follows:

| VMMLEVEL | | Registers on Return | |
|---|---|---|---|
| VMMCODE | VMMTEXT | Rx | Ry |
| ON | ON | no change | no change |
| ON | OFF | no change | 10 (length of code) |
| OFF | ON | pointer to text part of message | length of text alone |
| OFF | OFF | N/A | 0 |

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

## DIAGNOSE CODE X'60' - DETERMINING THE VIRTUAL MACHINE STORAGE SIZE

Execution of DIAGNOSE Code X'60' allows a virtual machine to determine its size. On return, the register specified as Rx contains the virtual machine storage size.

## DIAGNOSE CODE X'64' - FINDING, LOADING AND PURGING A NAMED SEGMENT

Execution of DIAGNOSE Code X'64' controls the linkage of discontiguous saved segments. The type of linkage that is peformed depends on the function subcode in the register specified as Ry.

| Subcode | Function |
|---|---|
| X '00' | LOADSYS-Loads a named segment in shared mode |
| X '04' | LOADSYS-Loads a named segment in nonshared mode |
| X '08' | PURGESYS-Releases the named segment from virtual storage |
| X '0C' | FINDSYS-Finds the starting address of the named segment |

The register specified as Rx must contain the address of the name of the segment. The segment name must be 8 bytes long, left justified, and padded with trailing blanks.

### The LOADSYS Function

When the LOADSYS Diagnose function is executed, CP finds the system name table entry for the segment and builds the necessary page and swap tables. CP releases all the virtual pages of storage that are to contain the named segment and then loads the segment in those virtual pages. When the LOADSYS function is executed, CP expands the virtual machine size dynamically, if necessary. CP also expands the segment tables to match any expansion of virtual storage.

When LOADSYS executes successfully, the address of where the named segment was loaded is returned in the register specified as Rx. When the LOADSYS function loads a segment in shared mode, it resets instruction and branch tracing, if either was active.

After a LOADSYS function executes, the storage occupied by the named segment is addressable by the virtual machine, even if that storage is beyond the storage defined for the virtual machine. However, any storage beyond that defined for the virtual machine and below that defined for the named segment is not addressable. Figure 30 shows the virtual storage that is addressable before and after the LOADSYS function executes.

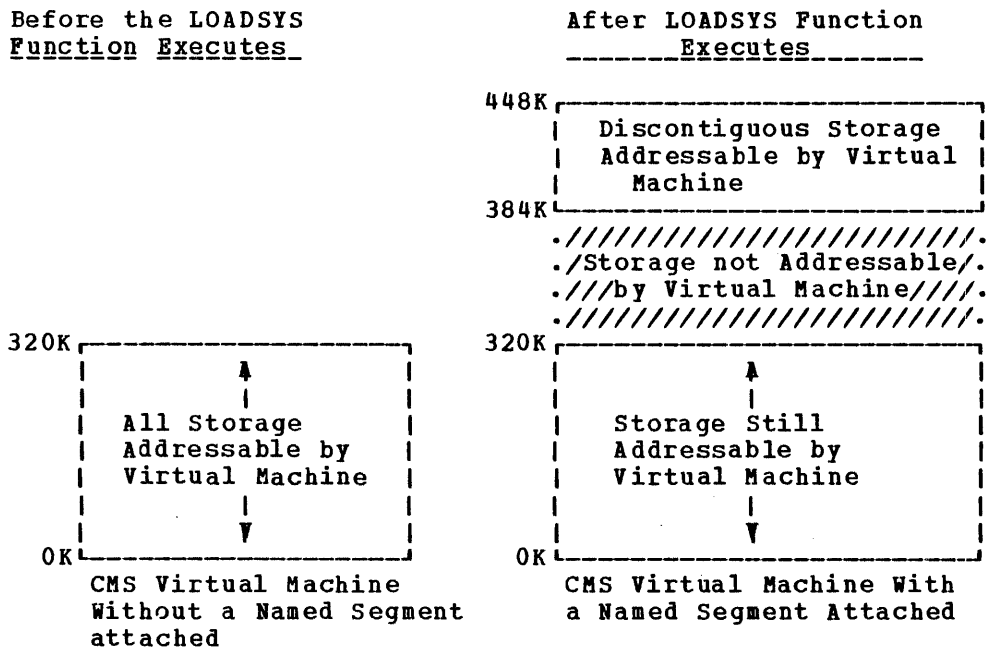



Figure 30. Addressable Storage Before and After a LOADSYS Function

When you save a named segment that is later loaded by the LOADSYS function, you must be sure that the addresses at which segments are saved are correct and do not overlay required areas of storage in the virtual machine. This is crucial because the LOADSYS function invokes the PURGESYS function before it builds the new page and swap tables. CP purges all saved systems that are overlayed in any way by the saved system it is loading.

A condition code of 0 in the PSW indicates that the named segment was loaded successfully; the Rx register contains the load address.

A condition code of 1 in the PSW indicates the named segment was loaded successfully within the defined storage of the virtual machine. The Rx register contains the address at which the named segment was loaded. The Ry register contains the ending address of the storage released before the named segment was loaded.

Note: CMS only allows named segments to be attached beyond the defined size of the virtual machine.

A condition code of 2 in the PSW indicates the LOADSYS function did not execute successfully. Examine the return code in the Ry register to determine the cause of the error.

| Return Code | Meaning |
|---|---|
| 44 | Named segment does not exist |
| 177 | Paging I/O errors |

## The PURGESYS Function

When the PURGESYS function is executed; CP releases the storage, and associated page and swap tables, that were acquired when the corresponding LOADSYS function was executed. If the storage occupied by the named segment was beyond the defined virtual machine storage size, that storage is no longer addressable by the virtual machine.

When a PURGESYS function is executed for a segment that was loaded in nonshared mode, the storage area is cleared to binary zeros. If PURGESYS is invoked for a named segment that was not previously loaded via LOADSYS, the request is ignored.

A condition code of 0 in the PSW indicates successful completion.

A condition code of 1 in the PSW indicates that the named segment was not found in the virtual machine.

A condition code of 2 in the PSW and a return code of 44 in the Ry register indicate that the named segment either does not exist or was not previously loaded via the LOADSYS function.

## The FINDSYS Function

When the FINDSYS function is executed, CP checks that the named segment exists and that it has not been loaded previously.

A condition code of 0 in the PSW indicates that the named segment is already loaded. The address at which it was loaded is returned in the register specified as Rx and its highest address is returned in the Ry register.

A condition code of 1 in the PSW indicates that the named segment exists but has not been loaded. In this case, the address at which the named segment is to be loaded is returned in the register specified as Rx and the highest address of the named segment is returned in the Ry register.

A condition code of 2 in the PSW indicates the FINDSYS function did not execute successfully. Examine the return code in the Ry register to determine the error that occurred.

| Return Code | Meaning |
|---|---|
| 44 | Named segment does not exist |
| 177 | Paging I/O errors |

## CP CODING CONVENTIONS

The following are coding conventions used by CP modules. This information should prove helpful if you debug, modify, or update CP.

1. FORMAT:

| Column | Contents |
|---|---|
| 1 | Labels |
| 10 | Op Code |
| 16 | Operands |
| 31, 36, 41, etc. | Comments (see Item 2) |

2. COMMENT:

   Approximately 75 percent of the source code contains comments. Sections of code performing distinct functions are separated from each other by a comment section.

3. CONSTANTS:

   Constants follow the executable code and precede the copy files and/or macros that contain DSECTs or system equates. Constants are defined in a section followed by a section containing initialized working storage, followed by working storage. Each of these sections is identified by a comment. Wherever possible for a module that is greater than a page, constants and working storage are within the same page in which they are referenced.

4. No program modifies its own instructions during execution.

5. No program uses its own unlabeled instructions as data.

6. REGISTER USAGE:

   For CP, in general

| Register | Use |
|---|---|
| 6 | RCHBLOK, VCHBLOK |
| 7 | RCUBLOK, VCUBLOK |
| 8 | RDEVBLOK, VDEVBLOK |
| 10 | IOBLOK |
| 11 | VMBLOK |
| 12 | Base register for modules called via SVC |
| 13 | SAVEAREA for modules called via SVC |
| 14 | Return linkage for modules called via BALR |
| 15 | Base address for modules called via BALR |

   For Virtual-to-Real address translation:

| Register | Use |
|---|---|
| 1 | Virtual address |
| 2 | Real address |

7. When describing an area of storage in mainline code, a copy file, or a macro, DSECT is issued containing DS instructions.

8. Meaningful names are used instead of self-defining terms, for example 5,X'02',C'I' to represent a quantity (absolute address, 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
```

etc. ...

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
```

9. Avoid absolute relative addressing in branches and data references, (that is, location counter value (*) or symbolic label plus or minus a self-defining term used to form a displacement).

10. When using a single operation to reference multiple values, specify each value referenced, for example:

```
LM  R2,R4,CONT  SET R2=CON1
                SET R3=CON2
                SET R4=CON3
.
.
.
CON1  DC  F'1'
CON2  DC  F'2'
CON3  DC  F'3'
```

11. Do not use PRINT NOGEN or PRINT OFF in source code.

12. MODULE NAMES:

Control Section Names and External References are as follows:

Control Section or Module Name

The first three letters of the name are the assigned component code.

Example: DMK

The next three letters of the Module Name identify the module and must be unique.

Example: DSP

This three-letter, unique module identifier is the label of the TITLE card.

Each entry point or external reference must be prefixed by the six letter unique identifier of the module.

Example: DMKDSPCH

13. TITLE Card:

    DSP TITLE 'DMKDSP VM/370 DISPATCHER VERSION v LEVEL 1'

14. PTF Card Example:

    CP/CMS: PUNCH 'xxxxxxxx APPLIED'

    where xxxxxxxx = APAR Number Response

15. ERROR MESSAGES:

    There should be no insertions into the message at execution time and the length of the message should be resolved by the assembler. If insertions must be made, the message must be assembled as different DC statements, and the insert positions are to be individually labeled.

16. For all RX instructions use a comma (,) to specify the base register when indexing is not being used, that is:

    L R2,AB(,R4)

17. To determine whether you are executing in a virtual machine or in 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 will make up the CP system. All modules following DMKCPE in the list are pageable CP modules. Each 4K page in this area may contain one or more modules. The module grouping is governed by the order in which they appear in the loadlist. An SPB[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.

---

[1]A 12-2-9 multipunch must be in column 1 of an SPB card.

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.

## How To Add a Console Function to CP

You can add your own command to your installation's VM/370. First, code the module to handle the command processing. You should follow the CP coding convention outlined in an earlier section of this book.

Second, you must add an entry for your command in the CP DMKCFM module. DMKCFM has two entry points: one for logged-on users and another for nonlogged-on users. If your command is for logged-on users, be sure its entry is beyond the label COMNBEG1.

To place an entry for your command in the DMKCFM module, insert a line with the following format:

```
| [label] | COMND | commandname,class,min,entrypt[,NCL=1]
```

where:

commandname is a one- to eight-character name.

class is the command privilege class (up to four classes are allowed). 0 is coded for nonlogged-on user commands.

min is the number of characters allowed as the minimum truncation.

entrypt is the entry point of the module you write to process the new command.

NCL=1 is specified only when class is "0".

After you have inserted the above entry in the DMKCFM module, you must reload DMKCFM as a resident module being sure it does not cross a page boundary. You must also load your own module which may or may not be a resident module.

Buffer images are supplied for the UCS (Universal Character Set) buffer, the UCSB (Universal Character Set Buffer), and the FCB (Forms Control Buffer). The VM/370 supplied buffer images are:

UCS - FOR THE 1403 PRINTER

| Name | Meaning |
|---|---|
| AN | Normal AN arrangement |
| HN | Normal HN arrangement |
| PCAN | Preferred character set, AN |
| PCHN | Preferred character set, HN |
| QN | PL/I - 60 graphics |
| QNC | PL/I - 60 graphics |
| RN | FORTRAN, COBOL commercial |
| YN | High speed alphanumeric |
| TN | Text printing 120 graphics |
| PN | PL/I - 60 graphics |
| SN | Text printing 84 graphics |

UCSB - FOR THE 3211 PRINTER

| Name | Meaning |
|---|---|
| A11 | Standard Commercial |
| H11 | Standard Scientific |
| G11 | ASCII |
| P11 | PL1 |
| T11 | Text Printing |

FCB - FOR THE 3211 PRINTER

There is only one name provided for an FCB image.

| Name | Meaning |
|---|---|
| FCB1 | Space 6 lines/inch<br>Length of page 66 lines |

| Line Represented | Channel Skip Specification |
|---|---|
| 1 | 1 |
| 3 | 2 |
| 5 | 3 |
| 7 | 4 |
| 9 | 5 |
| 11 | 6 |
| 13 | 7 |
| 15 | 8 |
| 19 | 10 |
| 21 | 11 |
| 23 | 12 |
| 64 | 9 |

Refer to the following publications for the exact contents of the buffer images:

- IBM 2821 Control Unit Component Description.
- IBM 3211 Printer, 3216 Interchangeable Train Cartridge, and 3811 Printer Control Unit Component Description and Operator's Guide.

If you find that the supplied buffer images do not meet your needs, you can alter a buffer image or create a new buffer image. Be careful not to violate the VM/370 coding conventions if you add a new buffer image; buffer images must not cross page boundaries.

## ADDING NEW PRINT BUFFER IMAGES

In order to add a new print buffer image to VM/370 you must:

1. Provide a buffer image name and 12 byte header for the buffer load.
2. Provide the exact image of the print chain.
3. Provide a means to print the buffer image if VER is specified on the LOADBUF command.
4. Reload the changed CP modules.

Macros are available which make the process of adding buffer images relatively easy.

### UCS BUFFER IMAGES

The UCS buffer contains up to 240 characters and supports the 1403 printer. To add a new UCS buffer image, first code the UCS macro. This creates a 12-byte header for the buffer load which is used by the CP module DMKCSO. The format of the UCS macro is:

```
┌──────────────────────────────────────────────────────────────────────┐
│     │ UCS    │ ucsname                                               │
└──────────────────────────────────────────────────────────────────────┘
```

where:

ucsname is a one- to four-character name which is assigned to the buffer load.

Next, supply the exact print image. The print image is supplied by coding DCs in hexadecimal or character format. The print image may consist of several DCs, the total length of the print image cannot exceed 240 characters.

The UCSCCW macro must immediately follow the print image. This macro creates a CCW string to print the buffer load image when VER is specified by the operator on the LOADBUF command. The format of the UCSCCW macro is:

```
|      | UCSCCW | ucsname[,(print1,print2,...,print12)]                |
```

where:

ucsname is the same as the "ucsname" specified on the UCS macro.

[(print1,...,print12)]
is the line length (or number of characters to be printed by the corresponding CCW) for the verify operation. Each count specified must be between 1 and 132 (the length of the print line on a 1403 printer) and the default line length is 48 characters. Up to 12 print fields may be specified. However, the total number of characters to be printed may not exceed 240.

Finally, insert the macros just coded, UCS and UCSCCW, into the DMKUCS module. This module must be reloaded. DMKUCS is a pageable module (with no executable code) that is called by DMKCSO. DMKUCS must be on a page boundary and cannot exceed a full page in size.

## Examples of New UCS Buffer Images

Example 1: You do not have to specify the line length for verification of the buffer load. Insert the following code in DMKUCS:

```
UCS    EX01
DC     5CL'1234567890A...Z1234567890*/'
UCSCCW EX01
```

The buffer image is 5 representations of a 48 character string containing:

- The alphabetic characters
- The numeric digits, twice
- The special characters: * and /

Since the line length for the print verification is not specified on the UCSCCW macro, it defaults to 48 characters per line for 5 lines.

Example 2: Insert the following code in DMKUCS:

```
UCS    NUM1
DC     24CL'1234567890'
UCSCCW NUM1,(60,60,60,60)
```

The NUM1 print buffer consists of 24 10-character entries. If, after DMKUCS is reloaded, the command

```
LOADBUF  00E  UCS  NUM1  VER
```

is specified, 4 lines of 60 characters (the 10-character string repeated 6 times) are printed to verify the buffer load).

Example 3: The print image can be specified in character or hexadecimal notation, or a combination of the two. The code in DMKUCS to support the preferred character set, AN, is as follows:

```
UCS     PCAN
DC      C'1234567890,-PQR#$@/STUVWXYZ',X'9C'
DC      C'.*1234567890,-JKLMNOABCDEFGHI+.*'
DC      C'1234567890,-PQR&&$%/STUVWXYZ',X'9C'
DC      C'.*1234567890,-JKLMNOABCDEFGHI+.*'
DC      C'1234567890,-PQR#$@/STUVWXYZ',X'9C'
DC      C'.*1234567890,-JKLMNOABCDEFGHI+.*'
DC      C'1234567890,-PQR&&$%/STUVWXYZ',X'9C'
DC      C'.*1234567890,-JKLMNOABCDEFGHI+.*'
UCSCCW PCAN,(60,60,60,60)
```

The DCs are coded in both character and hexadecimal notation. The hexadecimal code for the lozenge (X'9C') follows the character notation on 4 of the DCs. The DCs, when taken in pairs, represent 60 characters. When print verification of a buffer load is requested, 4 lines of 60 characters are printed.

## USCB BUFFER IMAGES

The UCSB buffer contains up to 512 characters and supports the 3211 printer. To add a new UCB buffer image, first code the UCB macro. This macro creates a 12-byte header record for the buffer load which is used by the CP module, DMKCSO. The format of the UCB macro is:

```
|      | UCB     | ucbname                                                  |
```

where:

ucbname is a one- to four-character name which is assigned to the buffer load.

Next, supply the exact print image. The print image is supplied by coding DCs in hexadecimal or character notation. The total length of the print image cannot exceed 512 characters.

The format of the UCB buffer is:

| Position | Contents |
|---|---|
| 1-432 | Print train image. |
| 433-447 | Reserved for IBM use. Must be all zeros. |
| 448-511 | Associative field. See Figure 31 for an explanation of the contents of this field. The associative field is used to check (during print line buffer (PLB) loading) that each character loaded into the PLB for printing also appears in the train image field of the USCB and, therefore, is on the print train. Any character loaded into the PLB without its associated code in the train image field of the USCB is nonprintable, and causes a 'print data check' to be set immediately. The associative field also contains dualing control bits. |
| 512 | Reserved for IBM use. Must be zero. |

| | Bit 0 | | Bit 1 | | Bit 2 | | Bit 3 | |
|---|---|---|---|---|---|---|---|---|
| UCSB Address | Hexa-decimal | Graphic & Control Symbols EBCDIC | Hexa-decimal | Graphic & Control Symbols EBCDIC | Hexa-decimal | Graphic & Control Symbols EBCDIC | Hexa-decimal | Graphic & Control Symbols EBCDIC |
| 448 | 00 | NUL | 40 | SP | 80 | | C0 | |
| 449 | 01 | | 41 | | 81 | a | C1 | A |
| 450 | 02 | | 42 | | 82 | b | C2 | B |
| 451 | 03 | | 43 | | 83 | c | C3 | C |
| 452 | 04 | PF | 44 | | 84 | d | C4 | D |
| 453 | 05 | HT | 45 | | 85 | e | C5 | E |
| 454 | 06 | LC | 46 | | 86 | f | C6 | F |
| 455 | 07 | DEL | 47 | | 87 | g | C7 | G |
| 456 | 08 | | 48 | | 88 | h | C8 | H |
| 457 | 09 | | 49 | | 89 | i | C9 | I |
| 458 | 0A | | 4A | ¢ | 8A | { | CA | |
| 459 | 0B | | 4B | . | 8B | ≤ | CB | |
| 460 | 0C | | 4C | < | 8C | ≤ | CC | ⌐ |
| 461 | 0D | | 4D | ( | 8D | ( | CD | |
| 462 | 0E | | 4E | + | 8E | + | CE | ⌐ |
| 463 | 0F | CU1 | 4F | \| | 8F | | CF | |
| 464 | 10 | | 50 | & | 90 | | D0 | |
| 465 | 11 | | 51 | | 91 | j | D1 | J |
| 466 | 12 | | 52 | | 92 | k | D2 | K |
| 467 | 13 | | 53 | | 93 | l | D3 | L |
| 468 | 14 | RES | 54 | | 94 | m | D4 | M |
| 469 | 15 | NL | 55 | | 95 | n | D5 | N |
| 470 | 16 | BS | 56 | | 96 | o | D6 | O |
| 471 | 17 | IL | 57 | | 97 | p | D7 | P |
| 472 | 18 | | 58 | | 98 | q | D8 | Q |
| 473 | 19 | | 59 | | 99 | r | D9 | R |
| 474 | 1A | CC | 5A | ! | 9A | ^ | DA | |
| 475 | 1B | | 5B | $ | 9B | } | DB | |
| 476 | 1C | | 5C | * | 9C | ⌐ | DC | |
| 477 | 1D | | 5D | ) | 9D | ) | DD | |
| 478 | 1E | | 5E | ; | 9E | ± | DE | |
| 479 | 1F | CU2 | 5F | ¬ | 9F | ■ | DF | |
| 480 | 20 | | 60 | - | A0 | o | E0 | |
| 481 | 21 | | 61 | / | A1 | | E1 | |
| 482 | 22 | | 62 | | A2 | s | E2 | S |
| 483 | 23 | | 63 | | A3 | t | E3 | T |
| 484 | 24 | BYP | 64 | | A4 | u | E4 | U |
| 485 | 25 | LF | 65 | | A5 | v | E5 | V |
| 486 | 26 | EOB | 66 | | A6 | w | E6 | W |
| 487 | 27 | PRE | 67 | | A7 | x | E7 | X |
| 488 | 28 | | 68 | | A8 | y | E8 | Y |
| 489 | 29 | | 69 | | A9 | z | E9 | Z |
| 490 | 2A | SM | 6A | | AA | | EA | |
| 491 | 2B | | 6B | | AB | | EB | |
| 492 | 2C | | 6C | % | AC | | EC | ⌐ |
| 493 | 2D | | 6D | _ | AD | [ | ED | |
| 494 | 2E | | 6E | > | AE | ≥ | EE | |
| 495 | 2F | CU3 | 6F | ? | AF | ● | EF | |
| 496 | 30 | | 70 | | B0 | 0 | F0 | 0 |
| 497 | 31 | | 71 | | B1 | 1 | F1 | 1 |
| 498 | 32 | | 72 | | B2 | 2 | F2 | 2 |
| 499 | 33 | | 73 | | B3 | 3 | F3 | 3 |
| 500 | 34 | PN | 74 | | B4 | 4 | F4 | 4 |
| 501 | 35 | RS | 75 | | B5 | 5 | F5 | 5 |
| 502 | 36 | UC | 76 | | B6 | 6 | F6 | 6 |
| 503 | 37 | EOT | 77 | | B7 | 7 | F7 | 7 |
| 504 | 38 | | 78 | | B8 | 8 | F8 | 8 |
| 505 | 39 | | 79 | | B9 | 9 | F9 | 9 |
| 506 | 3A | | 7A | : | BA | | FA | |
| 507 | 3B | | 7B | # | BB | ⌐ | FB | |
| 508 | 3C | | 7C | @ | BC | ⌐ | FC | |
| 509 | 3D | | 7D | ' | BD | ] | FD | |
| 510 | 3E | | 7E | = | BE | ≠ | FE | |
| 511 | 3F | | 7F | " | BF | — | FF | |

**Figure 31. UCSB Associative Field Chart**

The UCBCCW macro must immediately follow the print image. This macro creates a CCW string to print the buffer load image when the operator specifies VER on the LOADBUF command. The format of the UCBCCW macro is:

```
|      | UCBCCW | ucbname[,(print1,print2,...print12)]                    |
```

where:

ucbname is the same name specified on the corresponding UCB macro.

[(print1,...,print12)]
specifies the line length of each line (up to 12) printed to verify the buffer load. The line length must be between 1 and 150 (the length of a print line on a 3211 printer). The default specification for verification is 48 characters per line for nine lines. The total number of characters to be printed must not exceed the size of the print train image, 432 characters.

Finally, insert the two macros just coded, UCB and UCBCCW, into the DMKUCB module. This module must be reloaded before the new buffer image can be used. DMKUCB is a pageable module (with no executable code) that is called by DMKCSO. DMKUCB must be on a page boundary and cannot exceed a full page in size.

## Examples of USCB Buffer Images

The code for the A11 UCB buffer is as follows:

```
            A11 STANDARD COMMERCIAL 48 GRAPHICS 3211
UCB         A11
DC          9C'1<.+IHGFEDCBA*$-RPQONMLKJ%,&&ZYXWVUTS/@#098765432'
DC          X'000000' 433-435
DC          X'00000000000000000000000001010010' 436-450
DC          X'1010101010101000404040240004010' 451-465
DC          X'1010101010101010004040410000040' 466-480
DC          X'4010101010101010100040400000000' 481-495
DC          X'1010101010101010101000404044448' 496-510
DC          X'0000' 511-512
UCBCCW      A11,(48,48,48,48,48,48,48,48,48)
EJECT
```

Note that the DC specification contains 49 characters and the UCBCCW macro specifies 48 characters. The ampersand (&) must be coded twice in order to be accepted by the assembler. The single quote (') must also be specified twice in order to be accepted.

It would have been acceptable to code the UCBCCW as:

```
UCBCCW A11
```

since the default is what was coded.

## FORMS CONTROL BUFFER

It is possible to have a forms control buffer with both a virtual and real 3211 printer. A virtual 3211 file can be printed on a real 1403; in fact, one way to provide forms control for a 1403 is to define it virtually as a 3211.

There is an FCB macro to support forms control. The format of the FCB macro is:

```
|     | FCB    | fcbname,space,length,(line,channel...),index          |
```

where:

fcbname is the name of the forms control buffer. "fcbname" can be one to four alphameric characters.

space is the number of lines/inch. Valid specifications are 6 or 8. This operand may be omitted, the default is 6 lines/inch. When the space operand is omitted, a comma (,) must be coded. Spacing has no meaning for a virtual printer.

length is the number of print lines per page or carriage tape (1 to 180).

(line,channel...)
shows which print line (line) prints in each channel (1 to 12). The entries can be specified in any order.

index is an index value (from 1 to 31). "index" specifies the print position which is to be the first printed position. (The "index" specification can be overridden with the LOADBUF command).

One standard FCB image is supplied, FCB1. You will find FCB1 in the module DMKFCB. DMKFCB is a pageable module which is called by DMKCSO. It must start on a page boundary and cannot exceed a full page in size. As long as you follow these conventions, you can add additional forms control buffer images to DMKFCB.

Note: The GENERATE EXEC procedure has a facility to reassemble only the DMKFCB module. See the description of the GENERATE EXEC procedure in the VM/370: Planning and System Generation Guide.

Example 1:

If you wanted your printer to print:

- 8 lines/inch
- 60 lines/page
- print line 3 in channel 1
- print line 60 in channel 9
- print line 40 in channel 12
- print position 10 the first print position

you would code the FCB macro (with a name, SPEC) as:

```
FCB SPEC,8,60,(3,1,40,12,60,9),10
```

If you want another forms control buffer, called LONG, to be exactly the same as SPEC (except that only 6 lines print per inch) you could code either of the following:

```
FCB LONG,6,60,(3,1,40,12,60,9),10

FCB LONG,,60,(3,1,40,12,60,9),10
```

Example 2:

You could have your special forms control buffer (SPEC) loaded for either a virtual or real 3211 printer. The LOADVFCB command is for the virtual 3211 and the LOADBUF command is for the real 3211. If INDEX is not specified on these commands, no indexing is done. If INDEX is specified without a value, the value coded in the FCB macro is used and if INDEX is specified with a value, the specified value overrides the value coded in the FCB macro.

If you specify INDEX for the virtual 3211 printer and again for the real 3211 printer, the output is indexed the sum of the two specifications minus 1. For example, the command

```
LOADVFCB 00F FCB SPEC INDEX
```

indexes the virtual print file 10 positions because 10 was specified in the FCB macro for the SPEC forms control buffer. When this file is sent to the real printer, the command

```
LOADBUF 00E FCB SPEC INDEX 20
```

indexes the file an additional 20 positions. The value specified on the command line (20) overrides the value in the FCB macro (10). The output will start printing in print position 29 (10+20-1=29).

# Part 3. Conversational Monitor System (CMS)

Part 3 contains the following information:

- Introduction to CMS
- Interrupt Handling
- Functional Information (How CMS works)

  -- Register usage
  -- DMSNUC structure
  -- Storage structure
  -- Free storage management
  -- SVC handling

- How To Add a Command or EXEC Procedure to CMS
- OS Macro Simulation
- Saving the CMS system
- Batch Monitor
- Auxiliary Directories

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 which will alter the program. This is the most common method used to debug programs that run in CMS.

CMS, operating with the VM/370 Control Program, is a time sharing system suitable for problem solving, program development, and general work. It includes several programming language processors, file manipulation commands, utilities, and debugging aids. Additionally, CMS provides facilities to simplify the operation of other operating systems in a virtual machine environment when controlled from a remote terminal. For example, CMS capabilities are used to create and modify job streams, and to analyze virtual printer output.

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

## THE CMS COMMAND LANGUAGE

The CMS command language offers terminal users a wide range of functions. It supports a variety of programming languages, service functions, file manipulation, program execution control, and general system control. The CMS commands that are useful in debugging are discussed in the "Debugging with CMS" section of "Part 1. Debugging with VM/370." For detailed information on all other CMS commands, refer to the VM/370: CMS Commands and Macro Reference.

Figure 35 describes CMS command processing.

## THE FILE SYSTEM

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

User files in CMS are identified with three designators. The first is filename. The second is a filetype designator which may imply specific file characteristics to the CMS file management routines. The third is a filemode designator 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 32 describes the CMS file system.

## PROGRAM DEVELOPMENT

The Conversational Monitor System includes commands to create and compile source programs, to modify and correct source programs, to build test files, to execute test programs and to debug from the terminal. The commands of CMS are especially useful for OS and 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 Supervisor Call functions such as GETMAIN/FREEMAIN and TIME are simulated. The simulation restrictions concerning what types of OS object programs can be executed under CMS are primarily related to the OS/PCP, MFT, and MVT Indexed Sequential Access Method (ISAM) and the telecommunications access methods, while functions related to multitasking in OS and DOS/VS are ignored by CMS. For more information on macro simulation, see "OS Macro Simulation under CMS" and "DOS/VS Macro Simulation".

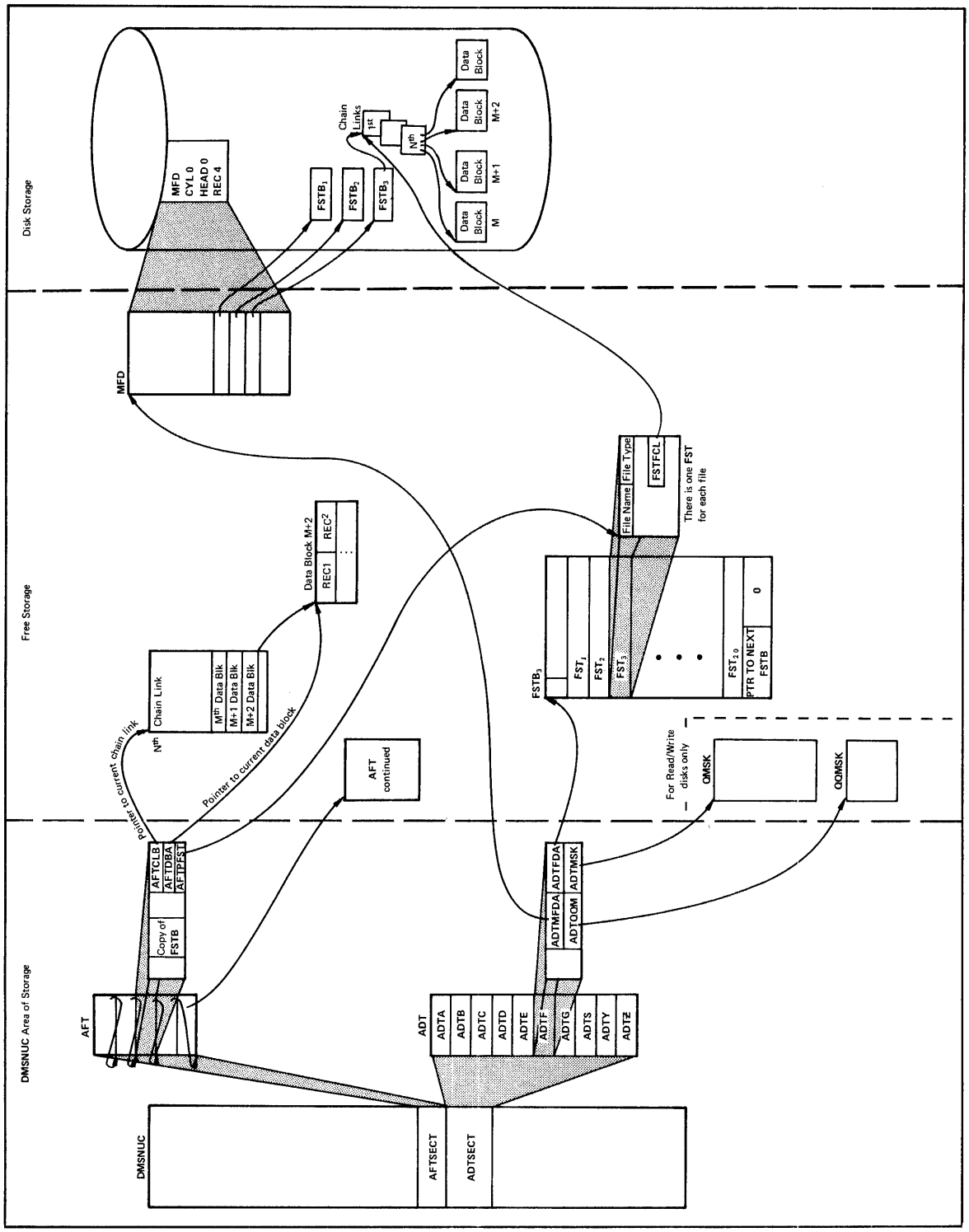



Figure 32. CMS File System

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 one has been set up by the DMSHDS program). If the user-defined SVC table is present, any SVC number (other than 202 or

203) is looked for in that table. If it is found, control is transferred to the routine at the specified address.

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

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

## INPUT/OUTPUT INTERRUPTIONS

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

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

If the wait bit is on, the interruption was probably caused by a nonterminal (synchronous) I/O operation. The program that initiated the operation most likely called the DMSIOW function routine to wait for a particular type of interruption (usually a device end.) In this case, DMSITI checks the pseudo-wait bit in the device table entry for the interrupting device. If this bit is off, the system is waiting for some event other than the interruption from the interrupting device; DMSITI returns to the wait state by loading the saved I/O old PSW. (This PSW has the wait bit on.)

If the pseudo-wait bit is on, the system is waiting for an interruption from that particular device. If this interruption is not the one being waited for, DMSITI loads the saved I/O old PSW. This 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 uppercase, followed by the lowercase (it moves the typeball without printing). Otherwise, control is passed to the DEBUG routine.

## MACHINE CHECK INTERRUPTIONS

Hard machine check interruptions on the real 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.

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. Figure 33 describes the devices supported by CMS.

## STRUCTURE OF CMS STORAGE

Figure 34 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.

| 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 33. Devices Supported by a CMS Virtual Machine

- Transient Program Area (X'0E000' to X'10000'). Since it is not essential to keep all nucleus functions resident in storage all the time, some of them are made "transient." This means that when they are needed, they are loaded from the disk into the Transient Program Area. Such programs may not be longer than two pages, because that is the size of the Transient Area. (A page is 4096 bytes of virtual storage.) All transient routines must be serially reusable since they are not read in each time they are needed.

- 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, which are required by the CMS Loader. These tables indicate which modules are currently loaded in the User Program Area (and the Transient Program Area after a LOAD COMMAND). The size of the Loader Tables can be varied by the SET LDRTBLS command. However, to successfully change the size of the Loader Tables, the SET LDRTBLS command must be issued immediately after IPL.

## FREE STORAGE MANAGEMENT

Free storage can be allocated by issuing 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.

### GETMAIN FREE STORAGE MANAGEMENT

All GETMAIN storage is allocated in the user program area, starting after the end of the user's actual program. Allocation begins at the location pointed to by the NUCON pointer MAINSTRT. The location MAINHIGH in NUCON is the "high-extend" pointer for GETMAIN storage.

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 ││                             │
│          │          │ └         └     ┘┘                             │
└──────────────────────────────────────────────────────────────────────┘
```

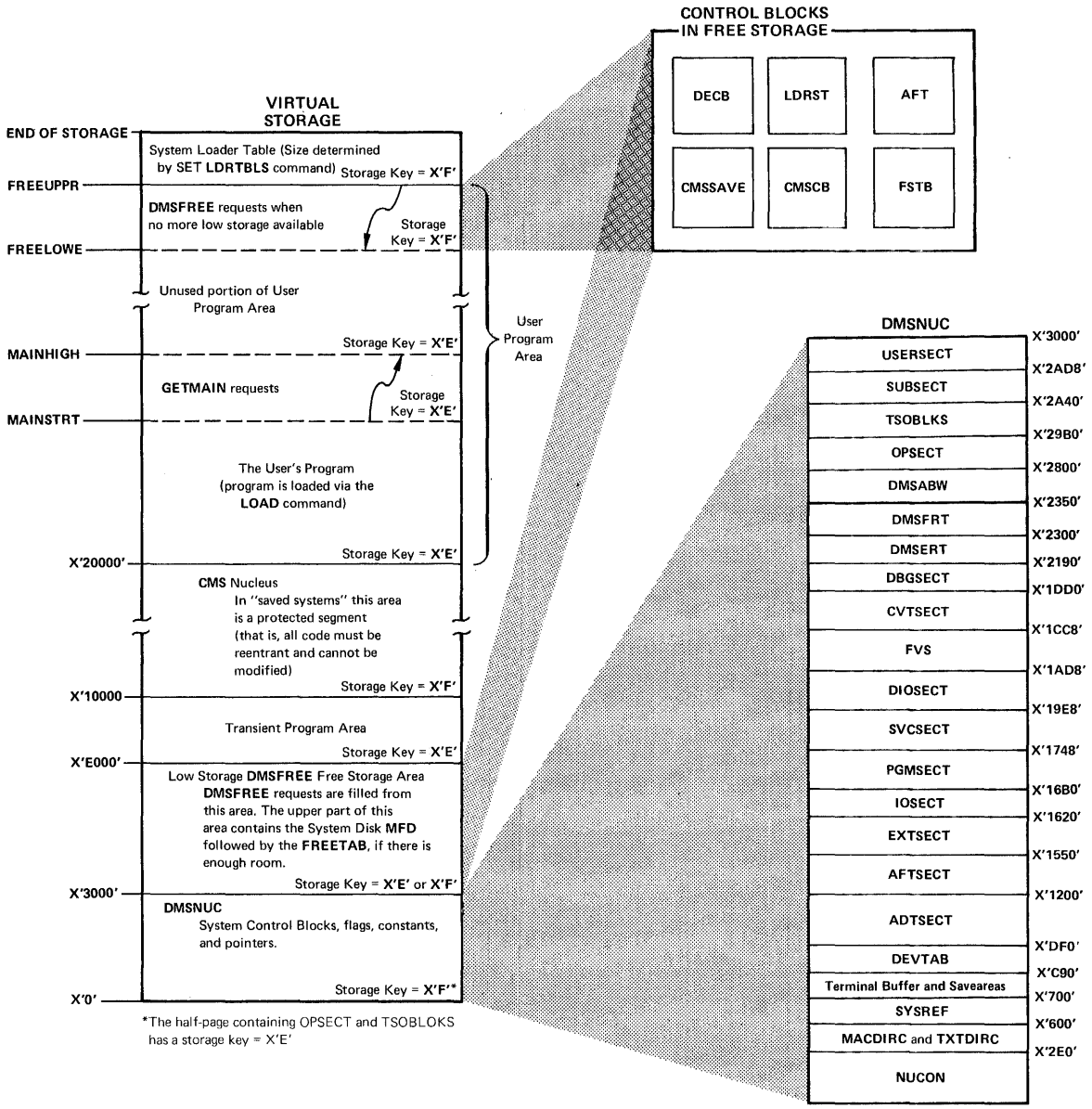



Figure 34. CMS Storage Map

where:

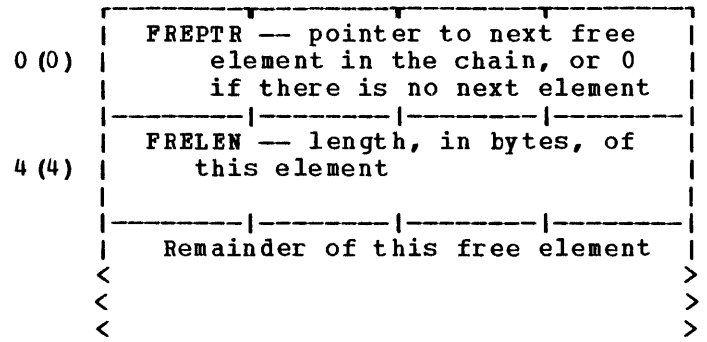
TYPCALL=[SVC / BALR] indicates how control is passed to DMSSTG, the routine that processes the STRINIT macro. Since DMSSTG is a nucleus-resident routine, other nucleus-resident routines can branch directly to it ( TYPCALL=BALR) while routines that are not nucleus-resident must use linkage SVC (TYPCALL=SVC). If no operands are specified the default is 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, then GETMAIN will take an error exit, indicating that insufficient storage is available to satisfy the request.

The area between MAINSTRT and MAINHIGH may contain blocks of storage which are not allocated, and which 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 34 for a description of CMS virtual storage usage.

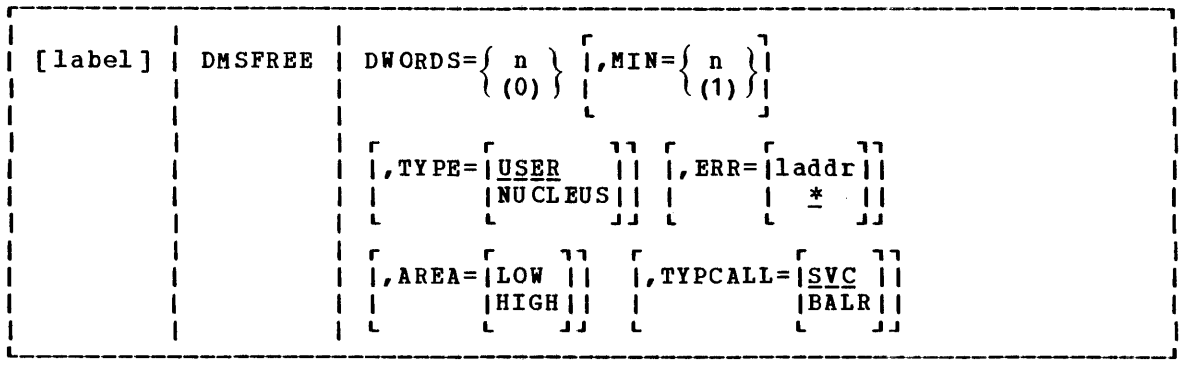
The format of an element on the GETMAIN free element chain is as follows:




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:




where:

label — is any valid assembler language label.

DWORDS={n / (0)} — is the number of doublewords of free storage requested. DWORDS=n specifies the number of doublewords directly and DWORDS=(0) indicates that register 0 contains the number of doublewords requested.

MIN={n / (1)} — indicates a variable request for free storage. If the exact number of doublewords indicated by the DWORDS operand is not available, then 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 requestor 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 an LA (load address) instruction. The error return is taken if there is a macro coding error or if there is not enough free storage available to fill the request. If * 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 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. Since 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 which 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 will be 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, then DMSFREE will take an error exit, indicating that insufficient storage is available to satisfy the request. Figure 34 shows the relationship of these storage areas.

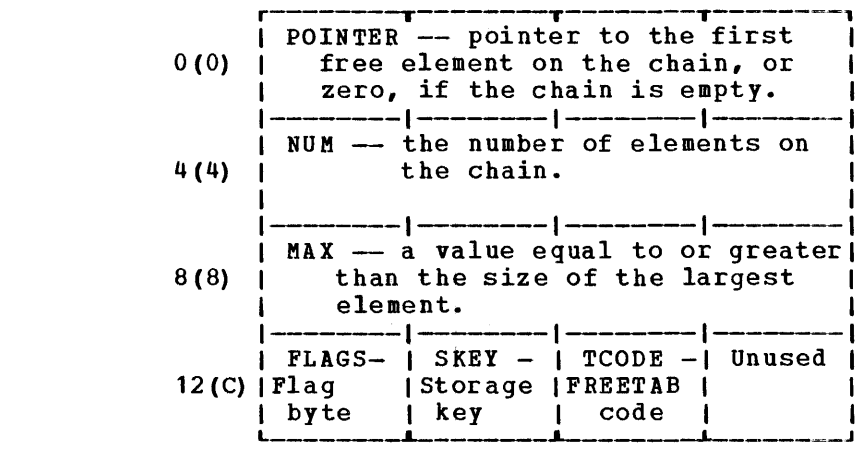
The FREETAB free storage table is kept in free storage, usually in low-storage, just below the Master File Directory for the System Disk (S-disk). However, the FREETAB may be located at the top of the user program area. This table contains one byte for each page of virtual storage. Each such byte contains 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:




where:

POINTER points to the first element on this chain of free elements. If there are no elements on this free chain, then the POINTER field contains all zeros.

NUM contains the number of elements on this chain of free elements. If there are no elements on this free chain, then this field contains all zeros.

MAX is used to avoid searches which will fail. It contains a number not exceeding the size, in bytes, of the largest element on the free chain. Thus, a search for an element of a given size will 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 will be necessary in the following circumstances:

- If one of the two high-storage chains contains a 4K page which 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 one-byte storage key assigned to storage on this chain.

TCODE contains the one-byte FREETAB table code for storage on this chain.

## Allocating User Free Storage

When DMSFREE with TYPE=USER (the default) is called, one or more of the following steps are taken in an attempt to satisfy the request. As soon as one of the following steps succeeds, then 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.

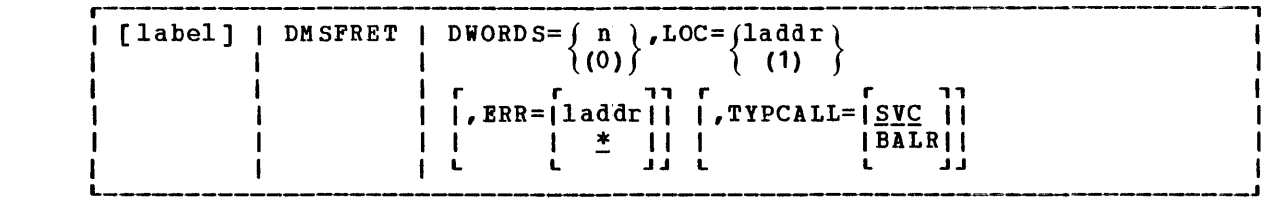
## Allocating Nucleus Free Storage

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:




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 referred to in an LA (load address) 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 an LA (Load Address) 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. Since DMSFRET is a nucleus-resident routine, other nucleus-resident routines can branch directly to it (TYPCALL=BALR) while routines that are not nucleus-resident must use SVC linkage (TYPCALL=SVC).

When DMSFRET is called, the block being released is placed on the appropriate chain. At that point, the final update operation is performed, if necessary, to advance FREELOWE, or to move pages from the nucleus chain to the corresponding user chain.

Similar update operations will be 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 will cause 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 DMSFRES macro instruction is used by the system to request certain free storage management services.

The format of the DMSFRES macro is:

```
┌──────────────────────────────────────────────────────────────────────┐
| [label] | DMSFRES | INIT1  ┌          ┌    ┐┐                        |
|         |         | INIT2  |,TYPCALL=|SVC ||                        |
|         |         | CHECK  |          |BALR||                        |
|         |         | 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 INIT1 is invoked, no free storage requests may be made. After INIT1 has been invoked, free storage requests may be made, but these are subject to the following restraints until the second free storage management initialization routine has been invoked:

- All requests for USER type storage are changed to requests for NUCLEUS type storage.
- Error checking is limited before initialization is complete. In particular, it is sometimes possible to release a block 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:

- Releases all storage which has been allocated in the high-storage area.
- Allocates the FREETAB free storage table. This table contains one byte for each 4K page of virtual storage, and so cannot be allocated until the size of virtual storage is known.
- The FREETAB table is initialized, and all storage protection keys are initialized.
- 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 will take much longer to be completed than before.

CKOFF turns off the flag which was turned on by the CKON option.

UREC is used by DMSABN during the ABEND recovery process to release all user storage.

CALOC is used by DMSABN after the ABEND recovery process has been completed. It invokes a routine which returns, in register 0, the number of doublewords of free storage which have been allocated. This number is used by DMSABN 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.

| Code | Error |
|---|---|
| 1 | (DMSFREE) Insufficient storage space is available to satisfy the request for free storage. In the case of a variable request, even the minimum request could not be satisfied. |
| 2 | (DMSFREE or DMSFRET) User storage pointers destroyed. |
| 3 | (DMSFREE, DMSFRET, or DMSFRES) Nucleus storage pointers destroyed. |
| 4 | (DMSFREE) An invalid size was requested. This error 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:<br>• The block does not lie entirely inside either the low-storage free storage area or the User Program Area between FREELOWE and FREEUPPR.<br>• The block crosses a page boundary which separates a page allocated for USER storage from a page allocated for NUCLEUS type storage.<br>• The block overlaps another block already on the free storage chain. |
| 7 | (DMSFRET) The address given for the block being released is not doubleword aligned. |
| 8 | (DMSFRES) An invalid request code was passed to the DMSFRES routine. Since all request codes are generated by the DMSFRES macro, this error code should never appear. |
| 9 | (DMSFREE, DMSFRET, or DMSFRES) Unexpected and unexplained error in the free storage management routine. |

## CMS HANDLING OF PSW KEYS

The purpose of the CMS Nucleus Protection scheme is to protect 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', since they have a constant need to modify nucleus pointers and storage. The nucleus routines called by the GET, PUT, READ, and WRITE macros run with a user PSW key of X'E', to increase efficiency.

Two macros 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 wish to enter a user-supplied exit routine; the exit routine is entered with the PSW key of the last user routine on the SVC system save area stack.

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 which 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 35 for a description of this search for a command name.

## USER AND TRANSIENT PROGRAM AREAS

Two areas can hold programs which are loaded from disk. These are called the User Program Area and the Transient Program Area. (See Figure 34 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. Since 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 which 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 which 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 interrupts.

The Transient Program Area is also used to handle 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 which 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 interrupts but starts programs running in the Transient Program Area disabled for all interrupts. The individual program may have to use the SSM (Set System Mask) instruction to change the current status of its system mask.

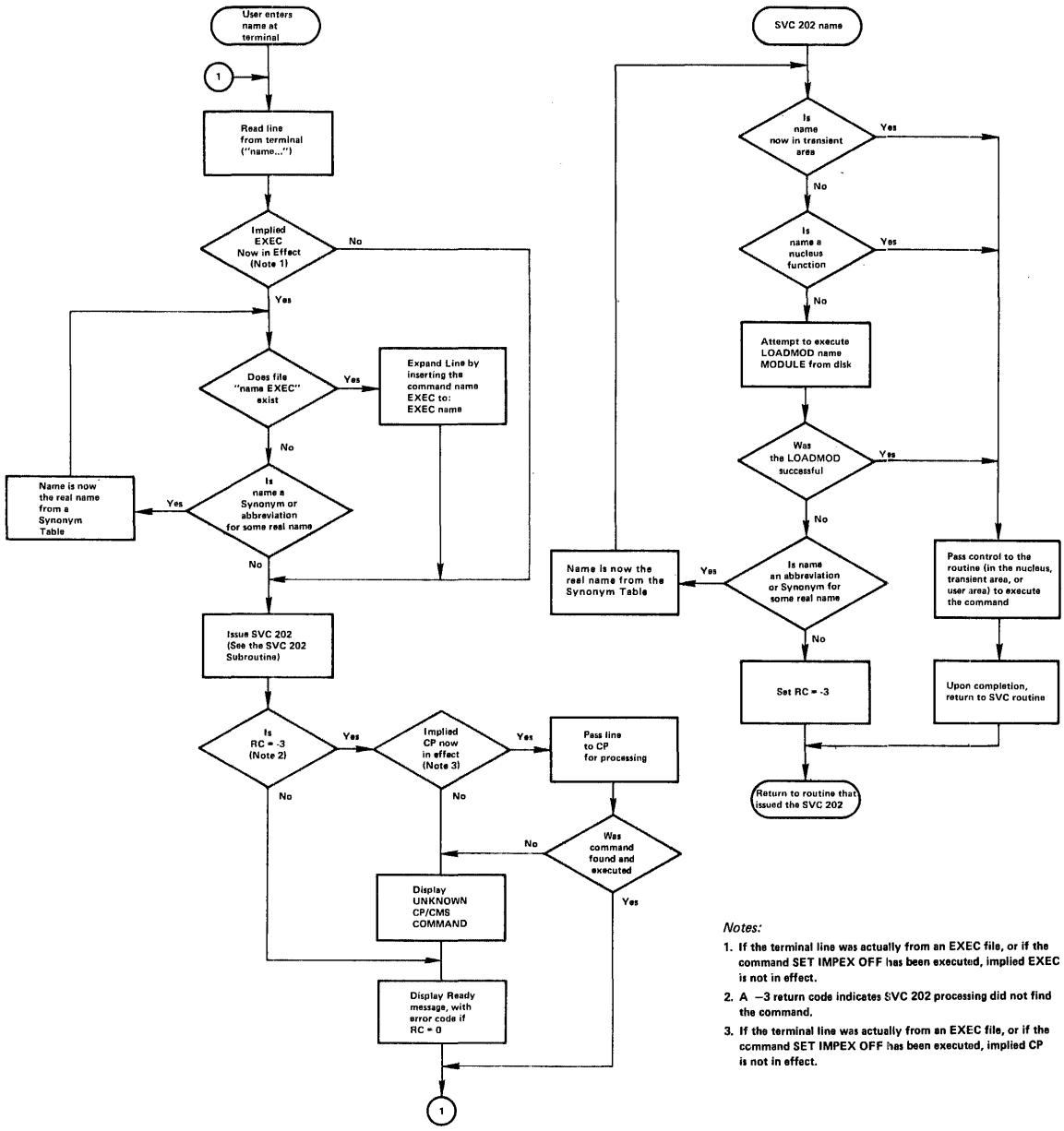



Figure 35. CMS Command (and Request) Processing

CALLED ROUTINE START-UP TABLE

Figures 36 and 37 show how the PSW and registers are set up when the called routine is entered.

| "Called" Type | System Mask | Storage Key | Problem Bit |
|---|---|---|---|
| SVC 202 or 203 - Nucleus resident | Disabled | System | Off |
| SVC 202 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 36. 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 | Unpre-dictable | 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 37. Register Contents When Called Routine Starts

RETURNING TO THE CALLING ROUTINE

When the called routine finishes processing, control is returned to DMSITS, which in turn returns control to the calling routine.

Return Location

The return is accomplished by loading the original SVC old PSW (which was saved at the time DMSITS was first entered), after possibly modifying the address field. The address field modification depends

upon the type of SVC call, and 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 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. Since SVC calls can be nested, there can be several of these save areas at one time. The System Save Area is allocated in protected free storage.

The User Save Area contains 12 doublewords (24 words), allocated in unprotected free storage. DMSITS does not use this area at all, but simply passes a pointer to this area (via register 13.) The called routine can use this area as a temporary work area, or as a register save area. There is one User Save Area for each System Save Area. The 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 Blocks Logic. The most important fields, and their uses, are as follows:

CALLER (Fullword) The address of the SVC instruction which 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.

You can create a module or EXEC procedure which executes in the user area and resides on disk. To execute such a command or EXEC procedure, you only have to enter the filename from the terminal. However, be aware of the CMS search order for terminal input. Once a match is found, the search stops. The search order is:

1. EXEC file on any currently accessed disk.
2. Valid abbreviation for an EXEC file on any currently accessed disk.
3. Nucleus resident or transient area command.
4. Command on any currently accessed disk.
5. Valid abbreviation or synonym for nucleus resident or transient area command.
6. Valid abbreviation for disk resident command.

For example, if you create an EXEC file with the same name as a disk resident command, the CMS search will always find the EXEC file first. Thus, the disk resident command will never get executed.

CMS has a function table containing the names of CMS functions. CMS reserves the following names, all entries in the CMS FUNCTAB (found in DMSFNC), for its own use:

```
ATTN          | DMSSMNAT        RETURN
CARDPH        | DMSVSR          START
CARDRD          ERASE         STATE
CMSTIME         EXEC          STATEW
CONREAD       | FETCH           SUBSET
CONWAIT         FINIS         SVCFREE
CP              GENMOD        SVCFRET
DEBUG           INCLUDE       TAPEIO
DESBUF          LOAD          TRAP
DMSCIOSI        LOADMOD       TYPLIN
DMSERR          POINT         WAIT
DMSLADAD        PRINTIO       WAITRD
DMSPIOCC        PRINTR        WRBUF
DMSPIOSI        RDBUF
```

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 38 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 38 and the "Simulation Notes", then read "Access Method Support" to see how CMS handles these access methods.

Since CMS does not simulate the indexed sequential access method (ISAM), no OS program 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 used as input to an OS program running under CMS.

However, an OS sequential or partitioned data set that resides on an OS disk can be written or updated only by an OS program running in a real OS machine.

CMS can execute programs that read and write VSAM files from OS programs written in the VS BASIC, COBOL, 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 38. 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 routines, 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 options of GETMAIN are supported except SP and HIARCHY, which are ignored by CMS, and LC and LV, which will result in abnormal termination is used. GETMAIN gets blocks of free storage. |
| FREEMAIN-SVC5 | All options of FREEMAIN are supported except SP, which is ignored by CMS, and L, which will result in abnormal termination if used. FREEMAIN frees blocks of storage acquired by GETMAIN. |
| LINK-SVC6 | The DCB and HIARCHY options are ignored by CMS. All other options of LINK are supported. LINK loads the specified program into storage (if necessary) and passes control to the specified entry point. |
| XCTL-SVC7 | The DCB and HIARCHY options are ignored by CMS. All other options of XCTL are supported. XCTL loads the specified program into storage (if necessary) and passes control to the specified entry point. |
| LOAD-SVC8 | The DCB and HIARCHY options are ignored by CMS. All other options of LOAD are supported. LOAD loads the specified program into storage (if necessary) and returns the address of the specified entry point in |

| Macro-SVC No. | Differences in Implementation |
|---|---|
| | 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 will attempt 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 except SP and HIARCHY, which are ignored by CMS. |
| 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. |

| Macro-SVC No. | Differences in Implementation |
|---|---|
| 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.<br><br>Since CMS is not a multitasking system, a phase requested by the ATTACH macro must return to CMS. |
| CHAP-SVC44 | The CHAP routine in CMS is a NOP. It returns control to the user. |
| TTIMER-SVC46 | All the options of TTIMER are supported. |
| STIMER-SVC47 | All options of STIMER are supported except for TASK and WAIT. The TASK option is treated as if the REAL option had been specified, and the WAIT option is treated as a NOP; it returns control to the user. |
| 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 cptions of FREEDBUF are supported. FREEDBUF returns a buffer to the buffer pool assigned to the specified DCB. |

| Macro-SVC No. | Differences in Implementation |
|---|---|
| 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: |

| 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 | - |

## 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 those portions of the OS control blocks that are needed by the OS simulation routines to support a program during execution. Most of the simulated supervisory OS control blocks are contained in the following two CMS control blocks:

CMSCVT
simulates the Communication Vector Table. Location 16 contains the address of the CVT control section.

CMSCB
is allocated from system free storage whenever a FILEDEF command or an OPEN (SVC19) is issued for a data set. The CMS Control Block consists of a File Control Block (FCB) for the data file, and partial simulation of the Job File Control Block (JFCB), Input/Output Block (IOB), and Data Extent Block (DEB).

The Data Control Block (DCB) and the Data Event Control Block (DECB) are used by the access method simulation routines of CMS.

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 dataset)
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.

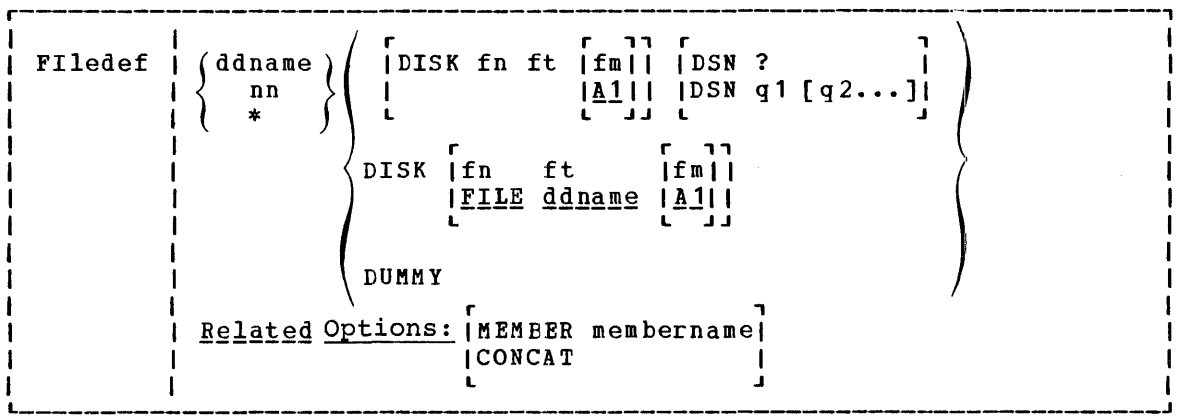
The format of the ACCESS command is:

```
ACCESS cuu mode[/ext]
```

You must not specify options or file identification when accessing an OS or DOS disk.

## The FILEDEF Command

You then 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 VM/370: CMS Command and Macro Reference for a complete description of the CMS ACCESS, FILEDEF, LISTDS, MOVEFILE, QUERY, RELEASE, and STATE commands.

For restrictions on reading OS data sets and 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 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.

CMS supports interactive program development for DOS/VS Release 31 (or later). 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

This environment is entered each time the CMS SET DOS ON command is issued; VSAM functions are only available in CMS/DOS if the SET DOS ON (VSAM) command is issued. In the CMS/DOS environment, CMS supports many DOS/VS facilities, but does not support OS simulation. 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 access method (SAM) and virtual storage access method (VSAM), 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.

## HARDWARE DEVICES SUPPORTED

CMS/DOS routines can read real DOS disks containing DOS data files and DOS private and system libraries. This read support is limited to the following disks supported by DOS/VS:

- 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 virtual 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

If a CMS/DOS command writes files on a CMS disk, that CMS disk can be a 3350 in native mode. For example, if a CMS/DOS user has a 3350 disk (native mode) accessed as his A-disk, he can execute the RSERV command, directing the output to disk. The output is written to the 3350 disk successfully because CMS supports 3350 in native mode.

Also, under CMS/DOS you can write VSAM data sets. VSAM data sets can only be written to disks that are supported by DOS/VS.

The following devices, which are supported by DOS/VS, are not supported by CMS/DOS:

- Card Readers: 1442, 2560P, 2560S, 2596, 3504, 5425P, and 5425S
- Printers: 2560P, 2560S, 3203, 3525, 5203, 5425P, and 5425S
- Disks: 2311

Also, CMS uses the CP spooling facilities and does not support dedicated unit record devices. Each CMS virtual machine supports only one virtual console, one reader, one punch, one printer, four tapes, and ten disks. Programs that are executed in CMS/DOS are limited to the number of devices supported by CMS.

## CMS SUPPORT OF DOS/VS FUNCTIONS

In addition to the CMS SET command used to invoke the CMS/DOS environment, there are a number of CMS/DOS commands and CMS commands with special CMS/DOS operands that provide CMS support of the following DOS/VS functions:

- Assignment of logical units to particular physical devices.
- Associating DOS files with particular logical units.
- DOS/VS Librarian Services.
- Compilation and testing of DOS/VS COBOL and DOS PL/I programs.
- Execution of DOS/VS COBOL and DOS PL/I programs.

Figure 39 summarizes these new commands and the new operands for existing commands. A detailed description and command format can be found in the VM/370: CMS Command and Macro Reference.

| Command | Operand | Comments |
|---|---|---|
| ASSGN | | Executable only in the CMS/DOS environment. Assigns CMS/DOS system or programmer logical units to a virtual device. |
| DLBL | | Defines a DOS or VSAM ddname and relates the ddname to a disk file. |
| DOSLIB | | Deletes, compacts, or lists information about the phases in a CMS/DOS phase library. |
| DOSLKED | | Executable only in the CMS/DOS environment. Link-edits CMS text file, or object modules from a DOS/VS relocatable library, and places them in executable forms in a CMS/DOS phase library. |
| DOSPLI | | Executable only in the CMS/DOS environment. Compiles DOS PL/I source programs. |
| DSERV | | Executable only in the CMS/DOS environment. Displays information about DOS/VS core image, relocatable, source statement, procedure and/or transient directories. |
| ESERV | | Executable only in the CMS/DOS environment. Displays, updates, punches, or prints edited (E sublibrary) DOS/VS source statement books. |
| FCOBOL | | Executable only in the CMS/DOS environment. Compiles DOS/VS COBOL source programs. |
| FETCH | | Executable only in the CMS/DOS environment. Fetches a CMS/DOS executable phase. |
| GENMOD | {OS / DOS / ALL} | Specifies the type of macro support needed to execute a module. The ALL operand is intended for CMS internal use. |
| GLOBAL | DOSLIB | The GLOBAL command can now specify CMS/DOS phase libraries, as well as text and macro libraries. |
| LISTIO | | Executable only in the CMS/DOS environment. Display information about CMS/DOS system and programmer logical units. |
| LOADMOD | | LOADMOD checks that a module generated to execute in a specific macro simulation environment (CMS/DOS or CMS) is in the correct environment. |

Figure 39. Summary of Changes to CMS Commands to Support CMS/DOS (Part 1 of 2)

| Command | Operand | Comments |
|---|---|---|
| OPTION | | Executable only in the CMS/DOS environment. Sets compiler options for DOS/VS COBOL. |
| PSERV | | Executable only in the CMS/DOS environment. Copies and displays procedures in the DOS/VS procedure libraries and/or spools the procedures to the CMS virtual printer and/or punch. |
| QUERY | UPSI | Executable only in the CMS/DOS environment. Displays current setting of CMS/DOS UPSI byte. |
| | OPTION | Executable only in the CMS/DOS environment. Displays CMS/DOS compiler options. |
| | DOS | Displays the current status (active or not active) of CMS/DOS. |
| | DOSLIB | Displays the names of all CMS/DOS phase libraries currently being searched for executable phases. |
| | LIBRARY | Displays the names of all CMS/DOS phase libraries to be searched, in addition to the text and macro libraries. |
| RSERV | | Executable only in the CMS/DOS environment. Copies and/or displays modules in a DOS/VS relocatable library. Output can also be directed to the virtual printer or punch. |
| SET | DOS {ON[fm] / [(VSAM] / OFF} | Makes the CMS/DOS environment active or not active. |
| | UPSI | Executable only in the CMS/DOS environment. Sets the CMS/DOS UPSI byte. |
| SSERV | | Executable only in the CMS/DOS environment. Copies or displays books from the DOS/VS source statement library. Output can also be directed to the virtual printer or punch. |

Figure 39. Summary of Changes to CMS Commands to Support CMS/DOS (Part 2 of 2)

## LOGICAL UNIT ASSIGNMENT

The DOS/VS supervisor contains two tables with which it keeps track of the assignment of logical units to physical devices. These are the logical Unit BLock (LUB) table and the Physical Unit Block (PUB) table. An entry in the LUB table for a particular logical unit, such as SYSRDR, would contain a pointer to a PUB table entry which contains the address of real reader, X'00C'.

On a real DOS/VS machine, logical unit assignments are made dynamically via the ASSGN job statement or the ASSGN operator command.

When using CMS/DOS, the CMS ASSGN command performs a similar function. The ASSGN command in CMS/DOS assigns (or unassigns) a system or programmer logical unit to (or from) a virtual I/O device. If a disk is being assigned to a logical unit, the disk must have been previously accessed via the ACCESS command. As in DOS/VS, you are not allowed to assign the system residence volume via the ASSGN command.

SYSLOG is the default value assigned to the terminal when SET DOS ON is issued.

The valid system logical units that can be assigned are:

| | | |
|---|---|---|
| SYSRDR | SYSLOG | SYSRLB |
| SYSIPT | SYSIN | SYSCAT |
| SYSPCH | SYSOUT | SYSCLB |
| SYSLST | SYSSLB | |

The following DOS/VS system logical units cannot be assigned:

| | | |
|---|---|---|
| SYSRES | SYSLNK | SYSVIS |
| SYSUSE | SYSREC | |

An error message is issued and the command terminated if any of these last five system logical units are specified in the ASSGN command. If SYSIN is specified, both the SYSIPT and SYSRDR LUB and PUB entries are filled in. If SYSOUT is specified, both the SYSLST and SYSPCH LUB and PUB entries are filled in.

If you wish to use DOS/VS private relocatable, core image or source statement libraries, you must assign SYSRLB, SYSCLB or SYSSLB, respectively.

You can assign programmer units SYS000 through SYS241 with the ASSGN command. This deviates from DOS/VS, where the number of programmer logical units varies according to the number of partitions.

ASSGN creates a DOS Logical Unit Block (LUB) and Physical Unit Block (PUB) entry if the device is unassigned or alters the existing LUB/PUB relationship if the device is already assigned. ASSGN fills in a one-byte index in the LUB which points to the proper PUB entry. This PUB entry contains the channel, unit, and device type information.

When a system or programmer logical unit is assigned to READER, PUNCH, or PRINTER, the reference is to a spooled unit record device. Card reader and terminal I/O data must not be blocked.

The ASSGN command is also used to ignore (IGN) or unassign (UA) a logical unit. An I/O operation for a logical unit that is in IGN status is effectively a NO-OP. When a logical unit is unassigned, its pointer to the PUB table is removed.

## Compiler Input/Output Assignments

The compilers supported by CMS/DOS expect input/output to be assigned to the following devices:

- SYSIN/SYSIPT must be assigned to the device where the input source file resides. Valid device types are reader, tape or disk.
- The user should assign the following logical units to any of the indicated device types:

SYSPCH to tape, punch, disk, or IGN
SYSLST to tape, printer, disk, or IGN
SYSLOG to terminal
SYS001, SYS002, and SYS006 to disk.
SYS003-SYS005 to tape or disk.

The maximum number of work files is six for DOS/VS COBOL Compiler (FCOBOL) and two for DOS PL/I Optimizing Compiler (DOSPLI).

You must assign SYSIN/SYSIPT. If it is unassigned at compilation time, an error message is issued and the FCOBOL or DOSPLI command is terminated.

If SYSPCH or SYSLST are unassigned at compilation time, the FCOBOL or DOSPLI EXEC file directs output to the disk where SYSIN resides if SYSIN is assigned to a read/write CMS disk. Otherwise, output is directed to the CMS read/write disk with the most read/write space. If SYSLOG is unassigned, it is assigned to the terminal. If SYS001 through SYSnnn are unassigned, output is directed to the CMS disk with the most read/write space.

## Interrogating I/O Assignments

The current I/O assignments may be displayed on the terminal by entering the CMS/DOS LISTIO command. You can selectively display the system and/or programmer logical units as a group or as a specific unit. With the EXEC option of the LISTIO command you can create a disk file containing the list of assignments.

## DOS/VS SUPERVISOR AND I/O MACROS SUPPORTED BY CMS/DOS

CMS/DOS supports the DOS/VS Supervisor macros and the SAM and VSAM I/O macros to the extent necessary to execute the DOS/VS COBOL Compiler and the DOS PL/I Optimizing Compiler under CMS/DOS. CMS/DOS supports Releases 31 and 32 of the DOS/VS Supervisor macros described in the publication DOS/VS Supervisor and I/O Macros, Order No. GC33-5373.

Since CMS is a single-user system executing in a virtual machine with virtual storage, DOS/VS operations, such as multitasking, that cannot be simulated in CMS are ignored.

The following information deals with the type of support that CMS/DOS provides in the simulation of DOS Supervisor and Sequential Access Method I/O macros. For a discussion of VSAM macros, see the section "CMS Support for OS and DOS VSAM Functions."

### SUPERVISOR MACROS

CMS/DOS supports physical IOCS macros and control program function macros for DOS/VS. Figure 40 lists the physical IOCS macros and describes their support and Figure 41 lists the control program function macros and their support.

| Macro | Support |
|---|---|
| CCB (command control block) | The CCB is generated. |
| EXCB (execute channel program) | The REAL operand is not supported; all other operands are supported. |
| WAIT | Supported. Issued whenever your program requires an I/O operation (started by an EXCP macro) to be completed before execution of program continues. |
| SECTVAL (sector value) | Supported for VSAM. See "CMS Support for OS and DOS VSAM Functions." |
| DTFPH | LBRET3 is not supported, labels cannot be rewritten in CMS/DOS. |
| OPEN/OPENR | Supported. Activates a data file. |
| LBRET (label processing return) | Not supported. |
| FEOV (forced end-of-volume) | Not supported. |
| SEOV (system end-of-volume) | Not supported. |
| CLOSE/CLOSER | Supported. Deactivates a data file. |

Figure 40. Physical IOCS Macros Supported by CMS/DOS

The following information deals with the type of support that CMS/DOS provides in the simulation of DOS Supervisor and Sequential Access Method I/O macros. for VSAM (Virtual Storage Method) and AMS (Access Method Services)".

| Macro | SVC No. | Support |
|---|---|---|
| Program Loading Macros | | |
| FETCH | | SYS=YES or NO operand is ignored. |
| | 02 | Reads a logical transient into storage and passes control to an entry point. |
| | 01 | Reads any phase into storage and passes control to an entry point. |
| GENL | -- | Generates a directory list with a 34-byte entry for each of the specified phases. |
| LOAD | | SYS=YES or NO operand is ignored. |
| | 04 | Reads any phase into storage and returns control to the calling phase. |
| Virtual Storage Macros | | |
| PFIX | 67 | No operation performed. |
| PFREE | 68 | No operation performed. |
| RELPAG | 85 | No operation performed. |
| FCEPGOUT | 86 | No operation performed. |
| PAGEIN | 87 | No operation performed. |
| RUNMODE | 66 | Returns code indicating program is running in virtual mode. |
| SETPFA | 71 | No operation performed. |
| VIRTAD | 70 | Not supported. Execution terminates with error message. |
| REALAD | 69 | Not supported. Execution terminates with error message. |
| GETVIS | 61 | Supported for VSAM. See note. |
| FREEVIS | 62 | Supported for VSAM. See note. |
| Program Communication Macros | | |
| COMRG | 33 | Returns address of background partition's communication region. |
| MVCOM | 05 | Modifies specified bytes within bytes 12-23 of the partition communication region. |
| Releasing Macros | | |
| RELEASE | 64 | Supported for VSAM. See note. |
| Time of Day Macro | | |
| GETIME | 34 | Gets time of day. The GMT operand is not supported. |

Note: VSAM macros are discussed in the section "CMS Support of OS and DOS VSAM Functions."

Figure 41. DOS/VS Macros Supported Under CMS (Part 1 of 3)

| Macro | SVC No. | Support |
|---|---|---|
| Interval Timer and Exit Macros | | |
| SETIME | | No operation performed. |
| | 10 | |
| | 24 | |
| STXIT (PC) | 16 | Provides/terminates supervisor linkage to user's PC routines. Under CMS/DOS, if a program check occurs in a simulated transient routine, a check is made to determine if linkage has been established to an AB routine. If it has, control is passed to the AB routine. If not, the program is canceled. If a program check occurs in a program other than a simulated transient, and if linkage has been established to a PC routine, control is passed there. If no PC routine is available, a check is made to see if linkage has been established to an AB routine. If so, control is passed to the AB routine. If no PC or AB routine is available, the program is canceled. |
| (IT) | 18 | No operation performed. |
| (OC) | 20 | No operation performed. |
| (AB) | 37 | Provides/terminates supervisor linkage to user's AB routine for abnormal termination of the routine. Many of the DOS/VS abnormal termination codes are not meaningful under CMS/DOS. Control is given to an abnormal termination routine on the following selected hexadecimal codes: 1A, 20, 21, 22, 25, 26, 27, 2B. |
| EXIT (PC) | 17 | Return from user's PC routine. |
| (IT) | 19 | Not supported. Execution terminates with error message. |
| (OC) | 21 | Not supported. Execution terminates with error message. |
| TECB | — | TECB control block generated. However, CMS/DOS does not support the use of the Timer Event Control Block. |
| TTIMER | 52 | Zero seconds are returned in register 0 as the time remaining in the interval. |
| WAIT | 07 | Wait for I/O completion. |
| WAITM | 29 | Not supported. Execution terminates with error message. |

Figure 41. DOS/VS Macros Supported Under CMS (Part 2 of 3)

| Macro | SVC No. | Support |
|---|---|---|
| PDUMP | -- | Provides hexadecimal dump of general registers and the virtual storage area contained between two addresses. Processing continues with the next instruction. CMS/DOS uses CP DUMP command to direct dump to the printer. |
| DUMP | -- | Provides hexadecimal dump of the partition and general registers. CMS/DOS uses CP DUMP command to direct dump to the printer. The routine then terminates the invoking program. |
| JDUMP | -- | Same as for DUMP. |
| CANCEL | 06 | Terminates processing. |
| EOJ | 14 | Processing terminates normally. |
| CHKPT | -- | Not supported. Execution terminates with error message. |
| Multitasking Macros | | |
| ATTACH | 38 | Not supported. Execution terminates with error message. |
| DETACH | 39 | Not supported. Execution terminates with error message. |
| RCB | -- | RCB control block generated. However, CMS/DOS does not support the use of Request Control Block. |
| ENQ | 41 | No operation performed. |
| DEQ | 42 | No operation performed. |
| WAITM | 29 | Not supported. Execution terminates with error message. |
| POST | 40 | Posts ECB (byte 2 bit 0 on). The SAVE=savearea operand is ignored by CMS/DOS. |
| FREE | 36 | No operation performed. |
| Program Linkage Macros | | |
| CALL | -- | Passes control from a program to a specified entry point in another program. |
| SAVE | -- | Stores the contents of specified registers in the save area provided by the calling program. |
| RETURN | -- | Restores registers whose contents were saved and returns control to the calling program. |

Figure 41. DOS/VS Macros Supported Under CMS (Part 3 of 3)

## SEQUENTIAL ACCESS METHOD -- DECLARATIVE MACROS

CMS/DOS supports the following declarative macros:

- DTFCD
- DTFCN
- DTFDI
- DTFMT
- DTFPR
- DTFSD

The CDMOD, DIMOD, MTMOD, PRMOD, and SDMOD macros generate the logical IOCS routines that correspond with the declarative macros. The operands that CMS/DOS supports for the DTF are also supported for the xxMOD macro. In addition, CMS/DOS supports three internal macros (DTFCP, CPMOD, and DTFSL) that are required by the COBOL and PL/I compilers.

### DTFCD Macro -- Defines the File for a Card Reader

CMS/DOS does not support the ASOCFLE, FUNC, TYPEFILE=CMBND, and OUBLKSZ operands of the DTFCD macro. CMS/DOS ignores the SSELECT operand and any mode other than MODE=E. Figure 42 describes the DTFCD macro operands and their support under CMS/DOS. An asterisk (*) in the status column indicates that CMS/DOS support differs from DOS/VS support.

| Operand | Status | Description |
|---|---|---|
| DEVADDR=SYSxxx | | Symbolic unit for reader-punch used for this file. |
| IOAREA1=xxxxxxxx | * | Name of the first I/O area. |
| ASOCFLE=xxxxxxxx | * | Not supported. |
| BLKSIZE=nnn | * | Length of one I/O area, in bytes. If omitted, 80 is assumed. If CTLCHR=YES is specified, BLKSIZE defaults to 81. |
| CONTROL=YES | | CNTRL macro used for this file. Omit CTLCHR for this file. Does not apply to 2501. |
| CRDERR=RETRY | * | Retry if punching error is detected. Applies to 2520 and 2540 only. However, this situation is never encountered under CMS/DOS because hardware errors are not passed to the LIOCS module. |
| CTLCHR=xxx | | (YES or ASA). Data records have control character. YES for S/370 character set; ASA for American National Standards Institute character set. Omit CONTROL for this file. |
| DEVICE=nnnn | * | (2501, 2520, 2540, 3505, or 3525). If omitted, 2540 is assumed. |
| EOFADDR=xxxxxxxx | | Name of your end-of-file routine. |
| ERROPT=xxxxxx | * | IGNORE, SKIP, or name. Applies to 3505 and 3525 only. |
| FUNC=xxx | * | Not supported. |
| IOAREA2=xxxxxxxx | * | If two output areas are used, name of second area. |
| IOREG=(nn) | | Register number, if two I/O areas used and GET or PUT does not specify a work area. Omit WORKA. |
| MODE=xx | * | Only MODE=E is supported. |
| MODNAME=xxxxxxxx | | Name of the logic module that is used with the DTF table to process the file. |
| OUBLKSZ=nn | * | Not supported. |
| RDONLY=YES | * | Causes a read-only module to be generated. |
| RECFORM=xxxxxx | | (FIXUNB, VARUNB, UNDEF). If omitted, FIXUNB is assumed. |
| RECSIZE=(nn) | * | Register number if RECFORM=UNDEF. |

Figure 42. CMS/DOS Support of DTFCD Macro (Part 1 of 2)

| Operand | Status | Description |
|---|---|---|
| SEPASMB=YES | | DTFCD is to be assembled separately. |
| SSELECT=n | * | Ignored. |
| TYPE=xxxxxx | * | Input or output. |
| WORKA=YES | | I/O records are processed in work areas instead of the I/O areas. |

Figure 42. CMS/DOS Support of DTFCD Macro (Part 2 of 2)

## DTFCN Macro - Define the File for a Console

CMS/DOS supports all of the operands of the DTFCN macro. Figure 43 describes the operands of the DTFCN macro and their support under CMS/DOS. The status column is blank because the CMS/DOS and DOS/VS support of DTFCN are the same.

| Operand | Status | Description |
|---|---|---|
| DEVADDR=SYSxxx | | Symbolic unit for the console used for this file. |
| IOAREA1=xxxxxxxx | | Name of I/O area. |
| BLKSIZE=nnn | | Length in bytes of I/O area (for PUTR macro usage, length of output part of I/O area). If RECFORM=UNDEF, maximum is 256. If omitted, 80 is assumed. |
| INPSIZE=nnn | | Length in bytes for input part of I/O area for PUTR macro usage. |
| MODNAME=xxxxxxxx | | Logic module name for this DTF. If omitted, IOCS generates a standard name.<br><br>The logic module is generated as part of the DTF. |
| RECFORM=xxxxxx | | (FIXUNB or UNDEF). If omitted, FIXUNB is assumed. |
| RECSIZE=(nn) | | Register number if RECFORM=UNDEF. General registers 2-12, written in parentheses. |
| TYPEFLE=xxxxxx | | (INPUT, OUTPUT, or CMBND). Input processes both input and output. CMBND must be specified for PUTR macro usage. If omitted, INPUT is assumed. |
| WORKA=YES | | GET or PUT specifies work area. |

Figure 43. CMS/DOS Support of DTFCN Macro

## DTFDI MACRO - Define the File for Device Independence for System Logical Units

CMS/DOS supports all the operands of the DTFDI macro. Figure 44 describes the operands of the DTFDI macro and their support under CMS/DOS. The status columm is blank because the CMS/DOS and DOS/VS support for DTFDI is the same.

| Operand | Status | Description |
|---|---|---|
| DEVADDR=SYSxxx | | (SYSIPT, SYSLST, SYSPCH, or SYSRDR). System logical unit. CMS/DOS issues an error message if the logical unit specified on the DTF does not match the logical unit specified on the corresponding DLBL command. |
| IOAREA1=xxxxxxxx | | Name of the first I/O area. |
| EOFADDR=xxxxxxxx | | Name of your end-of-file routine. |
| ERROPT=xxxxxxxx | | (IGNORE, SKIP, or name of your error routine). Prevents termination cn errors. |
| IOAREA2=xxxxxxxx | | If two I/O areas are used, name of second area. |
| IOREG2=(nn) | | Register number. If omitted and two I/O areas are used, register 2 is assumed. General registers 2-12, written in parentheses. |
| MODNAME=xxxxxxxx | | DIMOD name for this DTF. If omitted, IOCS generates a standard name. |
| RDONLY=YES | | Generates a read-only module. Requires a module save area for each routine using the module. |
| RECSIZE=nnn | | Number of characters in record. Assumed values: 121 (SYSLST), 81 (SYSPCH), 80 (otherwise). |
| SEPASMB=YES | | DTFDI to be assembled separately. |
| WLRERR=xxxxxxxx | | Name of your wrong-length-record routine. |

Figure 44. CMS/DOS Support of DTFDI Macro

## DTFMT Macro -- Define the File for a Magnetic Tape

CMS/DOS does not support the ASCII, BUFOFF, HDRINFO, LENCHK, and READ=BACK operands of the DTFMT macro. Tape I/O operations are limited to reading in the forward direction.

CMS/DOS creates unlabeled tapes and bypasses standard labels. User-written label processing routines are used, when supplied. CMS/DOS handles tapes labels as follows:

| If | Then |
|---|---|
| Input tape has label | The CMS/DOS open routine positions the tape at the first data record. |
| Input tape has a standard label | The CMS/DOS open routine positions the tape to the first data record (that is, standard labels are bypassed). If user labels are detected and if a user label routine is specified (LABADDR=xxxxxxxx) in the DTF table for the file, CMS/DOS exits to the user's routines to read and process the user labels. |
| Input tape has nonstandard label | The CMS/DOS open routine exits to the user's routine specified by the LABADDR=xxxxxxxx operand of the DTFMT macro. If no user routine is specified, the tape is positioned at the first data record. |
| Tape opened for output | CMS/DOS treats all tapes (standard labeled tapes, nonstandard labeled tapes, and unlabeled tapes) as if they were unlabeled. If a tape with a standard or nonstandard label is opened for output, CMS/DOS writes over the label. This is also true for tape workfiles because they are opened for output first. |

The CMS/DOS close routine does not perform trailer label checking on input files. No trailer label processing is provided for input or output tape files.

Figure 45 describes the DTFMT macro operands and their support under CMS/DOS. An asterisk (*) in the Status column indicates that CMS/DOS support differs from DOS/VS support.

| Operand | Status | Description |
|---|---|---|
| BLKSIZE=nnnnn | | Length of one I/O area in bytes (maximum = 32,767). |
| DEVADDR=SYSxxx | | Symbolic unit for tape drive used for this file. |
| EOFADDR=xxxxxxxx | | Name of your end-of-file routine. |
| FILABL=xxxx | * | (NO, STD, or NSTD). If NSTD specified, include LABADDR. User label routines are only supported for header labels on input tapes. |
| IOAREA1=xxxxxxxx | | Name of first I/O area. |
| ASCII=YES | * | Not supported. |
| BUFOFF=nn | * | Not supported. |
| CKPTREC=YES | | Checkpoint records are interspersed with input data records. IOCS bypasses checkpoint records. |
| ERREXT=YES | | Additional errors and ERET are desired. |
| ERROPT=xxxxxxxx | | (IGNORE, SKIP, or name of error routine). Prevents job termination on error records. |
| HDRINFO=YES | * | Not supported. |
| IOAREA2=xxxxxxxx | | If two I/O areas are used, the name of the second area. |
| IOREG=(nn) | | Register number. Use only if GET or PUT does not specify a work area or if two I/O areas are used. Omit WORKA. General register 2-12, written in parentheses. |
| LABADDR=xxxxxxxx | * | Name of your label routine if FILABL=NSTD, or if FILABL=STD and user-standard labels are processed. Note: User label routines are only supported for header labels on input tapes. |
| LENCHK=YES | * | Not supported. |
| MODNAME=xxxxxxxx | | Name of MTMOD logic module for this DTF. If omitted, IOCS generates standard name. |
| NOTEPNT=xxxxxx | | (YES or POINTS). YES if NOTE, POINTW, POINTR, or POINTS macro used. POINTS if only POINTS macro used. |
| RDONLY=YES | | Generate read-only module. Requires a module save area for each routine using the module. |
| READ=xxxxxxx | * | CMS/DOS only supports READ=FORWARD. |

Figure 45. CMS/DOS Support of DTFMT Macro (Part 1 of 2)

| Operand | Status | Description |
|---|---|---|
| RECFORM=xxxxxx | | (FIXUNB, FIXBLK, VARUNB, VARBLK, SPNUNB, SPNBLK, or UNDEF). For work files use FIXUNB or UNDEF. If omitted, FIXUNB is assumed. |
| RECSIZE=nnnn | | If RECFORM=FIXBLK, number of characters in the record. If RECFORM=UNDEF, register number. Not required for other records. General registers 2-12, written in parentheses. |
| REWIND=xxxxxx | | (UNLOAD or NORWD). Unload on CLOSE or end-of-volume, or prevent rewinding. If omitted, rewind only. |
| SEPASMB=YES | | DTFMT is to be assembled separately. |
| TPMARK=NO | | Prevent writing a tapemark ahead of data records if FILABL=NSTD or NO. |
| TYPEFLE=xxxxxx | | (INPUT, OUTPUT, or WORK). If omitted, INPUT is assumed. |
| VARBLD=(nn) | | Register number, if RECFORM=VARBLK and records are built in the output area. General registers 2-12 are written in parentheses. |
| WLRERR=xxxxxxxx | | Name of wrong-length-record routine. |
| WORKA=YES | | GET or PUT specifies a work area. Omit IOREG. |

Figure 45. CMS/DOS Support of DTFMT Macro (Part 2 of 2)

## DTFPR Macro - Define the File for a Printer

CMS/DOS does not support the ASOCFLE, ERROPT=IGNORE, and FUNC operands of the DTFPR macro. Figure 46 describes the operands of the DTFPR macro and their support under CMS/DOS. An asterisk (*) in the Status column indicates that CMS/DOS support differs from DOS/VS support.

| Operand | Status | Description |
|---|---|---|
| DEVADDR=SYSxxx | | Symbolic unit for the printer used for this file. |
| IOAREA1=xxxxxxxx | | Name for the first output area. |
| ASOCFLE=xxxxxxxx | * | Not supported. |
| BLKSIZE=nnn | * | Length of one output area, in bytes. If omitted, 121 is assumed. |
| CONTROL=YES | | CNTRL macro used for this file. Omit CTLCHR for this file. |
| CTLCHR=xxx | | (YES or ASA). Data records have control character. YES for S/370 character set; ASA for American National Standards Institute character set. Omit CONTROL for this file. |
| DEVICE=nnnn | * | (1403, 1443, or 3211). If omitted, 1403 is assumed. |
| ERROPT=xxxxxxxx | * | RETRY or the name of your error routine for 3211. Not allowed for other devices. IGNORE is not supported. |
| FUNC=xxxx | * | Not supported. |
| IOAREA2=xxxxxxxx | | If two output areas are used, name of second area. |
| IOREG=(nn) | | Register number; if two output areas used and PUT does not specify a work area. Omit WORKA. |
| MODNAME=xxxxxxxx | | Name of PRMOD logic module for this DTF. If omitted, IOCS generates standard name. |
| PRINTOV=YES | | PRTOV macro used for this file. |
| RDONLY=YES | | Generate a read-only module. Requires a module save area for each routine using the module. |
| RECFORM=xxxxxx | | (FIXUNB, VARUNB, or UNDEF). If omitted, FIXUNB is assumed. |
| RECSIZE=(nn) | | Register number if RECFORM=UNDEF. |
| SEPASMB=YES | | DTFPR is to be assembled separately. |
| STLIST=YES | | 1403 selective tape listing feature is to be used. |
| UCS=xxx | | (ON) process data checks. (OFF) ignores data checks. Only for printers with the UCS feature or 3211. If omitted, OFF is assumed. |
| WORKA=YES | | PUT specifies work area. Omit IOREG. |

Figure 46. CMS/DOS Support of DTFPR Macro

## DTFSD Macro - Define the File for a Sequential DASD

CMS/DOS does not support the FEOVD, HOLD, and LABADDR operands of the DTFSD macro. Figure 47 describes the operands of the DTFSD macro and their support under CMS/DOS. An asterisk (*) in the status column indicates that CMS/DOS support differs from DOS/VS support.

| Operand | Status | Description |
|---|---|---|
| BLKSIZE=nnnn | | Length of one I/O area, in bytes. |
| EOFADDR=xxxxxxxx | | Name of your end-of-file routine. |
| IOAREA1=xxxxxxxx | | Name of first I/O area. |
| CONTROL=YES | | CNTRL macro used for this file. |
| DELETFL=NO | * | If DELETFL=NO is specified, the work file is not erased. Otherwise, when the work file is closed, CMS/DOS erases it. |
| DEVADDR=SYSnnn | * | Symbolic unit. This operand is optional. If DEVADDR is not specified, all I/O requests are directed to the logical unit identified on the corresponding CMS/DOS DLBL command.<br><br>If a valid logical unit is specified with the DEVADDR operand of the DTF and a different, but also valid, logical unit is specified on the DLBL command, the unit specified on the DLBL command overrides the unit specified in the DTF. However, CMS/DOS issues an error message if a valid logical unit is specified in the DTF and no logical unit is specified on the corresponding DLBL command. |
| DEVICE=nnnn | * | (2314, 3330, 3340). If omitted, 2311 is assumed at compilation time. At execution time, when the CMS/DOS $$BOPEN routine is opening a DTFSD work file, the device code in the DTF corresponds to the device code of the device the logical unit is assigned to.<br><br>All DTFSD output files and DTFSD input files that reside on CMS disks are handled in the same manner. This device code cannot be overridden by the compilers.<br><br>You must specify the DEVICE=nnnn operand correctly for input files residing on DOS disks; otherwise, CMS/DOS issues an error |

Figure 47. CMS/DOS Support of DTFSD Macro (Part 1 of 3)

| Operand | Status | Description |
|---|---|---|
| | | message. |
| ERREXT=YES | | Additional error facilities and ERET are desired. Specify ERROPT. |
| ERROPT=xxxxxxxx | | (IGNORE, SKIP, or name of error routine). Prevents job termination on error records. Do not use SKIP for output files. |
| FEOVD=YES | * | Not supported. |
| HOLD=YES | * | Not supported. HOLD=YES is specified for DTFSD update or work files to provide a track hold capability. However, the CMS/DOS open routine sets the track hold bit off and bypasses track hold processing. |
| IOAREA2=xxxxxxxx | | If two I/O areas are used, name of second area. |
| IOREG=(nn) | | Register number. Use only if GET or PUT does not specify work area or if two I/O areas are used. Omit WORKA. |
| LABADDR=xxxxxxxx | * | Not supported. |
| MODNAME=xxxxxxxx | | Name of SDMODxx logic module for this DTF. If omitted, IOCS generates standard name. |
| NOTEPNT=xxxxxxxx | | (YES or POINTRW). YES if NOTE/POINTR/POINTW/ POINTS used. POINTRW if only NOTE/POINTR/ POINTW used. |
| RDONLY=YES | | Generates a read-only module. Requires a module save area for each routine using the module. |
| RECFORM=xxxxxx | | (FIXUNB, FIXBLK, VARUNB, SPNUNB, SPNBLK, VARBLK, or UNDEF). If omitted, FIXUNB is assumed.<br><br>For work files use FIXUNB or UNDEF. Although work files contain fixed-length, unblocked records, the CMS file system handles work files as variable-length record files. |
| RECSIZE=nnnnn | | If RECFORM=FIXBLK, number of characters in record. If RECFORM=SPNUNB, SPNBLK, or UNDEF, register number. Not required for other records. |
| SEPASMB=YES | | DTFSD is to be assembled separately. |
| TRUNCS=YES | | RECFORM=FIXBLK or TRUNC macro used for this file. |
| TYPEFLE=xxxxxx | | (INPUT, OUTPUT, or WORK). If omitted, INPUT is assumed. |
| UPDATE=YES | | Input file or work file is to be updated. |

Figure 47. CMS/DOS Support of DTFSD Macro (Part 2 of 3)

| Operand | Status | Description |
|---|---|---|
| VARBLD=(nn) | | Register number if RECFORM=VARBLK and records are built in the output area. Omit if WORKA=YES. |
| VERIFY=YES | | Check disk records after they are written. |
| WLRERR=xxxxxxxx | | Name of your wrong-length-record routine. |
| WORKA=YES | | GET or PUT specifies work area. Omit IOREG. Required for RECFORM=SPNUNB or SPNBLK. |

Figure 47. CMS/DOS Support of DTFSD Macro (Part 3 of 3)

## SEQUENTIAL ACCESS METHOD -- IMPERATIVE MACROS

CMS/DOS supports the following imperative macros:

- Initialization macros: OPEN and OPENR
- Processing macros: GET, PUT, PUTR, RELSE, TRUNC, CNTRL, ERET, and PRTOV. (Note: No code is generated for the CHNG macro.)
- Work file macros for tape and disk: READ, WRITE, CHECK, NOTE, POINTR, POINTW, and POINTS.
- Completion macros: CLOSE and CLOSER

CMS/DOS supports workfiles containing fixed-length, unblocked records and undefined records. Disk work files are supported as single volume, single pack files. Normal extents and split extents are both supported.

## DOS/VS TRANSIENT ROUTINES

CMS/DOS uses the DOS/VS LIOCS transient routines without change. CMS/DOS accesses the LIOCS routines directly from a DOS/VS system or private library. For this reason, you must order and install DOS/VS before you can use CMS/DOS.

However, CMS/DOS simulates the DOS/VS transients that are fetched by macro expansion or by the LIOCS modules. These simulation routines contain enough of the transient's function to support the DOS/VS COBOL compiler and DOS PL/I Optimizing Compiler. These routines which simulate the DOS/VS transients execute in the CMSDOS discontiguous shared segment.

The following DOS/VS transients are simulated by CMS/DOS.

| Transient | Function under CMS/DOS |
|---|---|
| $$BOPEN | Fetched by the DOS/VS OPEN macro expansion or by the DOS/VS LIOCS modules. $$BOPEN performs DTF initialization, dependent upon the device type, to ready the file for I/O operations. At entry to $$BOPEN, register 0 points to a list of fullword addresses containing a pointer to the DTFs. $$BOPEN checks for supported DTF types, and initializes DTFs in accordance with the device type. In the case of disk |

| Transient | Function under CMS/DOS |
|---|---|
| | files and tape data files, default DLBLs with the NOCHANGE option are issued. (The CMS STATE command is issued to verify the existence of the input files.) |
| | $$BOPEN is invoked to supply additional extent information for multi-extent real DOS data sets. $$BOPEN is also called to initialize DTFs with EXTENT information for private and system DOS libraries. The OPEN transient is responsible for providing the proper extent information as a result of POINTR/POINTS requests. If a VSAM file is being opened (Byte 20 = X'28' in the ACB), control is passed to the VSAM OPEN routine. When opening DTFSD files for output or DTFCP/DTFDI disk files for output, if a file exists on a CMS disk with the same filename, filetype, and filemode, the file is erased. |
| $$BOPNLB | Fetched by COBOL Compiler Phase 00 to read the appropriate system or private source statement library directory record and to determine whether active members are present for the library. |
| $$BCLOSE | Fetched by DOS/VS CLOSE macro expansion to deactivate a file. |
| $$BDUMP | Fetched when an abnormal termination condition is encountered. Control is not passed to a STXIT routine. CMS/DOS performs a CP dump to a virtual printer. The routine is canceled. |
| $$BOPENR | Fetched by a DOS/VS OPENR macro expansion. The function of $$BOPENR is to relocate all DTF table address constants from the assembled addresses to executable storage addresses. At entry to $$BOPENR, register 0 points to an assembled address constant followed by a list of DTF addresses tables that require address modification. |
| $$BOPNR3 | Fetched by $$BOPENR to relocate all DTF table address constants for unit record DTFs. |
| $$BOPNR2 | Fetched by $$BOPNR3 to relocate all DTF table address constants for DTFDI or DTFCP. |

## EXCP SUPPORT IN CMS/DOS

CMS/DOS simulates the EXCP (execute channel program) routines to the extent necessary to support the LIOCS routines described in the preceding section "DOS/VS Supervisor and I/O Macros Supported by CMS/DOS."

Because CMS/DOS uses the DOS/VS LIOCS routines unchanged, it must simulate all I/O at the EXCP level. The EXCP simulation routines convert all the I/O that is in the CCW format to CMS physical I/O requests. That is, CMS macros (such as RDBUF/WRBUF, CARDRD/CARDPH, PRINTIO, and WAITRD/TYPLIN) replace the CCW strings. If CMS/DOS is reading from DOS disks, I/O requests are handled via the DIAGNOSE interface.

When an I/O operation completes, CMS/DOS posts the CCB with the CMS return code. Partial RPS (rotational position sensing) support is available for I/O operations to CMS disks because CMS uses RPS in its channel programs. However, RPS is not supported when real DOS disks are read.

## DOS/VS SUPERVISOR CONTROL BLOCKS SIMULATED BY CMS/DOS

CMS/DOS supports DOS/VS program development and execution for a single partition: the background partition. Because CMS/DOS does not support the four foreground partitions, it also does not simulate the associated control blocks and fields for foreground partitions. CMS/DOS does simulate the following DOS/VS supervisor control blocks:

- ABTAB--Abnormal Termination Option Table
- BBOX--Boundary Box
- BGCOM--Background Partition Communication Region
- EXCPW--Work area for module DMSXCP
- FICL--First in Class
- LUB--Logical Unit Block
- NICL--Next in Class
- PCTAB--Program Check Option Table
- PIBTAB--Program Information Table
- PIB2TAB--Program Information Block Table Extension
- PUB--Physical Unit Block
- PUBOWNER--Physical Unit Block Ownership Table
- SYSCOM--System Communication Region

For detailed descriptions of CMS/DOS control blocks, refer to the VM/370: Data Areas and Control Blocks Logic.

## USER CONSIDERATIONS AND RESPONSIBILITIES

A critical design assumption of CMS/DOS is that installations that use CMS/DOS will also use and have available a DOS/VS system. Therefore, if you want to use CMS/DOS you must first order and install a DOS/VS system, Release 31 (or later). Also, if you want to use the DOS/VS COBOL and DOS PL/I Optimizing compilers under CMS/DOS, you must order them and install them on your DOS/VS system.

There are several other facts you should consider if you plan to use CMS/DOS. The following sections describe some of the user considerations and responsibilities.

## DOS/VS SYSTEM GENERATION AND UPDATING CONSIDERATIONS

The CMS/DOS support in CMS may use a real DOS/VS system pack. CMS/DOS provides the necessary interface and then fetches DOS/VS logical transients and system routines directly from the DOS/VS COBOL and DOS PL/I Optimizing compilers directly from the DOS/VS system or private core image libraries.

It is your responsibility to order a Release 31 (or later) DOS/VS system and then generate it. Also, if you plan to use DOS compilers, you must order the current level of the DOS/VS COBOL compiler and DOS PL/I Optimizing compilers and install them on the same DOS/VS system.

When you install the compilers on the DOS/VS system, you must link-edit all the compiler modules as relocatable phases using the following linkage editor control statement:

```
ACTION REL
```

You can place the link-edited phases in either the system or private core image library.

When you later invoke the compilers from CMS/DOS, the library (system or private) containing the compiler phases must be identified to CMS. You identify all the system libraries to CMS by coding the filemode letter that corresponds to that DOS/VS system disk on the SET DOS ON command when you invoke the CMS/DOS environment. You identify a private library by coding ASSGN and DLBL commands that describe it. The DOS/VS system and private disks must be linked to your virtual machine and accessed before you issue the commands to identify them for CMS.

CMS/DOS has no effect on the update procedures for DOS/VS, DOS/VS COBOL, or DOS PL/I. Normal update procedures for applying IBM-distributed coding changes apply.

For detailed information on how to generate VM/370 with CMS/DOS, refer to the publication VM/370: Planning and System Generation.

## VM/370 DIRECTORY ENTRIES

The DOS/VS system and private libraries are accessed in read-only mode under CMS/DOS. If more than one CMS virtual machine is using the CMS/DOS environments you should update the VM/370 directory entries so that the DOS/VS system residence volume and the DOS/VS private libraries are shared by all the CMS/DOS users.

The VM/370 directory entry for one of the CMS virtual machines should contain the MDISK statements defining the DOS/VS volumes. The VM/370 directory entries for the other CMS/DOS users should contain LINK statements.

For example, assume the DOS/VS system libraries are on cylinders 0-149 of a 3330 volume labeled DOSRES. And, assume the DOS/VS private libraries are on cylinders 0-99 of a 2314 volume labeled DOSPRI. Then, one CMS machine (for example, DOSUSER1) would have the MDISK statements in its directory entry.

```
USER DOSUSER1 password 320K 2M G
     .
     .
     .
MDISK 331 3330 0 150 DOSRES R rpass
MDISK 231 2314 0 100 DOSPRI R rpass
```

All the other CMS/DOS users would have links to these disks. For example

```
LINK DOSUSER1 331 331 R rpass
LINK DOSUSER1 231 231 R rpass
```

## CMS/DOS STORAGE REQUIREMENTS

CMS/DOS requires DASD space to contain its source, text, module, and EXEC files. This DASD requirement is in addition to the space already required for CMS system residence. The DASD space required by CMS/DOS is:

- 21 cylinders on a 2314/2319
- 12 cylinders on a 3330
- 33 cylinders on a 3340/3344
- 6 cylinders on a 3350

A simulated DOS/VS nucleus, eight DOSLIB directories, and the simulated DOS/VS control blocks (approximately 1300 decimal bytes) are located in the CMS nucleus.

CMS/DOS also uses the CMS user area. CMS/DOS executes the DOS compilers, linkage editor, and librarian programs in the CMS user area. The virtual storage requirements are:

- 60K plus buffers for the DOS/VS COBOL compiler
- 44K plus buffers for the DOS PL/I Optimizing compiler
- 20K for the CMS/DOS linkage editor
- 3K for the RSERV librarian program
- 2K for the PSERV librarian program
- 2K for the SSERV librarian program

CMS also uses the user area for its own purposes when processing CMS/DOS programs. For specific information on CMS use of free storage, refer to the section "Free Storage Management."

## WHEN THE DOS/VS SYSTEM MUST BE ONLINE

Most of what you do in the CMS/DOS environment requires that the DOS/VS system pack and/or the DOS/VS private libraries be available to CMS/DOS. In general, you need these DOS/VS volumes whenever:

- You use the DOS/VS COBOL compiler or DOS PL/I Optimizing compiler. The compilers are executed from the system or private core image libraries.

- Your source programs contain COPY, LIBRARY, %INCLUDE, or CBL statements. These statements copy books from your system or private source statement library.

- You invoke one of the librarian programs: DSERV, RSERV, SSERV, PSERV, or ESERV.

- You execute DOS programs that use LIOCS modules. CMS/DOS fetches most of the LIOCS routines directly from DOS/VS system or private libraries.

A DOS/VS system pack is usable when it is:

- Defined for your virtual machine
- Accessed
- Specified, by mode letter, on the SET DOS ON command

A DOS/VS private library is usable when it is:

- Defined for your virtual machine
- Accessed
- Identified via ASSGN and DLBL commands

## PERFORMANCE

Although you can use the CMS/DOS librarian services to place the DOS/VS COBOL compiler, DOS PL/I compiler, and ESERV program in a CMS DOSLIB, VM/370 recommends that you do not. CMS/DOS can fetch these directly from the DOS/VS system or private libraries faster than from a DOSLIB.

## EXECUTION CONSIDERATIONS AND RESTRICTIONS

The CMS/DOS environment does not support the execution of DOS programs that use:

- Sort exits. The DOS/VS COBOL and DOS PL/I SORT verbs are not supported in CMS/DOS.
- Teleprocessing or indexed sequential (ISAM) access methods. CMS/DOS supports only the sequential (SAM) and virtual storage (VSAM) access methods.
- Multitasking. CMS/DOS supports only a single partition, the background partition.

CMS/DOS can be executed in a CMS Batch Facility virtual machine. If any of the DOS programs that are executed in the batch machine read data from the card reader you must be sure that the end-of-data indication is recognized. Be sure that the program checks for end-of-data and be sure that a /* record follows the last data record.

If there is an error in the way you handle end-of-data, the DOS program could read the entire batch input stream as it's own data. The result is that jobs sent to the batch machine are never executed and the DOS program reads records that are not part of it's input file.

# 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 virtual 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

## DOS/VS SUPERVISOR MACROS AND LOGICAL TRANSIENTS SUPPORT FOR VSAM

CMS supports VSAM for OS and DOS users. However, the CMS support of VSAM is based on DOS/VS. The DOS/VS supervisor macros shown in Figure 48, which are used by the DOS/VS VSAM routines, are supported by CMS.

| Macro | SVC Number | Extent of CMS Support |
|---|---|---|
| CDLOAD | 65 | DOS/VS macro for internal use only. Loads a VSAM core image phase. CMS searches the VSAM saved segment for the phase instead of the DOS/VS SVA area. If an anchor table entry does not exist, CMS fetches the phase, creates an anchor table entry, and sets register values as DOS/VS would set them. |
| FREE | 36 | No operation is performed by CMS. |
| FREEVIS | 62 | CMS invokes its free storage handler to return the storage that is no longer needed. CMS follows the DOS/VS register and return code conventions. |
| GETVIS | 61 | CMS invokes its free storage handling routines to obtain free storage; it follows the DOS/VS register and return code conventions. The SVA operand does not apply to CMS and is not supported. The PAGE and POOL operands are ignored by CMS. |
| HOLD | 35 | No operation is performed by CMS. |
| POST | 40 | When a POST macro is issued for an ECB, Byte 2 Bit 0 is set on. The SAVE=savarea operand is ignored by CMS. |
| RELEASE | 64 | CMS reduces the RURTBL counter for the resource by one. |
| SECTVAL | 75 | CMS uses the data in registers 0 and 1 to calculate the sector number and returns the sector number in register 0. If any errors occur, CMS returns X'FF' in register 0. |
| USE | 63 | DOS/VS macro for internal use only. CMS supports this macro only to the extent necessary to support VSAM. If a counter for a particular resource is zero, CMS increments the counter by one and returns a zero in register 0. If a counter is greater than zero, CMS increments the counter by one and returns an eight in register 0. |

Figure 48. DOS/VS VSAM Macros Supported by CMS

CMS distributes the DOS/VS transients that are needed in the VSAM support. Thus, OS users do not need to have the DOS/VS system pack online when they are compiling and executing VSAM programs.

CMS uses all of the DOS/VS VSAM B-transients except those that build and release JIBs (job information blocks). The JIB is not supported in CMS and, thus, neither are the B-transients ($$BJIB00, $$BJIBFF, and $$BOVS03) that control the JIB.

The CMS/DOS shared segment contains the B-transients that are simulated for DOS support in CMS. Three B-transients that pertain only to VSAM are included in the VSAM saved segment: $$BOMSG1, $$BOMSG2, and $$BENDQB. The $$BENDQB transient is called by the ENQB macro and released by the DEQB macro.

## STORAGE REQUIREMENTS

The VSAM and Access Method Services support in CMS requires both DASD space and virtual storage.

The VSAM and Access Method Services support adds approximately 2K to the size of the CMS nucleus. In addition, this support uses free storage to execute the DOS/VS logical transients and for buffers and work areas. VSAM issues a GETVIS macro to request free storage.

If the CMS/DOS environment is invoked with the VSAM option

```
SET DOS ON (VSAM
```

part of the CMS/DOS virtual storage is set aside for VSAM use.

Disk storage requirements vary depending on device type:

| Device Type | Number of Cylinders Required DOS User | OS User |
|---|---|---|
| 2314 | 10 | 20 |
| 2319 | 10 | 20 |
| 3330 Model 1 | 6 | 12 |
| 3340 | 15 | 30 |
| 3344 | 15 | 30 |
| 3350 | 3 | 6 |

## DATA SET COMPATIBILITY CONSIDERATIONS

CMS can read and update VSAM data sets that were created under DOS/VS or OS/VS. In addition, VSAM data sets created under CMS can be read and updated by DOS/VS or OS/VS.

However, if you perform allocation on a minidisk in CMS, you cannot use that minidisk in an OS virtual machine in any manner that causes further allocation. DOS/VS VSAM (and thus CMS) ignores the format-5, free space, DSCB on VSAM disks when it allocates extents. If allocation later occurs in an OS machine, OS attempts to create a format-5 DSCB. However, the format-5 DSCB created by OS does not correctly reflect the free space on the minidisk. In CMS, allocation occurs whenever data spaces or unique data sets are defined. Space is released whenever data spaces, catalogs, and unique data spaces are deleted.

## ISAM INTERFACE PROGRAM (IIP)

CMS does not support the VSAM ISAM Interface Program (IIP). Thus, any program that creates and accesses ISAM (indexed sequential access method) data sets cannot be used to access VSAM key sequential data sets. There is one exception to this restriction. If you have (1) OS PL/I programs that have files declared as ENV(INDEXED) and (2) if the library routines detect that the data set being accessed is a VSAM data set, your programs will execute VSAM I/O requests.

Only named systems can be saved. The NAMESYS macro must be used to name a system. A discussion on creating a named system is found under "Generating Named System" in "Part 2: Control Program (CP)".

The DMKSNT module must have been configured (by coding the NAMESYS macro) when CP was generated. The DMKSNT module contains the system name, size of the system, and its real disk location. The CMS system may be saved by entering the command 'SAVESYS name' as the first command after the IPL command (that is, after the CMS version identification is displayed), where 'name' is the name to be assigned to the saved system.

The CMS S, D, and Y disks (and, optionally, the A disk) should be mounted and attached to the virtual machine creating the saved system before the SAVESYS command is issued. This ensures that the CMS file directory is saved correctly.

The status of the saved system proceeds, upon a subsequent IPL, as if an IPL of a specific device had occurred, with the single exception that the file directory for the system disk is part of the nucleus.

## THE CMSSEG DISCONTIGUOUS SAVED SEGMENT

The CMSSEG discontiguous saved segment contains the CMS modules that perform the CMS Editor, EXEC, or OS simulation functions. These same modules are also loaded on the S-disk of a CMS virtual machine.

When CMSSEG has been generated and is in use during execution of the CMS virtual machine, CMS handles a call to the Editor or EXEC processors by first searching for the requested Editor or EXEC load modules on all accessed CMS disks, except the S-disk. CMS next attempts to attach the CMSSEG segment; CMSSEG may not be available, depending on how the CMS virtual machine was generated. If this is the case, CMS attempts to load the appropriate modules from the CMS S-disk.

To handle OS simulation routines, CMS first attempts to attach the CMSSEG segment. If the segment is not available, CMS searches all accessed disks for the OS simulation load modules and loads them into high user storage if they are found. The OS simulation modules are then kept in storage until CMS is reloaded or until a SET SYSNAMES command is issued for a valid CMSSEG saved segment.

There is overhead associated with controlling discontiguous saved segments and with ensuring their integrity. In small systems, the overhead associated with the use of the CMSSEG saved segment may not be offset by the benefits of sharing storage between users. Therefore, the use of CMSSEG must be determined by the user for his own environment.

### CMSSEG USAGE OPTIONS

At system generation time, you may choose to not generate the CMSSEG segment. If you choose to use it, information on how to generate it is contained in the publication VM/370: Planning and System Generation Guide.

Once generated, users also have the option of choosing whether or not to use CMSSEG. The IPL command provides the facility to either use or not use CMSSEG. When you IPL your CMS virtual machine, you can request that CMSSEG be used by specifying

```
IPL CMS PARM SEG=CMSSEG
```

SEG=CMMSEG is the default option.

To request that CMSSEG not be used for your virtual machine, specify an invalid segment name in the IPL command, for example,

```
IPL CMS PARM SEG=DUMMY
```

When the CMSSEG segment is not loaded, the routines that perform the Editor, EXEC, and OS simulation functions execute in the CMS user area.

The SET command also can be used to control the use of the CMSSEG segment. If the CMS system is IPLed with an invalid CMSSEG segment name specified, DMSSVT is loaded in the CMS user area to provide support for OS simulation routines. In this case the Editor and EXEC modules must be available on the S-disk or another accessed idsk, in this case.

When the command SET SYSNAME CMSSEG segmentname is specified (where segmentname is the name of a defined CMS discontiguous saved segment), free storage containing OS simulation routines is released and OS simulation routines contained in the CMSSEG segment are used by the virtual machine to provide OS simulation functions.

## SAVED SYSTEM RESTRICTIONS FOR CMS

There are several coding restrictions that must be imposed on CMS if it is to run as a saved system.

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.

It 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.

If you intend to modify a shared CMS system, be sure that all code that is to be shared resides in the shared segment, CMS Nucleus (X'10000'-X'20000'). To make room for additional code in the CMS Nucleus, you may have to move some of the existing code. You can use the USERSECT area of DMSNUC to contain nonshared instructions.

The CMS Batch Facility is a VM/370 programming facility that runs under the CMS subsystem. It allows VM/370 users to run their jobs in batch mode by sending jobs either from their virtual machines or through the real (system) card reader to a virtual machine dedicated to running batch jobs. The Batch Facility then executes these jobs, freeing user machines for other uses.

If both CMS Batch Facility and the Remote Spooling Communications Subsystem (RSCS) are running under the same VM/370 system, job input streams can be transmitted to the Batch Facility from remote stations via communication lines. Also, the output of the batch processing can be transmitted back to the remote station. For additional information, see "Remote Job Entry to CMS Batch" in the VM/370: Remote Spooling Communications Subsystem (RSCS) User's Guide.

The Batch Facility virtual machine is generated and controlled on a userid dedicated to execution of jobs in batch mode. The system operator generates the "batch machine" by loading (via IPL) the CMS subsystem, and then issuing the CMSBATCH command. The CMSBATCH module loads the DMSBTP TEXT S2 file which is the actual Batch processor. After each job is executed, the Batch Facility will IPL itself, thereby providing a continuously running batch machine. The Batch Processor will IPL itself by using the PARM option of the CP IPL command, followed by a character string which CMS recognizes as peculiar to a batch virtual machine performing its IPL. Jobs are sent to the batch machine's virtual card reader from users' terminals and executed sequentially. When there are no jobs waiting for execution, the Batch Facility remains in a wait state ready to execute a user job. See the VM/370: Operator's Guide for more information about controlling the batch machine.

The Batch Facility is particularly useful for compute-bound jobs such as assemblies and compilations and for execution of large user programs, since interactive users can continue working at their terminals while their time-consuming jobs are run in another virtual machine.

The System Programmer controls the Batch Facility virtual machine environment by resetting the Batch Facility machine's system limits, by writing routines that handle special installation input to the Batch Facility, and by writing EXEC procedures that make the Batch Facility facility easier to use.

## RESETTING BATCH FACILITY SYSTEM LIMITS

Each job running under the Batch Facility is limited by default to the maximum value of 32,767 seconds of virtual CPU time, 32,767 punched cards output, and 32,767 printed lines of output. You can reset these limits by modifying the BATLIMIT MACRO file, which is found in the CMSLIB macro library, and reassembling DMSBTP.

## WRITING ROUTINES TO HANDLE SPECIAL INSTALLATION INPUT

The Batch Facility can handle user-specified control language and special installation Batch Facility /JOB control cards. These handling

mechanisms are built into the system in the form of user exits from Batch; you are responsible for generating two routines to make use of them. These routines must be named BATEXIT1 and BATEXIT2, respectively, and must have a filetype of TEXT and a filemode number of 2, if placed on the system disk or an extension of the system disk. (See the VM/370: CMS User's Guide for information on how to write and use Batch Facility control cards.) The routines you write are responsible for saving registers, including general register 12, which saves addressability for the Batch Facility. These routines (if made available on the system disk) are included with the Batch Facility each time it is loaded.

### BATEXIT1: PROCESSING USER-SPECIFIED CONTROL LANGUAGE

BATEXIT1 is an entry point provided so that users may write their own routine to check non-CMS control statements. For example, it could be written to scan for the OS job control language needed to compile, link edit, and execute a FORTRAN job. BATEXIT1 receives control after each read from the Batch Facility virtual card reader is issued. General register 1 contains the address of the Batch Facility Read Buffer, which contains the card image to be executed by the Batch Facility. This enables BATEXIT1 to scan each card it receives as input for the type of control information you specify.

If, after the card is processed by BATEXIT1, general register 15 contains a nonzero return code, the Batch Facility flushes the card and reads the next card. If a zero is returned in general register 15, the Batch Facility continues processing by passing the card to CMS for execution.

### BATEXIT2: PROCESSING THE BATCH FACILITY /JOB CONTROL CARD

BATEXIT2 is an entry point provided so that users can code their own routine to use the /JOB card for additional information. BATEXIT2 receives control before the VM/370 routine used to process the Batch Facility /JOB card begins its processing, but after CMS has scanned the /JOB card and built the parameter list. When BATEXIT2 is processing, general register 1 points to the CMS parameter list buffer. This buffer is a series of 8-byte entries, one for each item on the /JOB card. If the return code found in general register 15 resulting from BATEXIT2 procesing of this card is nonzero, an error message is generated and the job is flushed. If general register 15 contains a zero, normal checking is done for a valid userid and the existence of an account number. Finally, execution of this job begins.

## EXEC PROCEDURES FOR THE BATCH FACILITY VIRTUAL MACHINE

You can control the Batch Facility virtual machine using EXEC procedures. For example, you can use:

- An EXEC to produce the proper sequence of CP/CMS commands for users who do not know CMS commands and controls.
- An EXEC to provide the sequence of commands needed to execute the most common jobs (assemblies and compilations) in a particular installation.

For information on how to use the EXEC facility to control the Batch Facility virtual machine, see the VM/370: CMS User's Guide.

## DATA SECURITY UNDER THE BATCH FACILITY

After each job, the Batch Facility will IPL itself, destroying all nucleus data and work areas. All disks linked to during the previous job are detached.

At the beginning of each job, the Batch Facility work disk is accessed and then immediately erased, preventing the current user job from accessing files that might remain from the previous job. Because of this, execution of the PROFILE EXEC is disabled for the Batch Facility machine. You may, however, create an EXEC procedure called BATPROF EXEC and store it on any system disk to be used instead of the ordinary PROFILE EXEC. The Batch Facility will then execute this EXEC at each job initialization time.

## IMPROVED IPL PERFORMANCE USING A SAVED SYSTEM

Since the CMS batch processor goes through an IPL procedure after each user job, an installation may experience a a more efficient IPL procedure by using a saved CMS system when processing batch jobs.

This can be accomplished by passing the name of the saved system to the Batch Facility via the optional "sysname" operand in the CMSBATCH command line.

The Batch Facility saves the name of the saved system until the end of the first job at which time it stores the name in the IPL command line both as the "device address" and as the PARM character string. The latter entry informs the CMS initialization routine (DMSINS) that a saved system has been loaded and that the name is to be saved for subsequent IPL procedures.

When a disk is accessed, each module that fits the description specified on the ACCESS command is included in the resident directory. An auxiliary directory is an extension of the resident directory, containing the name and location of certain CMS modules that are not included in the resident directory. These modules, if added to the resident directory, would significantly increase its size, thus increasing the search time and storage requirements. An auxiliary directory can reference modules that reside on the system (S) disk; or, if the proper linkage is provided, reference modules that reside on any other read-only CMS disk. To take advantage of the saving in search time and storage, modules that are referenced via an auxiliary directory should not also be in the resident directory. The disk on which these modules reside should be accessed in a way that excludes these modules.

## HOW TO ADD AN AUXILIARY DIRECTORY

To add an auxiliary directory to CMS, the system programmer must generate the directory, initialize it, and establish the proper linkage. Only when all three tasks are completed, can a module described in an auxiliary directory be properly located.

### GENERATION OF THE AUXILIARY DIRECTORY

An auxiliary directory TEXT deck is generated by assembling a set of DMSFST macros, one for each module name. The format of the DMSFST macro is:

```
┌──────────────────────────────────────────────────────────────────────┐
| | DMSFST | (filename                    )                              |
| |        | (                ┌          ┐  )                              |
| |        | (filename        |,filetype|)  )[,aliasname]                 |
| |        | (                |,MODULE  |  )                              |
| |        | (                └          ┘  )                              |
└──────────────────────────────────────────────────────────────────────┘
```

where:

| | |
|---|---|
| filename,filetype | is the name of the module whose File Status Table (FST) information is to be copied. |
| aliasname | is another name by which the module is to be known. |

### INITIALIZING THE AUXILIARY DIRECTORY

After the auxiliary directory is generated via the DMSFST macro, it must be initialized. The CMS GENDIRT command initializes the auxiliary directory with the name and location of the modules to reside in an auxiliary directory. By using the GENDIRT command, the file entries for

a given module are loaded only when the module is invoked. The format of the GENDIRT command is:

```
| GENDIRT | directoryname [targetmode]
```

where:

directoryname is the entry point of the auxiliary directory.

targetmode is the mode letter of the disk containing the modules referenced in the auxiliary directory. The letter is the mode of the disk containing the modules at execution time, not the mode of the disk at the initialization of the directory. The default value for targetmode is S, the system disk. It is your responsibility to determine the usefulness of this operand at your installation and to inform users of programs utilizing auxiliary directories of the proper accesses.

## ESTABLISHING THE PROPER LINKAGE

The CMS module, DMSLAD, entry point DMSLADAD, must be called by a user program or interface to initialize the directory search order. The subroutine, DMSLADAD, must be called via an SVC 202 with register 1 pointing to the apropriate PLIST. The disk containing the modules listed in the auxiliary directory must be accessed as the mode specified, or implied, with the GENDIRT command before the call is issued. If it is not, the user will receive messages indicating either 'file not found' or 'error reading file'.

The coding necessary for the call is:

```
        LA    R1,PLIST
        SVC   202
        DC    AL4(error return)
```

This call must be executed before the call to any module that is to be located via an auxiliary directory.

The PLIST should be:

```
PLIST   DS    0F
        DC    CL8'DMSLADAD'
        DC    V(directoryname)
        DC    F'0'
```

The auxiliary directory is copied to nucleus free storage. The Active Disk Table (ADT) for the targetmode expressed or implied with GENDIRT is found and its file directory address chain (ADTFDA) is modified to include the nucleus copy of the auxiliary directory. A flag, ADTPSTM, in ADTFLG2 is set to indicate that the directory chain has been modified.

The address of the nucleus copy of the auxiliary directory is saved in the third word of the input parameter list and the high order byte of

the third word is set to X'80' to indicate that the directory search chain was modified and that the next call to DMSLADAD is a clear request.

To reset the directory search chain, a second call is made to DMSLADAD using the modified PLIST. DMSLADAD removes the nucleus copy of the auxiliary directory from the chain and frees it. DMSLADAD does not, however, restore the caller's PLIST to it initial state.

### Error Handling and Return Codes

An error handling routine should be coded to handle non-zero return codes in register 15. When register 15 contains 1 and the condition code is set to 2, the disk specified by the targetmode operand of the GENDIRT command was not accessed as that mode.

When register 15 contains 2 and the condition code is set to 2, the disk specified by the target mode operand of the GENDIRT command has not previously had its file directory chains modified, therefore a call to DMSLADAD to restore the chain is invalid.

## AN EXAMPLE OF CREATING AN AUXILIARY DIRECTORY

Consider an application called PAYROLL consisting of several modules. It is possible to put these modules in an auxiliary directory rather than in the resident directory. It is further possible to put the auxiliary directory on a disk other than the system disk. In this example, the auxiliary directory will be placed on the Y disk.

First, generate the auxiliary directory TEXT deck for the payroll application using the DMSFST macro:

```
PAYDIRT  START   0
         DC      F'40' LENGTH OF FST ENTRY
         DC      A(DIRTEND-DIRTBEG) SIZE OF DIRECTORY
DIRTBEG  EQU     *
         DMSFST  PAYROLL1
         DMSFST  PAYROLL2
         DMSFST  PAYROLL3
         DMSFST  PAYFICA
         DMSFST  PAYFEDTX
         DMSFST  PAYSTATE
         DMSFST  PAYCITY
         DMSFST  PAYCREDU
         DMSFST  PAYOVERT
         DMSFST  PAYSICK
         DMSFST  PAYSHIFT
         DC      2A(0) POINTER TO NEXT FST BLOCK
DIRTEND  EQU     *
         END
```

In this example, the payroll control program (PAYROLL), the payroll auxiliary directory (PAYDIRT), and all the payroll modules reside on the 194 disk.

In the payroll control module (PAYROLL), the subroutine DMSLADAD must be called to establish the linkage to the auxiliary directory. This

call must be executed before any call is made to a payroll module that is in the PAYDIRT auxiliary directory.

```
            LA    R1, PLIST
            SVC   202
            DC    AL4(ERRTN)

PLIST       DS    0F
            DC    CL8'DMSLADAD'
            DC    V(PAYDIRT)
            DC    F'0'
```

Next, all payroll modules must have their absolute core-image files generated and the payroll auxiliary directory must be initialized. In the example, the payroll control module (PAYROLL) is given a mode number of 2 while the other payroll modules are given a mode number of 1. When the PAYROLL program is finally executed, only the files on the 194 disk with a mode number of 2 will be accessed. This means only the PAYROLL control program (which includes the payroll auxiliary directory) will be referenced from the resident directory. All the other payroll modules, because they have mode numbers of 1, will be referenced via the payroll auxilary directory.

The following sequence of commands will create the absolute core-image files for the payroll modules and initialize the payroll auxiliary directory.

```
ACCESS 194 A
LOAD PAYROLL PAYDIRT
GENMOD PAYROLL          (now the auxiliary directory is included in the
                         payroll control module, but it is not yet
                         initialized.)
LOADMOD PAYROLL
INCLUDE PAYROLL1
GENMOD PAYROLL1         (this sequence of three commands is repeated for
   .                     each payroll module called by PAYROLL.)
   .
   .
LOADMOD PAYROLL
INCLUDE PAYSHIFT
GENMOD PAYSHIFT

LOADMOD PAYROLL
GENDIRT PAYDIRT Y
GENMOD PAYROLL MODULE A2
```

When it is time to execute the PAYROLL program the 194 disk must be accessed as the Y disk (the same mode letter as specified on the GENDIRT command). Also, the 194 disk is accessed in a way that includes the PAYROLL control program in the resident directory but not the other payroll modules. This is done by specifying a mode number of 2 on the ACCESS command.

```
ACCESS 194 Y/S * * Y2
```

Now, a request for a payroll module, such as PAYOVERT, can be successfully fulfilled. The auxiliary directory will be searched and PAYOVERT will be found on the Y disk.

Note: A disk referred to by an auxiliary directory must be accessed as a read-only disk.

The minimum size virtual machine required by the assembler is 256K bytes. However, better performance is generally achieved if the assembler is run in 320K bytes of virtual storage. This size is recommended for medium and large assemblies.

If more virtual storage is allocated to the assembler, the size of buffers and work space can be increased. The amount of storage allocated to buffers and work space determines assembler speed and capacity. Generally, as more storage is allocated to work space, larger and more complex macro definitions can be handled.

You can control the buffer sizes for the assembler utility data sets (SYSUT1, SYSUT2, and SYSUT3) and the size of the work space used during macro processing, by specifying the BUFSIZE assembler option. Of the storage given, the assembler first allocates storage for the ASSEMBLE and CMSLIB buffers according to the specifications in the DD statements supplied by the FILEDEF for the data sets. It then allocates storage for the modules of the assembler. The remainder of the virtual machine is allocated to utility data set buffers and macro generation dictionaries according to the BUFSIZE option specified:

BUFSIZE (STD): 37 percent is allocated to buffers, and 63 percent to work space. This is the default chosen, if you do not specify any BUFSIZE option.

BUFSIZE(MIN): Each utility data set is allocated a single 790-byte buffer. The remaining storage is allocated to work space. This allows relatively complex macro definitions to be processed in a given virtual machine size, but the speed of the assembly is substantially reduced.

## OVERLAY STRUCTURES

An overlay structure can be created in CMS in two different ways, although CMS has no overlay supervision.

See the VM/370: CMS Command and Macro Reference for descriptions of all the CMS commands mentioned.

### PRESTRUCTURED OVERLAY

A prestructured overlay program is created using the LOAD, INCLUDE and GENMOD commands. Each overlay phase or segment is a nonrelocatable core-image module, created by GENMOD. The phases may be brought into storage with the LOADMOD command.

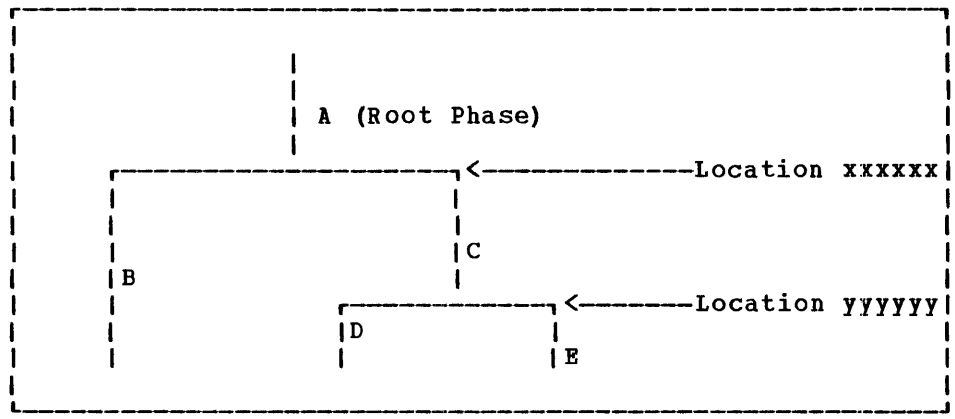


Figure 49. An Overlay Structure

The overlay structure shown in Figure 48 could be prestructured using the following sequence of commands (Programs A, B, C, D, and E are the names of TEXT files; the overlay phases will be named Root, Second, Third, etc.):

```
LOAD    A B
GENMOD  ROOT   (FROM A TO B STR)
GENMOD  SECOND (FROM B)
LOADMOD ROOT
INCLUDE C D
GENMOD  THIRD  (FROM C TO D)
GENMOD  FOURTH (FROM D)
LOADMOD THIRD
INCLUDE E
GENMOD  FIFTH  (FROM E)
```

The programmer need not know the storage address where each phase begins. A TEXT file can be made to load at the proper address by reloading earlier phases. In the foregoing example, the command sequences, "LOADMOD ROOT/INCLUDE C D" and "LOADMOD THIRD/INCLUDE E," cause TEXT files C, D, and E to load at the proper addresses.

If the root phase contains address constants to the other phases, one copy of the root must be kept in storage while each of the other phases is brought in by the LOAD or INCLUDE commands without an intervening GENMOD. The root phase is then processed by GENMOD after all address constants have been satisfied. In this case, the programmer must know the address where nonroot phases begin (in Figure 48, locations xxxxxx and yyyyyy). The following sequence of commands could be used:

```
LOAD    A B
GENMOD  SECOND (FROM B)
INCLUDE C D    (ORIGIN xxxxxx)
GENMOD  THIRD  (FROM C TO D)
GENMOD  FOURTH (FROM D)
INCLUDE E      (ORIGIN yyyyyy)
GENMOD  FIFTH  (FROM E)
LOAD    A B
INCLUDE C D    (ORIGIN xxxxxx)
INCLUDE E      (ORIGIN yyyyyy)
GENMOD  ROOT   (FROM A TO C STR)
```

The ORIGIN option of the INCLUDE command is used to cause the included file to overlay a previously loaded file. The address at which a phase begins must be a doubleword boundary. For example, if the root phase were X'2BD' bytes long, starting at virtual storage location X'20000', then location xxxxxx would be the next doubleword boundary, or X'202C0'.

The STR option, which is specified in the GENMOD of the root phase, specifies that whenever that module is brought into storage with the LOADMOD command, the Storage Initialization routine should be invoked. This routine initializes user free storage pointers.

At execution time of the prestructed overlay program, each phase is brought into storage with the LOADMOD command. The phases can call LOADMOD. The OS macros LINK, LOAD, and XCTL normally invoke the INCLUDE command, which loads TEXT files. These macros will invoke LOADMOD if a switch, called COMPSWT, in the CMS Nucleus Constant area, NUCON, is turned on.

With COMPSWT set, overlay phases that use LINK, LOAD, and XCTL must be prestructured MODULE files.

## DYNAMIC LOAD OVERLAY

The dynamic load method of using an overlay structure is to have all the phases in the form of relocatable object code in TEXT files or members of a TEXT library, filetype TXTLIB. The OS macros, LINK, LOAD, and XCTL may then be used to pass control from one phase to another. The XCTL macro causes the calling program to be overlayed by the called program except when it is issued from the root phase. When issued from the root phase, CMS treats XCTL as a LINK, adding the new code at the end of the root phase.

The COMPSWT flag in OSSFLAGS must be off when the dynamic load method is used.

# Part 4. IBM 3704 and 3705 Communications Controllers

Part 4 describes the procedures a system programmer must follow to load, test and run a 3704/3705 control program with VM/370. Part 4 includes the following information:

- Introduction
- VM/370 Support of the 3704/3705
- Loading the 3704/3705 Control Program
- Testing the 3704/3705 Control Program

# Introduction to the IBM 3704 and 3705 Communications Controllers

The IBM 3704 and 3705 Communications Controllers are programmable units. One of three programs can be generated to execute in 3704/3705 storage:

1. The Network Control Program (NCP) performs many of the teleprocessing line control and line servicing functions previously performed by the central processing unit.

2. The Emulation program (EP) permits existing teleprocessing systems, including VM/370, that use the IBM 2701, 2702, or 2703 Transmission Control Units or the Integrated Communications Adapter (ICA) of the System/370 Model 135, to run without change on the 3704/3705.

3. The Partitioned Emulation Program (PEP) allows the 3704/3705 to be divided so that both the NCP and EP can execute in one 3704/3705.

In this publication, the term 3704/3705 control program is used to refer to any of the three types of control programs: NCP, EP, or PEP.

VM/370 supports both the:

- IBM 3704 Communications Controller, Models A1-A4
- IBM 3705 Communications Controller, Models A1-D8

when attached to an IBM System/370 Model 135, 145, 155 II, 158, 165 II, or 168. Four terminals are supported: 1050, 2741, CPT-TWX 33/35, and 3270. You can generate any of three kinds of 3704/3705 control programs (NCP, EP, or PEP) to run under VM/370. The 3704/3705 must use emulation mode for the 3270 Information Display Systems.

The minimum amount of 3704/3705 storage required for a NCP or PEP control program is 48K and the minimum required by an EP control program is 16K.

## VM/370 SUPPORT OF THE 3704 AND 3705

The IBM 3704/3705 Communications Controllers can support:

- Up to 352 low-speed start-stop lines
- Up to 60 medium-speed synchronous lines
- Line speeds from 45.2 baud to 50.0K baud
- Modem capability within the 3704/3705
- Limited-distance "hard-wire" capability.
- 16K to 240K internal storage

VM/370 supports all three versions of the 3704/3705 control programs:

- Emulation Program (EP)
- Network Control Program (NCP)
- Partitioned Emulation Program (PEP)

VM/370's support of the 3704/3705 does not include:

- Remote 3704/3705 Communications Controllers
- Bisynchronous terminals if attached to lines in other than emulation mode.

## EMULATION PROGRAM (EP) WITH VM/370

The EP 3704/3705 control program under VM/370:

- Emulates 2701, 2702, and 2703 operations
- Attaches to a System/370 byte-multiplexer channel
- Supports up to 255 start-stop lines
- Supports up to 50 medium-speed sychronous lines

This support is equivalent to that provided in Release 1 of VM/370. However, Release 2 of VM/370 provides additional support:

- Service programs and special CMS commands allow you to easily generate the EP control program in a CMS virtual machine.
- The CP NETWORK command allows you to load or dump the 3704/3705 and provides for automatic dumping and reloading if a fatal error occurs.

## NETWORK CONTROL PROGRAM (NCP) WITH VM/370

The NCP 3704/3705 control program under VM/370 provides:

- A device-independent EBCDIC interface
- Communications line control handling
- Attachment to a System/370 byte-multiplexer, block-multiplier, or selector channel
- Block and character checking
- Block and message buffering
- Assembly and disassembly of multiple-block transmissions
- Line error recovery procedures
- Checkpoint/restart switching to a back-up processor
- User-written message processor routines

However, when an NCP 3704/3705 control program is under the control of VM/370:

- The CP DIAL command is not supported.
- The CP TERMINAL APL ON or APL OFF command line is not supported. If you issue the TERMINAL APL ON command at a terminal that is connected to VM/370 on a 3704/3705 line in NCP mode, you will not be able to execute your program and may have to IPL again to continue.

- NCP resources cannot be shared between VM/370 (the Control Program) and the virtual machines. A virtual 3704/3705 or 2701/2702/2703 is not supported.

- Special sign-on procedures are required (1) if you have the Multiple Terminal Access feature generated for your NCP control program and (2) if you have a 2741 terminal on an NCP-mode line. These procedures are described in the section "Special Sign-on Procedures for Lines in NCP Mode with MTA Feature."

A VM/370 nucleus that supports the NCP version of the 3704/3705 control program is smaller than one that supports the EP or PEP versions. Each line in NCP mode requires 48 bytes of free storage while each line in emulator mode requires 80 bytes of nucleus storage.

## PARTITIONED EMULATION PROGRAM (PEP) WITH VM/370

The PEP 3704/3705 control program under VM/370:

- Combines the 2701/2702/2703 Emulation Program and the Network Control Program

- Allows for the concurrent use of NCP and emulator interfaces

- Provides for programmable switching of lines between NCP mode and emulator mode

If you execute a PEP control program under the control of VM/370, you should know that:

- The CP DIAL command is supported for lines generated as emulator lines and lines generated to vary between emulator mode and NCP mode. If the DIAL command is issued for a line that may be varied (and, if that line is currently in NCP mode), that line is automatically varied to emulator mode.

- The TERMINAL APL ON or APL OFF command line is supported only for lines currently in emulator mode.

- Only 3704/3705 lines in emulator mode may be dedicated.

## GENERATING A VM/370 SYSTEM THAT SUPPORTS THE 3704 AND 3705

The VM/370 system that will run the 3704/3705 control program must be generated with:

- An RDEVICE macro describing the 3704/3705.

- One or more entries for the 3704/3705 control program in the System Name Table (DMKSNT).

- Space reserved on a CP-owned volume for a page-format copy of the 3704/3705 control program.

A detailed discussion on coding the RDEVICE macro, creating an entry in the system name table, and reserving DASD space for the 3704/3705 control program image can be found in the VM/370: Planning and System Generation Guide.

There are several commands and EXEC procedures to generate and load the 3704/3705 control program. These commands and EXEC procedures run in a CMS virtual machine. The commands are a part of the VM/370 system and are distributed with it.

Special versions of the IBM 3704/3705 Network Control Program Support Package for OS/VS, Order Numbers 5744-BA1 and 5744-BA2, are available from PID for use under VM/370. These versions of the 3704/3705 package contain two CMS EXEC procedures for generating and loading the 3704/3705 control programs that are not available in the standard OS/VS 3704/3705 support package, Order No. 5744-AN1. VM/370 supports the 5744-BA1 package for NCP, PEP, and EP, but supports the 5744-BA2 package for EP only.

A step-by-step procedure for generating the 3704/3705 control program can be found in the VM/370: Planning and System Generations Guide. Each EXEC procedure and command is described as it is used. The action required at each step is first summarized and then explained in detail.

## SAVE THE 3704/3705 CONTROL PROGRAM IMAGE ON DISK

If the SAVE operand on the GEN3705 command was specified during system generation, the SAVENCP command is automatically executed for you. If you did not specify SAVE on the GEN3705 command, you must issue the SAVENCP command yourself.

Note: The VM/370 command privilege class A, B, or C is required to use the SAVENCP command.

### THE SAVENCP COMMAND

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

```
SAVENCP   |   fname    [  (options.. [) ]]
          |
          |   Options:
          |
          |   [ ENTRY symbol | CXFINIT ]
          |
          |   [ NAME  ncpname| fname   ]
          |
          |   [ LIBE  libraryname | fname ]
```

where:

fname  is the filename of the LOADLIB file where the 3704/3705 control program load module resides; unless LIBE is specified, in which case, it specifies the member name of the image within the LOADLIB. This name is used as the 'ncpname' for the DIAGNOSE instruction, unless the NAME option is also specified.

Options

ENTRY symbol
is the external symbol of the entry point in the 3704/3705 control program load module. The default, CXFINIT, is the entry point of the standard NCP or PEP control programs. (The standard entry for the Emulation Program is CYASTART.) If the SAVE option of the GEN3705 command is specified, this symbol is set in the output EXEC file according to the Stage 2 input file.

NAME ncpname
is the 'ncpname' to be used when the DIAGNOSE parameter list is built. The ncpname specified must match an entry in the system name table. These entries are created with the NAMENCP macro when VM/370 is generated.

LIBE libraryname
is the filename of a load module library file, filetype LOADLIB, which contains the control program image as member 'fname'.

## EXECUTION OF THE SAVENCP PROGRAM

The DIAGNOSE X'50' instruction invokes the CP module DMKSNC to:

- Interpret the parameter list (CCPARM) built by SAVENCP.
- Check the parameter specifications against the NAMENCP macro for the 3704/3705 control program.
- Write the page-format image of the control program onto the appropriate CP-owned volume.

The parameter list for the DIAGNOSE instruction must start on a 4096-byte boundary. See the VM/370: Service Routines Program Logic for a description of the CCPARM control block.

When the DIAGNOSE X'50' instruction is executed, the module DMKSNC searches the DMKSNT module for a NAMENCP macro of the same 'ncpname' as the one in the CCPARM parameter list. The values specified in the parameter list are compared to those specified in the NAMENCP macro. If any parameters conflict, an error message is displayed at the terminal. If no error conditions are detected, DMKSNC starts to transfer the control program image from CMS virtual storage to the CP-owned volume specified in the NAMENCP macro. Successful completion of this process completes the generation of a 3704/3705 control program for VM/370 use.

## LOAD THE 3704/3705 CONTROL PROGRAM

The 3704/3705 control program is automatically loaded each time the VM/370 system is loaded, if the CPNAME operand was specified on the RDEVICE macro when VM/370 was generated and if the 3704/3705 is online. If the CPNAME operand was not coded, you must issue the CP NETWORK LOAD command line to load a 3704/3705 control program into the 3704/3705 Communications Controllers' storage.

## THE NETWORK LOAD COMMAND LINE

Use the NETWORK LOAD command to initiate the loading of an NCP, PEP, or EP control program into a 3704/3705 Communications Controller. The format of the NETWORK LOAD command line is:

```
| NETwork  | LOAD raddr ncpname |
```

where:

LOAD — initiates the control program load operation.

raddr — is the real address of the 3704/3705 to be loaded.

ncpname — is the name, defined by a NAMENCP macro, of the 3704/3705 control program image to be loaded into the 3704/3705 specified by 'raddr'.

## EXECUTION OF THE NETWORK LOAD COMMAND

The NETWORK LOAD command accesses the control program image using the information in DMKSNT created by the NAMENCP macro. If the 3704/3705 specified in the command is not in an "IPL Required" state at the time the command is issued, the message:

DMKNET461R CTLR raddr IPL NOT REQUIRED; ENTER "YES" TO CONTINUE:

appears at the terminal. If the reply to the message is other than "YES", the command terminates without loading the 3704/3705. Otherwise, the loader bootstrap routines are written to the 3704/3705 and loading starts. VM/370 does not run the "bring-up" test routines as a part of the load process. If these tests are to be made they must be run from a virtual machine with the 3704/3705 dedicated.

When the load of the control program image is complete, the command processor verifies that the 3704/3705 configuration described by the control program can be serviced by the VM/370 CP control blocks in storage. In the case of CPTYPE=NCP (or PEP), this involves creating (or refreshing) the control list associated with the 3704/3705 RDEVBLOK. The information necessary to do this is contained in the system pages at the beginning of the control program image on secondary storage.

## CONSIDERATIONS FOR LOADING 3704/3705 CONTROL PROGRAMS

When using the NETWORK LOAD command to load a 3704/3705 device with an NCP, EP, or PEP program, the following apply:

NCP - All active users are disconnected and active I/O operations are reset.

EP - All active I/O operations are reset; all active users are disconnected; dedicated devices are detached and released; dialed lines are released; devices that are enabled but not dedicated are reset; and binary synchronous communcations lines are reset.

PEP - All the above conditions apply; and, in addition, any lines switched from NCP mode to EP mode will revert back to NCP mode.

## SPECIAL CONSIDERATIONS FOR LOADING THE EP 3704/3705 CONTROL PROGRAM

If a 3704/3705 Emulation Program is automatically reloaded after a 3704/3705 failure, the system may loop after the restart. The message

```
DMKRNH463I CTLR raddr UNIT CHECK; RESTART IN PROGRESS
```

and two responses

```
CTLR raddr DUMP COMPLETE
CTLR raddr ncpname LOAD COMPLETE
```

indicate that the 3704/3705 has been reloaded. If the system loops after the second response, you must reset all emulator lines from the 3704/3705 control panel.

If the automatic dump feature is not enabled, one of the messages

```
DMKRNH462I CTLR raddr UNIT CHECK; IPL REQUIRED
DMKRNH464I CTLR 'raddr' CC=3; DEPRESS 370X "LOAD" BUTTON
```

indicates a 3704/3705 abnormal termination. The 3704/3705 Emulation Program must be reloaded via the NETWORK LOAD command. If the system loops when an attempt is made to enable the lines, you must reset all emulator lines from the 3704/3705 control panel.

The *IBM 3704 and 3705 Communications Controllers Operator's Guide* describes the procedure for resetting emulator lines from the 3704/3705 control panel in its "Generating Channel End/Device End with Emulator Program" section.

## SPECIAL CONSIDERATIONS FOR LOADING THE NCP AND PEP 3704/3705 CONTROL PROGRAMS

While the 3704/3705 Emulation Program may be loaded at any time, special care must be taken when loading a Network Control Program or Partitioned Emulation Program. The NETWORK LOAD command should not be used to load either an NCP or PEP except at VM/370 system IPL time, unless that same 3704/3705 control program was active just prior to the load (that is, unless it is reloaded immediately). A VM/370 system abnormal termination (code PTR007) may result if the 3704/3705 is loaded, for the first time, during normal operation with an NCP or PEP program. However, if an NCP or PEP 3704/3705 control program must be loaded during system operation, all resources must first be freed.

If there are active resources, the resources must be disabled and the NETWORK SHUTDOWN command must be issued before the operator can successfully issue the NETWORK LOAD command.

## LOGGING ON THROUGH THE 3704/3705

Because a 3704/3705 can support emulator-mode lines, NCP-mode lines, and lines that can be varied to either mode, and can also support a variety of terminals, the procedure for logging on is sometimes complicated. Use the following procedure to log on to VM/370.

First, turn the power on for your terminal and wait 15 to 30 seconds. Next, look for an online message at your terminal. If one of the following messages appears at your terminal

```
vm/370 online      xxxxxx xxxxxx

            -- or --

xxxxxx xxxxxx      vm/370 online
```

your terminal is a 2741 connected to VM/370 via a 2701/2702/2703 line or via a 3704/3705 line in emulation mode. You can then proceed with the normal logon procedure for your type of terminal, as described in VM/370: Terminal User's Guide.

If the messsage

```
vm/370 online
```

appears at your terminal, it is a 2741 connected to VM/370 via a 3704/3705 line in NCP mode without the Multiple Terminal Access feature, or it is a 1050, or CPT-TWX (Model 33/35) terminal in EP mode. You can proceed with the normal logon procedure for your terminal type. This procedure is described in the VM/370: Terminal User's Guide.

If you receive a message at your terminal in the form

```
xxxxxx xxxxxx
```

where the x's indicate that the message is unintelligible, your terminal is most likely connected to a 3704/3705 line in NCP mode that is defined for a different terminal type. For example, you may have an EBCD terminal on a line defined for Correspondence terminals. Use the @ (at sign) character to determine what kind of terminal you are using. If the @ character is an uppercase 2, your terminal is a Correspondence 2741; otherwise, it is an EBCD 2741.

If a 2741 terminal is connected to VM/370 via a 3704/3705 line in NCP mode, you must press the Return key before the "vm/370 online" message will appear at the terminal. If a terminal is connected to VM/370 via a 3704/3705 line in NCP mode, and with the Multiple Terminal Access (MTA) feature, the "vm/370 online" message does not appear at the terminal and, after approximately 15 seconds, the terminal locks and unlocks. You must perform a special sign-on procedure before continuing with the normal logon procedure.

The sign-on procedures for the terminals supported for use with the 3704/3705 under the control of VM/370 are summarized in the following paragraphs. If you have any further difficulties accessing VM/370 through a 3704/3705, see the VM/370: Terminal User's Guide and the 3704 and 3705 Generation and Utilities Guide for complete descriptions of the procedures summarized here.

## SPECIAL SIGN-ON PROCEDURES FOR 3704/3705 LINES IN NCP MODE AND WITH THE MTA FEATURE

Three sign-on procedures are described: the 2741, 1050, and TWX sign-on procedures for terminals on lines with the MTA feature. Once the sign-on procedure is completed, the message

```
vm/370 online
```

should appear at your terminal. You can then proceed with the normal logon procedure described in the VM/370: Terminal User's Guide.

### MTA Sign-on Procedures for IBM 2741

1. Dial the telephone number of the MTA line to be used for communicating with the controller.
2. When the keyboard unlocks, enter /".
3. Press the Return key.

If the "vm/370 online" message appears at your terminal, you have signed on successfully and may proceed with the normal logon. If no message appears at your terminal but your terminal unlocks, press the Return key in an attempt to get the "vm/370 online" message. However, if the type element moves back and forth, the sign-on procedure was unsuccessful; you must repeat steps 2 and 3 of the sign-on procedure.

### MTA Sign-on Procedure for IBM 1050

1. Dial the telephone number of the MTA line to be used for communicating with the controller.
2. When the Proceed light comes on, enter /".
3. Press the Return key.
4. Enter EOB.

If the "vm/370 online" message appears at your terminal, you have signed on successfully and may proceed with the normal logon. If no message appears at your terminal but your terminal unlocks, press the Return key in an attempt to get the "vm/370 online" message. However, if the type element moves back and forth, the sign-on procedure was unsuccessful; you must repeat steps 2, 3, and 4 of the sign-on procedure.

### MTA Sign-on Procedure for CPT-TWX Terminal

1. Dial the telephone number of the MTA line to be used for communicating with the controller.
2. Press the WRU (Where Are You) key within three seconds after the audible data tone begins.

If the typing mechanism does not "jump" within a few seconds, you have signed on successfully and may proceed with the normal logon. If the typing mechanism does jump, the sign-on procedure was unsuccessful; press the WRU key again or repeat both steps of the sign-on procedure.

## LOGGING ON AFTER AN NCP CONTROL PROGRAM HAS ABNORMALLY TERMINATED

If an NCP 3704/3705 control program (with the automatic dump and reload options previously set) abnormally terminates but VM/370 continues to run, VM/370:

1. Disconnects all the 3704/3705 users
2. Dumps the contents of 3704/3705 storage
3. Reloads the 3704/3705 control program
4. Enables the lines again

At this point, each user must log on again. Any user that does not log on within 15 minutes is logged off the system.

## APPLYING PTFS TO THE 3704/3705 LOAD LIBRARY

If necessary, it is possible to apply Program Temporary Fixes (PTFs) directly to the 3704/3705 load library using the VM/370 ZAP Service Program.

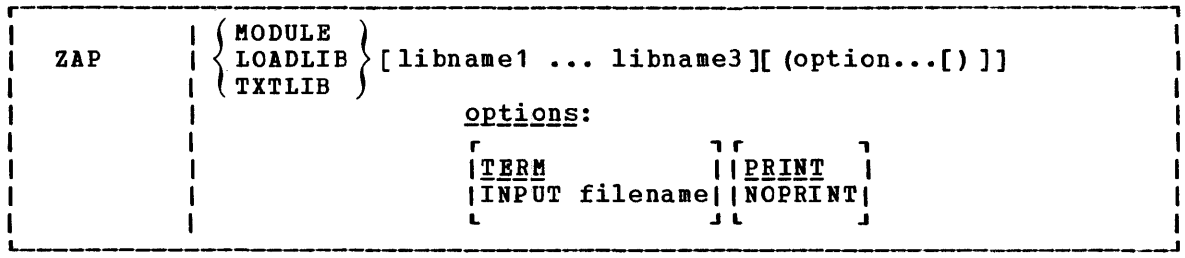
## THE ZAP SERVICE PROGRAM

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:




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.

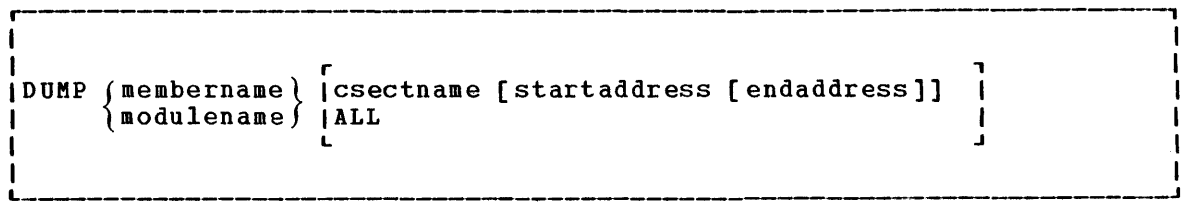
The program sets the NOGO switch on if a control record is found to be in error. A file cannot be modified once the NOGO switch is turned on. The next valid NAME record turns the NOGO switch off. This means that if the control record is the NAME record, all succeeding records are ignored until the next NAME, DUMP, or END record. For any other error, only REP control records that follow are ignored.

### DUMP Control Record

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:




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.

After you have generated a 3704/3705 control program, loaded it, and logged on, you may want to test the 3704/3705 control program. Several CP commands are provided to control the operation, check the status, and dump the contents of the 3704/3705. The NETWORK command loads and dumps any 3704/3705 control program. It also controls the operation of NCP and PEP 3704/3705 control programs, and remote 3270 resources connected to a 2701 or 2703, or a 3704/3705 in EP mode, or a 3704/3705 in PEP mode with lines in EP mode. The CP ENABLE, DISABLE, QUERY, DISPLAY, VARY, and HALT commands provide the same services for 3704/3705 EP control programs.

The NCPDUMP command formats and prints a dump of 3704/3705 storage. Use these commands to test the 3704/3705 control program.

## NETWORK COMMAND USAGE FOR REMOTE 3270

Use the NETWORK command to control remote 3270 resources. The NETWORK command may be used whether the remote 3270 resources are connected to a 2701, or 2703, or to a 3704/3705 in EP mode or PEP mode with lines in EP mode.

The operands that specifically affect remote 3270s are:

- SHUTDOWN (class A)
- POLLDLAY (class A,B)
- ENABLE (class A,B)
- DISABLE (class A,B)
- QUERY (class A,B)
- VARY (class A,B)

## RESOURCE IDENTIFICATION FOR REMOTE 3270S

Resources are defined as display stations, printer, and control units in the 3270 remote system. Whenever a remote 3270 resouce is referred to in the NETWORK command, and is identified in a VM/370 system message, the format is a four-character hexadecimal number, where the three low-order characters are the actual station resource ID, and the high-order hexadecimal character is a relative line code associated with a particular physical Binary Synchronous Communication (BSC) line that supports the remote 3270 terminal.

When the 3270 resource is referred to in the SET PFnn COPY command, the three-character hexadecimal number is the remote resource ID. The 3270 remote resource ID is a twelve-bit binary value which runs from a low value of zero to a high value determined by the number of CLUSTER and TERMINAL macro instructions defined for a communication line (RDEVICE). For example, assume that there are two BSC lines for remote 3270 display systems in a particular configuration. The first line to appear in the DMKRIO module would be assigned a lie code of zero, the second line would have a line code of one. Any other bisync lines used for other purposes are ignored. Resource ID 12 (decimal) on the first communication line would be represented as '000C'; resource ID 31 on the second communication line would be represented as '101F', and so on.

## NETWORK

Privilege Classes: A, B, and F

The CP NETWORK command loads, dumps, and controls the operation of 3704/3705 control programs and remote 3270 devices in the VM/370 environment. NETWORK:

- Causes 3704/3705 dump operations
- Initiates 3704/3705 load operations
- Enables or disables terminal resources
- Varies resources on or offline
- Alters the operating mode of a Partitioned Emulation Program line resource
- Halts a particular resource
- Ceases all 3704/3705 operations
- Queries and displays 3704/3705 resource status and storage
- Traces line activity to and from a 3704/3705 resource
- Varies the polling delay for 3270 remote binary synchronous lines
- Varies the polling delay interval of 3270 BSC lines.

### HOW TO USE THE NETWORK COMMAND

When using the NETWORK command to control the operation of the 3704/3705 Network Control Program (NCP), the NCP portion of the Partitioned Emulation Program (PEP), or remote 3270s on 2701 or 2703, or 3704/3705s in EP mode or in PEP mode with lines in EP mode. The operator must be aware of the different classes of resources defined at generation time for the 3704/3705 control program.

When operating with a 2701/2702/2703 or an Emulation Program (EP), there is only a single reference for each logon device, and that is the physical subchannel address for the telecommunications line. When operating with the NCP, the line is a separate entity, and the actual logon device is the terminal, which is also separately addressable. For a simple leased line configuration, there is one resource ID for each line, and one resource ID for each terminal (one terminal per line), alternating in numeric value.

The majority of the NETWORK command operations are performed for terminal resources. For example, NETWORK ENABLE, DISABLE, QUERY, HALT, VARY ONLINE, and VARY OFFLINE all operate for terminals. The NETWORK QUERY command line can be used to display the status of a line resource, but only when the 'NETWORK QUERY resource' command format is used. The possible states of a line resource are:

- OFFLINE (that is, inactive)
- ACTIVE
- EP-MODE (PEP only)

While the NETWORK VARY ONLINE and VARY OFFLINE command lines may be used for a line resource, they are primarily intended for use with terminal resources, because the state of the line changes automatically if the terminal is enabled or disabled. Also, NETWORK VARY EP and VARY NCP are valid only for line resources, and in this case the terminal resources change state when the line changes state.

The only way to tell which resources are lines and which are terminals is to examine the output from the first stage of the 3704/3705 control program generation. The installation system programmer (or whoever performs the 3704/3705 control program generation), should prepare a cross-reference list of resource IDs and their characteristics (such as, line or terminal, type of line, location, and so on) for the operations personnel. In summary, use

```
NETWORK ENABLE
NETWORK DISABLE
NETWORK QUERY ACTIVE
NETWORK QUERY FREE
NETWORK QUERY OFFLINE
NETWORK QUERY ALL
```

for terminals only. Use

```
NETWORK VARY EP
NETWORK VARY NCP
NETWORK TRACE resource
```

for lines only. And, use

```
NETWORK QUERY resource
NETWORK HALT
NETWORK VARY ONLINE
NETWORK VARY OFFLINE
```

for lines or terminals.

The format of the class A NETWORK command is:

where:

HALT resource attempts to terminate any active channel program on the specified resource (line or terminal). "resource" is a 4-digit hexadecimal identity of a 3704/3705 resource. The last three digits are the actual NCP resource ID. The first digit is a device sequence number associated with a particular 3704/3705. This device sequence number designates the relative position of the device in the DMKRIO module: the first 3704/3705 listed has a device sequence number 0, the second listed has a device sequence number 1, and so on.

```
SHUTDOWN ┌     ┐
         |raddr|
         | ALL |
         └     ┘
```

ceases telecommunications on 3704/3705 Communications Controllers or remote 3270 BSC lines. "raddr" is the real address of a 3704/3705 or BSC line. When "raddr" is specified, telecommunications are stopped on only the specified 3704/3705 or BSC line. When ALL is specified, telecommunications are stopped on all 3704/3705s or BSC lines.

No attempt is made to preserve line status or messages in the 3704/3705. Any virtual machines that depend on a 3704/3705 or BSC line for which the SHUTDOWN command is issued are placed in a disconnected state.

Responses:

NETWORK HALT

The normal response is:

```
DEVICE HALTED
```

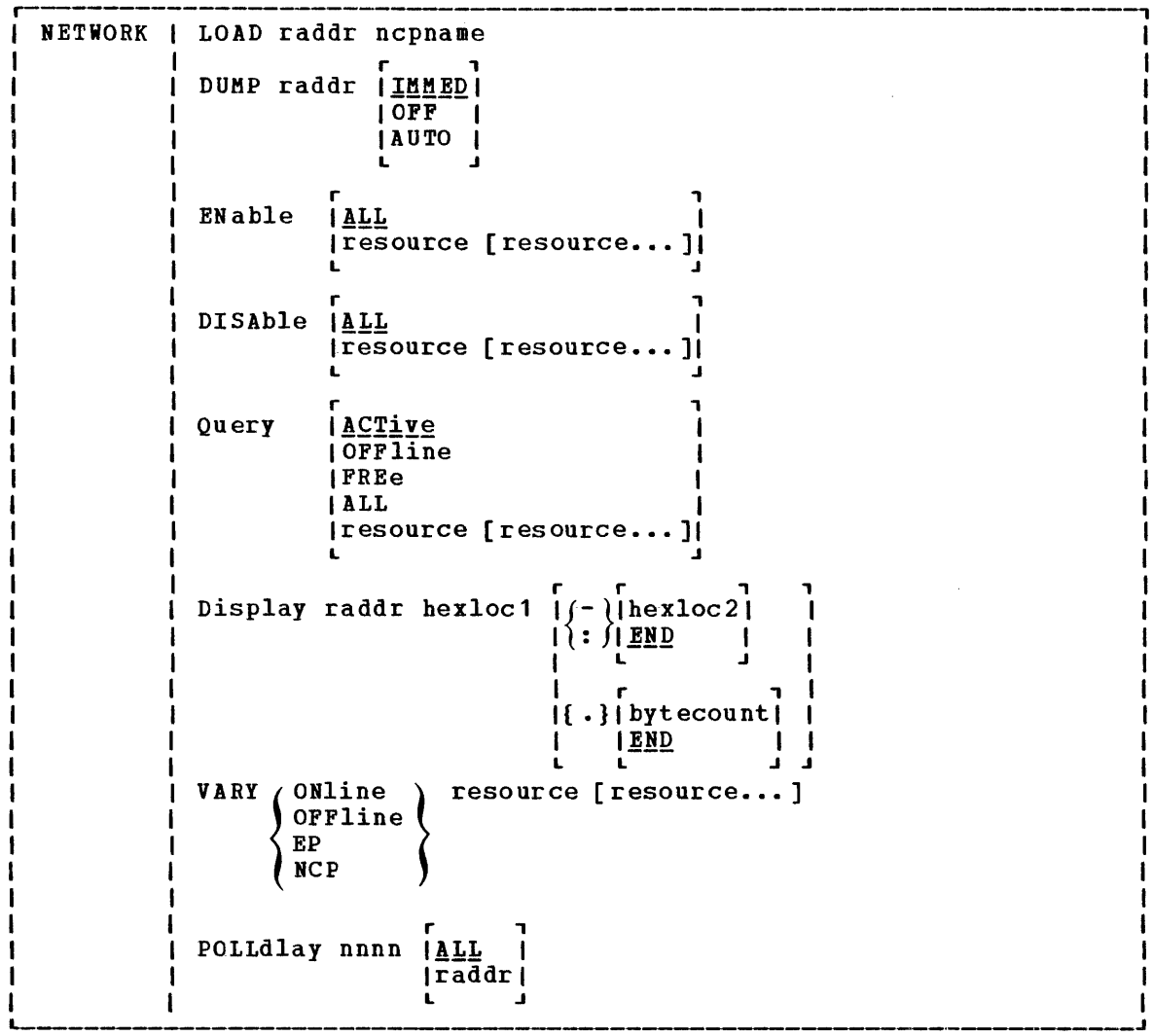
This response indicates that VM/370 has attempted to reset status and halt the device.

NETWORK SHUTDOWN

The normal response is:

```
COMMAND COMPLETE
```

The format of the class A and B NETWORK command is:




where:

LOAD raddr ncpname
: loads an NCP, PEP, or EP 3704/3705 control program. "raddr" is the real address of the 3704/3705 to be loaded. "ncpname" is the name, previously defined by a NAMENCP macro and saved on a CP volume, of the 3704/3705 control program image to be loaded into the 3704/3705 specified by "raddr".

DUMP raddr IMMED | OFF | AUTO
: dumps the contents of 3704/3705 storage for NCP, PEP, or EP 3704/3705 control programs. "raddr" is the real address of the 3704/3705 to be dumped.

  IMMED specifies that the 3704/3705 is to be dumped immediately. See the "NETWORK Dump Operations" section

in the VM/370: Operator's Guide for additional information.

OFF specifies that the 3704/3705 is not to be dumped automatically if the 3704/3705 control program abnormally terminates.

AUTO specifies the automatic dumping and reloading of the 3704/3705 if the 3704/3705 control program abnormally terminates.

```
ENABLE ┌──────────────────────┐
       │ALL                   │
       │resource [resource...]│
       └──────────────────────┘
```

activates 3704/3705 resources (terminals only) and remote 3270 resources for use by VM/370. ALL enables all the available resources. Resources may be enabled selectively by specifying the 4-digit hexadecimal identity of the terminal resource to be enabled. The last three digits are the actual resource ID. The first digit is a device or line sequence number associated with a particular 3704/3705 or BSC line. This device number designates the relative position of the device or line in the DMKRIO module: the first 3704/3705 or BSC line listed has a device sequence number 0, the second listed has a sequence number 1, and so on.

The resource specified must be a terminal device. The NETWORK ENABLE command first ensures that the associated line resource is activated, and then enables the terminal device. If the terminal is a display station, the enabling process also formats the display screen. Response from the enabled terminal devices causes the "vm/370 online" message to appear on the terminal.

```
DISABLE ┌──────────────────────┐
        │ALL                   │
        │resource [resource...]│
        └──────────────────────┘
```

disables 3704/3705 resources (terminals only) and remote 3270 resources. ALL disables all 3704/3705 terminals. To disable selective resources, specify the 4-digit, hexadecimal identity of the terminal resources to be disabled. The last three digits are the actual resource ID. The first digit is a device or line sequence number associated with a particular 3704/3705 or BSC line. This device number designates the relative position of the device or line in the DMKRIO module: the first 3704/3705 or BSC line listed has a sequence number 0, the second listed has a sequence number 1, and so on.

If any of the resources specified on the NETWORK DISABLE command are in use at the time the command is issued, they are not immediately disabled. However, as soon as the resource becomes free (usually after a LOGOFF command is issued), the resource is automatically disabled.

```
        ┌                        ┐
QUERY   |ACTIVE                  |
        |OFFLINE                 |
        |FREE                    |
        |ALL                     |
        |resource [resource...]|
        └                        ┘
```

displays the status of 3704/3705 resources (lines or terminals) and remote 3270 resources.

ACTIVE displays only of the resources (terminals, display and printer stations) that are active (those being used by VM/370 users).

OFFLINE displays only resources (terminals, display and printer stations) that are not available to VM/370 users.

FREE displays only resources (terminals, display and printer stations) that are not offline and also not currently in use.

ALL displays all the resources (terminals only) of all the 3704/3705s and all the resources (display and printer stations) of all the 3271/3275 control units.

"resource" displays the status of the resources (lines or terminals) whose 4-digit, hexadecimal identifiers are specified. The last three digits of resource are the actual resource ID. The first digit is a device or line sequence number associated with a particular 3704/3705 or BSC line. This device sequence number designates the relative position of the device or line in the DMKRIO module: the first 3704/3705 or BSC line listed has sequence number 0, the second listed has sequence number 1, and so on.

```
                        ┌    ┌        ┐    ┐
DISPLAY raddr hexloc1   |{-}|hexloc2|    |
                        |{:}|END     |    |
                        |    └        ┘    |
                        |    ┌          ┐  |
                        |{.}|bytecount|  |
                        |    |END      |  |
                        └    └          ┘  ┘
```

displays the contents of 3704/3705 storage. The data is displayed in fullwords. No EBCDIC translation is provided.

"raddr" is the real address of the 3704/3705 whose storage is to be displayed. "hexloc1" specifies the hexadecimal address of the start of the display and must be specified. To display more than one fullword, a - or : or . must be specified. "hexloc2" specifies the hexadecimal location of the end of the display. "bytecount" specifies the number of bytes to be displayed. END indicates that the display will continue until the end of storage is reached and is the default if "hexloc2" or "bytecount" are not specified.

VARY { ONLINE / OFFLINE / EP / NCP } resource [resource...]

varies the status of specified 3704/3705 resources or changes the operational mode of a PEP 3704/3705 control program. ONLINE places a resource (line or terminal) online; OFFLINE places a resource (line or terminal) offline. For remote 3270 resources, ONLINE and OFFLINE are the only valid operands.

EP changes the operational mode of the PEP 3704/3705 resource (line only) to emulation mode. NCP changes the operational mode of the PEP 3704/3705 resource (line only) to NCP mode.

"resource" is a 4-digit hexadecimal identity. The last three digits of resource are the actual resource ID. The first digit is a device or line sequence number associated with a particular 3704/3705 or BSC line. This device number designates the relative position of the device or line in the DMKRIO module: the first 3704/3705 or BSC line listed has a sequence number 0, the second listed has a sequence number 1, and so on.

POLLDLAY nnnn [ALL / raddr]

changes the duration of the polling delay interval for the bisync line to the value of nnnn. The address of the bisync line is raddr and nnnn is the decimal number in tenths of a second (not to exceed 9999) for the polling delay interval. If ALL is specified, the polling delay interval is set for all the 3270 remote lines.

The polling delay interval that is defined at system generation is two seconds.

Note: The polling delay interval is that period of time from the time a bisync line receives a negative response from a general polling sequence until the polling delay interval expires, or a message is sent to the station on the bisync line.

The polling delay interval minimizes unproductive polling and CPU meter time. In general, if no data or other communications is being received from the stations on the bisync line, the polling delay interval is started and control is given to the dispatcher.

Responses

NETWORK LOAD

CTLR raddr ncpname LOAD COMPLETE

The 3704/3705 'raddr' was successfully loaded with the control program 'ncpname'.

NETWORK DUMP

CTLR raddr DUMP COMPLETE

The 3704/3705 'raddr' was successfully dumped.

NETWORK ENABLE, NETWORK DISABLE, NETWORK VARY

The normal response is:

COMMAND COMPLETE

NETWORK HALT

The normal response is:

DEVICE HALTED

NETWORK QUERY

```
DEV rid LOGON AS userid
DEV rid DISABLE
DEV rid ENABLED
DEV rid OFFLINE

LINE rid ACTIVE
LINE rid EP-MODE raddr
LINE rid OFFLINE

DEV rid1 ENABLED, DEV rid2 ENABLED, DEV rid3 ENABLED,...
DEV rid1 DISABLE, DEV rid2 DISABLE, DEV rid3 DISABLE,...
DEV rid1 OFFLINE, DEV rid2 OFFLINE, DEV rid3 OFFLINE,...
```

where:

LOGON indicates that the resource is in use as a virtual machine operator console, by 'userid'.

DISABLE indicates that the resource is online but is not available for access to VM/370.

ENABLED indicates that the resource is available for user access to VM/370.

ACTIVE indicates that the line resource is online and has been activated. Terminals on the line may or may not be in use.

EP-MODE indicates that the line resource is a PEP line currently in emulation mode at real address 'raddr'.

OFFLINE indicates that the resource is inactive and unavailable for use.

rid is the real resource identifier.

userid is the user identifier.

raddr is the real device address.

The format of the class F NETWORK command is:

where:

TRACE BTU raddr
formats (in hexadecimal) each BTU (Basic Transmission Unit) sent to the specified 3704/3705, and each one received in response. The trace output is spooled to a virtual printer on the virtual machine of the user issuing the command. Each BTU is time-stamped at the time when it is traced, and the trace record consists of the 14-byte BTU header plus the first 4 bytes of the BTU data area.

"raddr" is the real address of the 3704/3705 to be traced.

TRACE resource
activates the NCP line trace facility for the specified resource (lines only). This facility provides a response BTU to the host whenever I/O activity for the specified resource exists. These responses are formatted in a manner similar to the BTU trace output, and they are likewise spooled to a virtual printer.

"resource" is a 4-digit hexadecimal identifier of a specific telecommunication line. The last three digits are the actual resource ID. The first digit is a device sequence number associated with a particular 3704/3705. This device sequence number designates the relative position of this device in the DMKRIO module. The first listed 3704/3705 would be designated as 0, the second 3704/3705 listed in the DMKRIO module would be designated as 1, and so on.

TRACE END terminates the trace operation.

Note: NETWORK TRACE can be set active for only a single physical 3704/3705 at any one time.

Responses:

TRACE STARTED

is the response given to the BTU and the resource operand.

COMMAND COMPLETE

is the response given to the END operand (trace termination).

## NCPDUMP SERVICE PROGRAM AND HOW TO USE IT

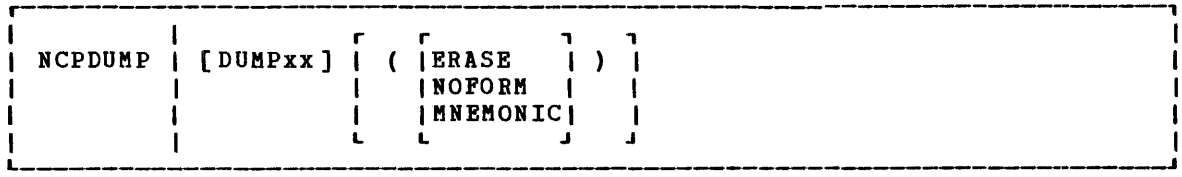
NCPDUMP is a CMS command that processes CP spool reader files created by 3704/3705 dump operations.

The NCPDUMP command:

- Creates the CMS NCPDUMP file from the spool file.
- Formats the dump.
- Prints the dump.
- Erases the CMS NCPDUMP file (if specified) after printing it.

Although NCPDUMP is a CMS command, its effective use is restricted to a specific user identified by the SYSDUMP operand of the SYSOPER macro in DMKSYS. The operation of NCPDUMP is similar to VMFDUMP operations. A general description of the NCPDUMP operation follows the command description.

The NCPDUMP command has the following format:




where:

DUMPxx is the filename of the CMS file containing a 3704/3705 control program dump. This dump was created by a previously invoked NCPDUMP command with the ERASE option not specified.

ERASE erases the current dump file or a previously created CMS dump file (DUMPxx).

NOFORM suppresses the formatting of the control block.

MNEMONIC includes the 3705 Assembler mnemonic operation codes in the printed output.

This command is also described in the VM/370: Operator's Guide.

### USING THE NCPDUMP COMMAND

The NETWORK command invoked with the DUMPxx operand produces CP files. These CP files contain the 3704/3705 storage dump and are spooled reader input assigned to a system-designated user. The CMS NCPDUMP command invoked by this user formats (optionally) and prints the contents of these files.

A CMS file, with a filename DUMPxx, and a filetype of NCPDUMP, is created and the original spooled dump reader file (created when the NETWORK DUMP command was issued) is deleted. If ERASE was specified on the NCPDUMP command line, the CMS dump file is also erased; otherwise, it is saved.

A maximum of ten dumped spooled files can be processed and saved, and later recalled if necessary, by the system assignment of the xx suffix to the CMS-created DUMPxx filename. 'xx' is a decimal number from 00 to 09, depending on any existing files of similar name. For example, if the files 'DUMP00 NCPDUMP' and 'DUMP01 NCPDUMP' already exist, the new file is called 'DUMP02 NCPDUMP'. The file thus created is retained for later use unless the ERASE option is specified, in which case the file is erased immediately following the dump printing.

# Part 5. Remote Spooling Communications Subsystem (RSCS)

Part 5 contains the following information:

- Introduction to RSCS
- Structure of RSCS virtual storage
- Functional information
- Logging I/O activity

The Remote Spooling Communications Subsystem (RSCS), a component of VM/370, provides telecommunication facilities for the transmission of bulk files between VM/370 users and remote stations. RSCS is a single purpose operating system for a virtual machine, dedicated to the management of files spooled to it by VM/370 users or transmitted to it by remote stations via communication lines. Remote stations can submit files to a VM/370 user or CMS Batch facility for processing and receive printer and punch output in return. VM/370 users can submit job streams to a remote HASP- or ASP-type batch processor. Remote stations can send printer and punch files to other remote stations.

## LOCATIONS AND LINKS

Under RSCS, all remote locations as well as the local RSCS virtual machine are assigned a one- to eight-character alphameric location identification. The transmission path between the RSCS virtual machine and any single remote station is defined as a link. A link has certain attributes that make up a link definition and these attributes are assigned at system generation time or dynamically via the RSCS DEFINE command. A link definition consists of a linkid (the location identifier of the remote station), the type of remote station, the line address to be used or transmission, the class of files to be processed, and other information unique to the link. RSCS maintains a table of link definitions (link table) in the module DMTSYS. A maximum of 64 links may be defined of which any 16 may be active at any one time.

## REMOTE STATIONS

A remote station, in the context of RSCS, is any terminal or system on the other end of the link from the RSCS virtual machine. The RSCS virtual machine is also referred to as the local RSCS station. RSCS supports two general types of I/O configurations used as remote stations.

Nonprogrammable remote terminals, such as the IBM 2780, are I/O configurations where the line protocol necessary for them to function as remote stations is provided by the hardware. These devices are managed by the Nonprogrammable Terminal (NPT) line driver of RSCS.

Programmable remote stations, such as the IBM System/3 and System/360, are IBM processing systems with attached binary synchronous communications adapters. These systems must be programmed to provide a MULTI-LEAVING line protocol necessary for their devices to function as remote stations. For a detailed description of MULTI-LEAVING, see "Appendix B: MULTI-LEAVING." 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.

## VM/370 SPOOL SYSTEM INTERFACE

RSCS uses the VM/370 spool system to interface with VM/370 users.

When a user generates a file to be transmitted to a remote location by RSCS, he must comply with two requirements. The file must be spooled to the RSCS virtual machine and the spool file tag associated with the file must contain, as the first entry, the linkid (location identifier) of the remote station to which the file is being transmitted.

When a remote station transmits a card file to RSCS, the file must be preceded by an ID card containing the userid of the virtual machine that is to receive the file. RSCS punches the file on a virtual punch and spools it to the appropriate virtual machine. If the userid is that of the RSCS virtual machine and the ID card also contained valid tag data, RSCS will retrieve the file from the VM/370 spool system and forward it to the remote station designated by the linkid in the tag data.

## RSCS COMMAND LANGUAGE

The RSCS command language provides the RSCS virtual machine operator with the following capabilities:

- Manipulate the status, transmission priority, class and order of files owned by the RSCS virtual machine.
- Initialize, suspend or terminate transmission of files to remote terminals or stations.
- Reposition or restart files currently being transmitted.
- Send or forward messages and commands to remote terminals and stations.
- Query file, link or system information.
- Monitor link activity for any remote location.

A summary of the RSCS commands is shown in Figure 50; for a full description and format, refer to "Appendix A: Remote Spooling" Communications Subsystem Commands" in the VM/370: Remote Spooling Communications Subsystem (RSCS) User's Guide.

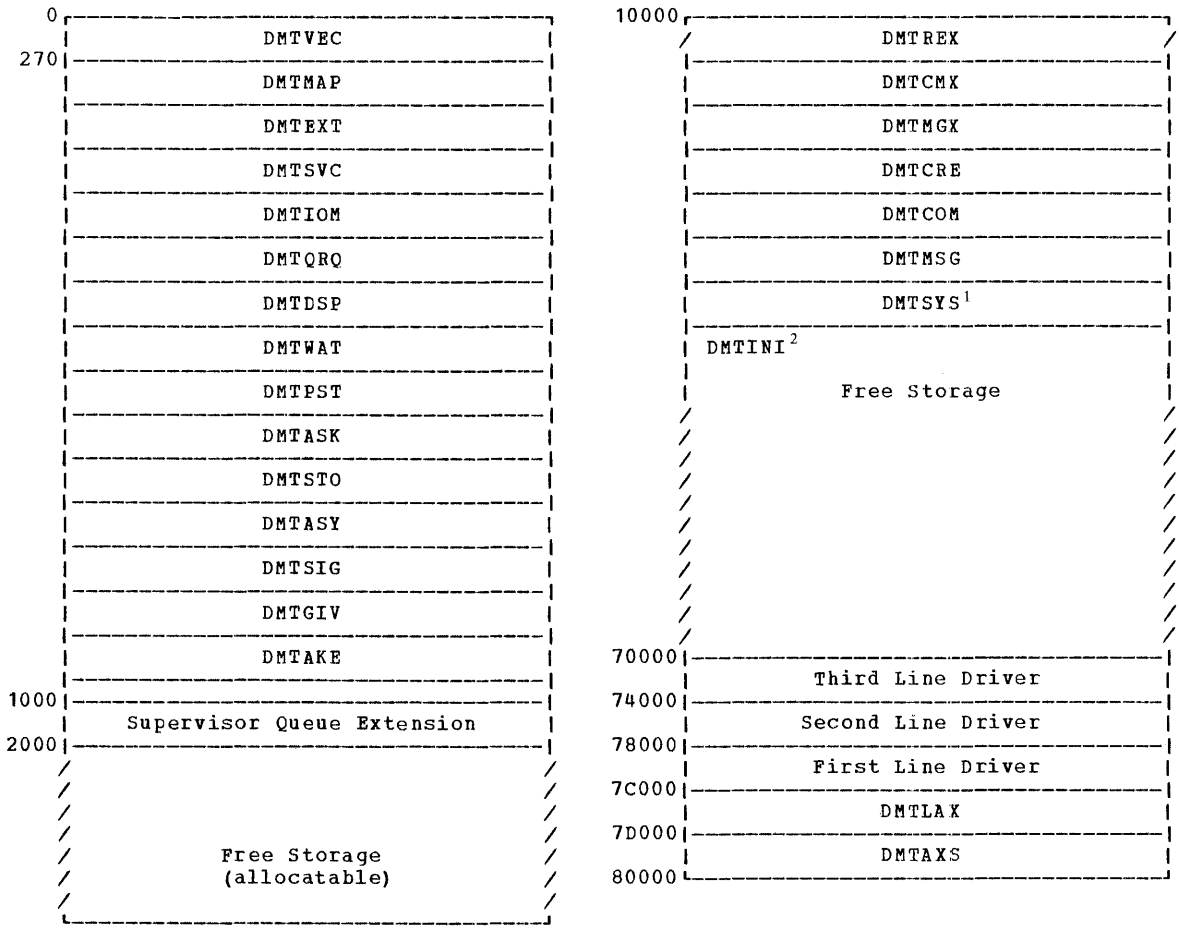
| Command Name | Function |
|---|---|
| BACKSPAC | Restart or reposition in a backward direction the file currently being transmitted. |
| CHANGE | Alters one or more attributes of a file owned by RSCS. |
| CMD | Control certain functions performed by a remote system, or control the logging of I/O activity on a specified link. |
| DEFINE | Temporarily add a new link definition to the RSCS link table or temporarily redefine an existing link. |
| DELETE | Temporarily delete a link definition from the RSCS link table. |
| DISCONN | Place RSCS in disconnect mode and optionally direct output to another virtual machine. |
| DRAIN | Deactivate an active communication link. |
| FLUSH | Discontinue processing the current file on the specified link. |
| FREE | Resume transmission on a communication link previously in HOLD status. |
| FWDSPACE | Reposition in a forward direction the file currently being transmitted. |
| HOLD | Suspend file transmission on an active link without deactivating the line. |
| MSG | Send a message to a local or remote station. |
| ORDER | Reorder files enqueued on a specific link. |
| PURGE | Remove all or specified files from a link. |
| QUERY | Request system information for a link, a file, or for the system in general. |
| START | Activate a specified communication link. |
| TRACE | Monitor line activity on a specified link. |

Figure 50. RSCS Command Summary

A subset of the RSCS commands is available to the remote station operators. In general, the remote operator can issue only those commands that offset his specific link. The commands are punched, one per card, and entered at the remote card reader. Commands from remote stations are only accepted before the ID card of an input card file or after the file has been completely processed (end-of-file generated).

## Structure of RSCS Virtual Storage

RSCS virtual storage is made up of fixed address storage areas, supervisor service routines, system service modules, line driver modules, and available free storage for active tasks. Figure 51 shows




[1] The DMTSYS module can vary in size depending on the number of macros specified when the RSCS system was generated. Free storage starts on the first page boundary following the end of DMTSYS.
[2] The DMTINI module is loaded at the beginning of the free storage area. After initialization, the storage it occupied is freed and becomes part of free storage.

Figure 51. RSCS Storage Allocation

## RSCS SUPERVISOR (X'00000' TO X'01000')

The first 4K bytes of storage contain hardware and supervisor-defined constants, control areas, and supervisor service routines.

DMTVEC: The first 512 bytes of DMTVEC are defined by System/370 architecture and contain hardware-defined constants. This area is initialized by the DMTINI routine at initial program load time.

The rest of DMTVEC, 112 bytes, contains supervisor-defined addresses and constants used for dispatching, storage mapping, queue management, and task management.

DMTMAP: The supervisor storage area contains the main storage map and the first extent of the supervisor queue.

The main storage map is a table comprising one byte for each page in accessible main storage. Each byte displacement in the table implies an associated main storage number.

The supervisor queue is a chain of 16 byte elements, formatted during initialization, maintained by the DMTQRQ routine, and containing the status information for all system tasks running or waiting to be dispatched. The length of this chain is such that the service routines that follow are located at the end of the page of storage.

Supervisor Service Routines - the rest of the supervisor contains service routines that provide services to other system tasks. The thirteen routines and their functions are:

DMTEXT - Handle external interruptions
DMTSVC - Handle SVC interruptions
DMTIOM - Handle I/O interrupts and requests
DMTQRQ - Manage the supervisor status queue
DMTDSP - Dispatch eligible tasks
DMTWAT - Suspend task execution
DMTPST - Signal completion of an event
DMTASK - Create and delete system service tasks
DMTSTO - Reserve and release main storage pages
DMTASY - Provide asynchronous task-to-task exits
DMTSIG - Interrupt a task, immediately, for an ALERT request
DMTGIV - Enqueue a GIVE request element for another task
DMTAKE - Process a GIVE request element

## SUPERVISOR QUEUE EXTENSION (X'1000' TO X'2000')

The supervisor queue extension is a chain of 16 byte elements that provide an extension to the supervisor queue located in DMTMAP.

## FREE STORAGE (X'2000' TO X'10000')

This area of free storage is managed by the DMTSTO module. System tasks reserve and release virtual storage in full page increments as required.

## SYSTEM CONTROL TASK (X'10000' TO END OF DMTSYS)

The system control task consists of five executable and two non-executable modules. Their functions are:

DMTREX - Handle console I/O; process request elements for service routines; terminate system service and line driver tasks.

DMTCRE - Start a line driver task and create the DMTAXS and DMTLAX tasks during initialization.

DMTCMX - Handle all console functions.

DMTMGX - Build and forward message request elements.

DMTCOM - Perform miscellaneous system service functions.

DMTMSG - Table of message texts and codes.

DMTSYS - Link table, file tag storage area, tag queue pointers, and switched line port table.

## FREE STORAGE AND LINE DRIVERS (PAGE BOUNDARY FOLLOWING DMTSYS TO X'7C000')

This area of free storage is also managed by DMTSTO. In addition to providing storage for system tasks, it is used for line driver storage. For each active link that is initialized by DMTCRE, a copy of a DMTSML or DMTNPT line driver is brought into virtual storage. Line driver storage is assigned downward from X'7C000', in four-page increments. Free storage for system tasks is assigned upwards from the page boundary following DMTSYS, in one-page increments.

## LINE ALLOCATION TASK (X'7C000' TO X'7D000')

The DMTLAX module allocates a line port to a link when its line driver task is started. If a line address has been previously assigned in the link definition or is specified in the START command, DMTLAX verifies that the line is for a valid device type and is not already in use. If a line address has not been previously assigned and is not specified in the START command, DMTLAX scans the table of switchable line ports for an available line and assigns it to the link's line driver task. If a line is not available or is incorrectly specified, an error message is issued to the RSCS operator.

## SPOOL FILE ACCESS TASK (X'7D000' TO X'80000')

The DMTAXS module accepts files from the VM/370 spool system and maintains the queues of main storage file tag slots; executes the ORDER, CHANGE, and PURGE commands; and opens and closes input and output VM/370 spool files.

The RSCS virtual machine performs certain basic functions as it manages the transmission of files between the host VM/370 and remote locations. These functions include:

- Virtual storage management
- File management
- Task-to-task communication
- RSCS command processing
- RSCS message handling
- Interruption handling

## VIRTUAL STORAGE MANAGEMENT

The RSCS supervisor controls virtual storage in blocks of either 4096 bytes (page size) or in 16 byte queue elements. Tasks running under the supervisor obtain their working storage area in page size blocks and then allocate variable size blocks as their functions require.

### PAGE ALLOCATION

Page allocation is performed by the supervisor service routine, DMTSTO. A storage allocation map, 256 bytes in length, is located in the supervisor area and is pointed to by MAINMAP in the DMTVEC data area. Each byte represents a page of virtual storage and contains X'00' if the page is free. MAINSIZE, also in DMTVEC, contains the total number of pages defined for the particular RSCS virtual machine.

When a task requires a page of storage, it first searches the storage allocation map for a free page (X'00'). The page number is placed in register 1 and a call to DMTSTO reserves the page. DMTSTO replaces the storage map byte with the one-byte TASKID assigned to the calling task by the supervisor. To release storage, a task has only to clear the appropriate bytes in the storage map.

### QUEUE ELEMENT MANAGEMENT

With the exception of a few words of low address storage used by the dispatcher, the rest of the supervisor status information is stored in chains of 16-byte queue elements managed by DMTQRQ. The first extent of these queues is in the supervisor and occupies the area between the main storage allocation map and DMTEXT. A supervisor queue extension area, one page in length, is located at X'1000'. Queue elements are dequeued from the free element queue pointed to by FREEQ in DMTVEC and enqueued on one of the active queues (TASKQ, MPXIOQ, SELIOQ, IOEXTQ, EXTQ, ALERTQ or GIVEQ). When the queue element is released, it is returned to the free element queue.

## FILE MANAGEMENT

RSCS uses the VM/370 spool file system to interface with VM/370 users. A user who generates a file intended for transmission to a remote location must spool the file to the RSCS virtual machine via the CP SPOOL command. In addition, he must also enter the identification of the remote location into the spool file tag area via the CP TAG command.

A remote station submitting a file to RSCS for transmission to another remote location must meet the same requirements as a VM/370 user. The ID card that precedes the input card file being transmitted to RSCS must include the userid of the RSCS virtual machine and a tag field containing the location identifier of the remote station that is to receive the file.

A remote station submitting a file destined for a VM/370 user need only specify that user's userid on the ID card.

When the RSCS virtual machine is initially logged on, one of the first tasks that is started is the Spool File Access task, DMTAXS. Two main functions of DMTAXS are: to provide access to the VM/370 spool file system, and to manage the queues of tag slots used by RSCS to control the status and flow of files through the system.

### TAG SLOT QUEUES

The DMTAXS task in RSCS manages a file tag storage area pointed to by TTAGQ in DMTVEC. This area is made up of a fixed number of tag slots, each containing 108 bytes. The total number of slots is determined, at the time RSCS is generated, by the value specified in the GENTAGQ macro. The number of slots reserved for each link is part of the link definition stored in the RSCS link table. The contents of each file tag include file attributes from the file's SFBLOK and transmission destination and priority from the associated spool file tag.

File tags are chained on one of four types of queues:

- The active input queue, pointed to by TAGACIN in TAGAREA, contains the tags for those files that are currently being processed for transmission to remote locations.
- The active output queue, pointed to by TAGACOUT in TAGAREA, contains the tags for those files that are currently being received from remote locations.
- An inactive file queue exists for each link that has one or more files waiting to be transmitted. Each link's file tag queue is pointed to by the LPOINTER field in the corresponding link table entry.
- The free slot queue, pointed to by TAGAFREE in TAGAREA, is made up of all the slots not currently on any of the other tag slot queues.

### SPOOL FILE ACCESS

The Spool File Access task, DMTAXS, uses the "retrieve subsequent file descriptor" option of the CP DIAGNOSE X'014' command to access the spool

file block (SFBLOK) and spool file tag for each of the files enqueued on the RSCS virtual reader.

Using the location identifier in the spool file tag, DMTAXS interrogates the link table entry for the specified link to determine if a tag slot is available. If so, a tag is built, using information in the SFBLOK and spool file tag, and then enqueued on the link's chain of inactive files pointed to by LPOINTER in the link table entry. If a tag slot is not available, the file is placed in a pending status and the link table entry count of pending files (LPENDING) is incremented by one. Pending files are added to the inactive file queues as slots become available.

When a line driver task is started for a link via the RSCS START command, the highest priority file on that link's inactive queue (LPOINTER) is dequeued and placed in the system's active input queue (TAGACIN). The file's tag and first spool buffer are then passed to the line driver task for transmission. Any additional spool buffers for that file are directly obtained by the line driver task.

## TASK-TO-TASK COMMUNICATION

RSCS provides two methods of task-to-task communications: GIVE/TAKE requests, and ALERT requests.

GIVE/TAKE requests are issued by lower priority tasks, such as line drivers, to request a service from a higher priority task, such as a supervisor service routine. The requesting task builds a request table containing the name of the task that is to perform the service, along with pointers to a request buffer containing the data required for the service. If appropriate, a pointer to a response buffer is also supplied. This information is passed to the DMTGIV module. DMTGIV builds a GIVE element that points to the requestor's request table and chains it on the GIVE element queue for execution.

Service tasks pass control to DMTAKE whenever they complete the execution of a particular service. DMTAKE locates the GIVE element for the service that was just completed, passes any response data back to the requestor via the response buffer, locates the next GIVE element for that service task, and passes the corresponding request table data to the service task for execution.

ALERT requests are issued by high priority tasks for services to be performed by a lower priority task. These requests are not queued; the lower priority task is executed as soon as it is received. ALERT requests are handled by the DMTSIG module.

## RSCS COMMAND PROCESSING

The primary command processor in RSCS is the DMTCMX module of the system control task. DMTCMX receives commands either as a result of a console read started by the DMTREX module in response to attention interruption from the RSCS operator console, or through a GIVE request pointer to a command element, provided by an active line driver task.

The DEFINE, DELETE, DISCONN, QUERY and START commands are processed entirely by the system control task, as they may involve the referencing and updating of the system status tables (DMTSYS).

For the CHANGE, PURGE and ORDER commands, DMTCMX builds a formatted table called a command element and passes it, via an ALERT request, to the DMTAXS task for execution.

The BACKSPAC, CMD, DRAIN, FLUSH, FREE, FWDSPACE, HOLD, MSG, and TRACE commands are passed to the line driver task for the associated active link via a command element and ALERT request.

## RSCS MESSAGE HANDLING

Messages can occur in response to a command or spontaneously as a result of a system malfunction.

The task that originates the message passes the message number and the variable portion of the message text to the message handler, DMTMGX. DMTMGX obtains the fixed portion of the message text and routing information from the DMTMSG module and issues the message to the appropriate operator.

Messages can be addressed to the local RSCS operator, remote station operator, local VM/370 virtual machine, VM/370 system operator, or combinations of these.

Messages directed to the VM/370 system operator or VM/370 user are issued via the CP MSG command using the Virtual Console Function of the Diagnose interface. Messages for the local RSCS operator are enqueued for output by DMTREX. Messages for the remote station operator are presented to the line drivers for the associated links via an RSCS MSG command element and ALERT request.

## INTERRUPTION HANDLING

Three types of interruptions are handled by the supervisor service routines: external interruptions, SVC interruptions, and I/O interruptions.

### EXTERNAL INTERRUPTIONS

External interruptions are handled by the DMTEXT module. Each bit of the external interruption code (bytes 16-31 of the external old PSW in low storage) is inspected. When a bit is set to one, a scan of the external exit request queue is made to locate the first requested exit for the bit that was set. If one is found, the exit is taken; otherwise, processing continues until the entire interruption code has been inspected.

### SVC INTERRUPTIONS

The DMTSVC module receives control directly on an SVC interruption. RSCS uses the SVC interruption to "freeze" the execution of a task while it is waiting for the results of some service that it has requested of another task. The left half of the SVC old PSW is moved to the left half of the resume PSW in the task's save area; the right half is loaded

with the contents of register 14 (resume PSW address). The register contents at interruption time are also stored in the task's save area.

DMTSVC returns control to the caller by setting register 14 to the address of the task element of the "frozen" task and loading a PSW with all mask bits set off (except machine check) and execution address as stored in the SVC old PSW.

## I/O INTERRUPTIONS

I/O interruptions are handled by the DMTIOM module at entry point DMTIOMIN. DMTIOM first searches for an active I/O request element on the appropriate queue (MPXIOQ or SELIOQ). If one is found, the I/O request table is updated to reflect the new status. If this is not the final interruption, control is immediately returned to the dispatcher. If the I/O has completed without unit check, the synch lock in the I/O table is posted; and, if there is no further I/O enqueued for that subchannel, control is passed to the dispatcher. If I/O is enqueued for that subchannel, it is started.

If the I/O has completed, but there was a unit check and automatic sense was requested, the sense channel program is built in a new element and the new element is chained to the request element. The sense operation is started and if not completed immediately, control is passed to the dispatcher.

If an active I/O request element was not found, the asynchronous I/O exit queue (IOEXITQ) is scanned for a matching device address. If found, the asynchronous exit is taken.

If neither an active I/O request element nor an asynchronous exit request element is found, the interrupt is ignored and control is passed to the dispatcher.

## Logging I/O Activity

The RSCS component of VM/370 contains a facility for logging all I/O activity on a particular teleprocessing link. This logging feature can be utilized if a problem arises where tracing I/O activity on a line becomes a necessity.

The RSCS operator can turn the feature on and off by issuing the RSCS CMD command with the LOG or NOLOG operand. The format of the CMD command, when used to control logging, is as follows:

```
CMD    linkid {LOG   }
              {NOLOG}
```

where:

linkid is the location identifier for the link on which logging is to be performed.

LOG is the keyword that starts the logging of I/O activity.

NOLOG is the keyword that stops the logging of I/O activity.

The logging output is a printer spool file containing a one-line record for each I/O transaction on the teleprocessing line. A transaction is defined as any read or write of a teleprocessing buffer. When logging is turned off, the output is automatically spooled to a printer. The distribution code on the printer output is the linkid that was specified in the CMD command.

The output log record is printed in hexadecimal notation unless otherwise noted.

### THE SML LOG RECORD

The contents of the SML log record are as follows:

1-42 The first 21 bytes of the teleprocessing buffer, including BSC bytes, MULTI-LEAVING bytes, and enough initial bytes of data to fill the field.

44-57 For read I/O: the last seven bytes of the CSW. For write I/O: The first seven bytes of the SML buffer header that is used internally by SML but not transmitted.

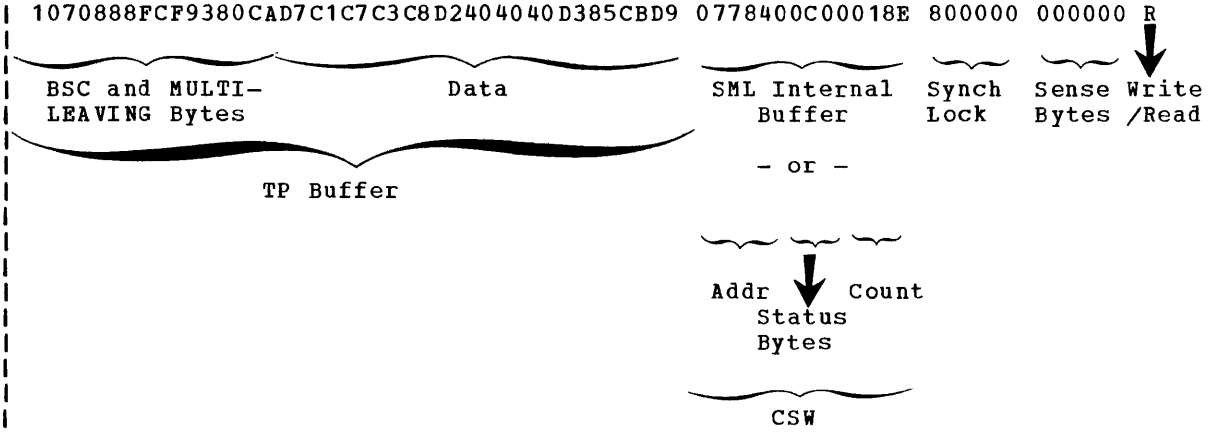
59-64 The first three bytes of the RSCS I/O synch lock for this transaction.

66-71 The first three sense bytes, if any.

73 "R" for read I/O; "W" for write I/O (alphabetic character).

The fields of the record are separated by blanks. The following are samples of read and write log records for SML:

```
1002878FCF938089C2D3D484F9D9F56BD9F66B7EC1 00071198015C08 000000 000000 W
1070878FCF938089C2D3D484F9D9F565D9F66B7EC1 0778400C00018E 800000 000000 R
1002888FCF9380CAD7C1C7C3C8D2404040D385CBD9 000714C8016008 000000 000000 W
1070888FCF9380CAD7C1C7C3C8D2404040D385CBD9 0778400C00018E 800000 000000 R
```




## THE NPT LOG RECORD

The contents of the NPT log record are as follows:

1-42 the first 14 bytes of the teleprocessing buffer including BSC bytes, 2 alphabetic dashes, and the last 6 bytes of the teleprocessing buffer.

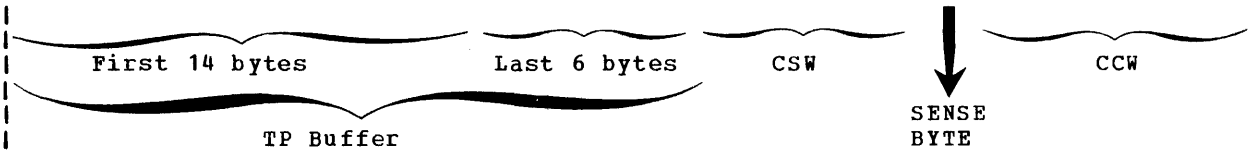
44-57 For read I/O: the last seven bytes of the CSW.
For write I/O: blank, not applicable

59-60 The sense byte, if any.

63-78 The CCW associated with the I/O operations.

The fields of the record are separated by blanks. The following are samples of read and write log records for NPT:

```
2D                                              0797E00C000003     02079BF820000004
1070                                            0798280C0001AC     0207987A60000002
1002E2C9C7D5D6D5404040E3C5E2--404040404003                         02079BF820000200
1061                                            0798280D0001FF     0107987A60000002
37                                                                 02079BF820000200
--                                              0797E00E000004 01  02079BF820000004
--                                              0797E00E000004 01  02079BF820000004
```




Note: The dashes in record positions 1 and 2 indicate that there was no data transfer for that I/O transaction.

## CONTROL REGISTERS

The control registers are used to maintain and manipulate control information that resides outside the PSW. There are sixteen 32-bit registers for control purposes. The control registers are not part of addressable storage.

At the time the registers are loaded, the information is not checked for exceptions, such as invalid segment-size or page-size code or an address designating an unavailable or a protected location. The validity of the information is checked and the errors, if any, indicated at the time the information is used.

Figure 52 is a summary of the control register allocation and Figure 53 lists the facility associated with each control register.

Figure 54 is a description of the EC (Extended Control) PSW.

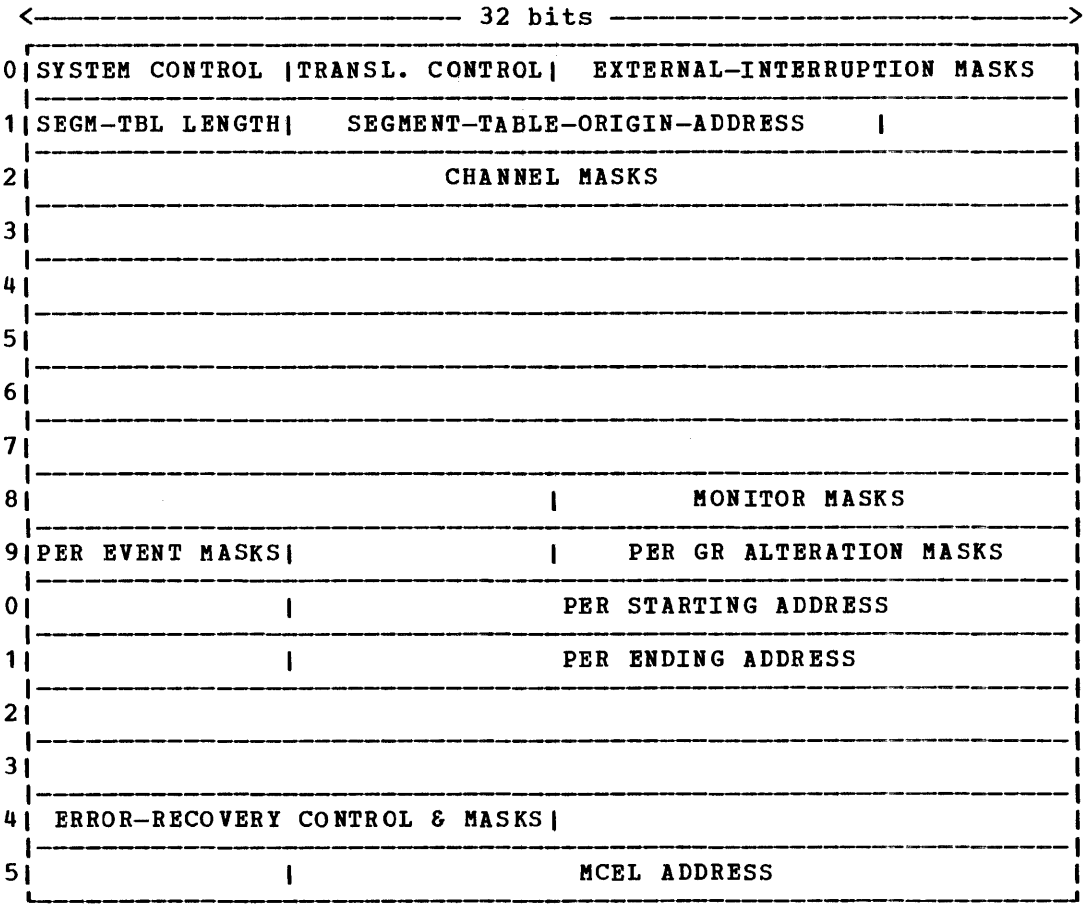


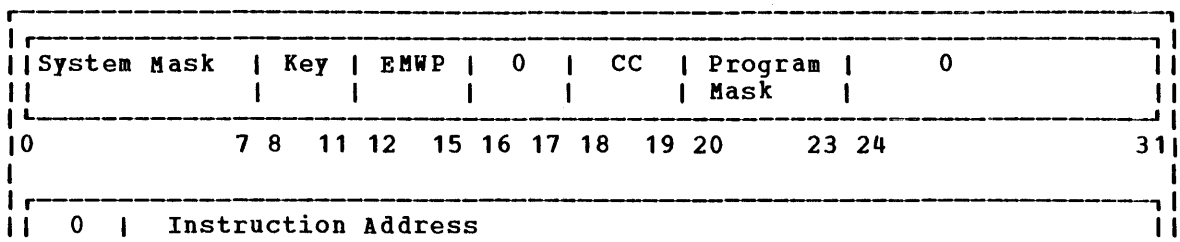

Figure 52. Control Register Allocation

| Word | Bits | Name of Field | | Initial Value |
|---|---|---|---|---|
| 0 | 0 | Block-Multiplexing Mode | Block-Multiplexing Control | 1 |
| 0 | 1 | SSM Suppression | Extended Control | 0 |
| 0 | 8-9 | Page Size[1] | Dynamic Addr. Translation | 10 |
| 0 | 10 | Reserved | Dynamic Addr. Translation | 0 |
| 0 | 11-12 | Segment size[1] | Dynamic Addr. Translation | 00 |
| 0 | 20 | Clock Comparator Mask | Clock Comparator | 1 |
| 0 | 21 | CPU Timer Mask | CPU Timer | 0 |
| 0 | 24 | Interval Timer Mask | External Interruption | 1 |
| 0 | 25 | Interrupt Key Mask | External Interruption | 1 |
| 0 | 26 | External Signal Mask | External Interruption | 0 |
| 1 | 0-7 | Segment Table Length | Dynamic Addr. Translation | Set by CP. Value |
| 1 | 8-25 | Segment Table Address | Dynamic Addr. Translation | varies with the type of virtual machine. |
| 2 | 0-31 | Channel Masks | I/O Interruptions | FFFFFFFF |
| 8 | 16-31 | Monitor Masks | Monitoring | Value depends on virtual machine. |
| 9 | 0-7 | PER[2] Event Masks | Program-Event Recording | Value depends on |
| 9 | 16-31 | PER GR Alteration Masks | Program-Event Recording | virtual machine. |
| 10 | 8-31 | PER Starting Address | Program-Event Recording | Value depends on virtual machine. |
| 11 | 8-31 | PER Ending Address | Program-Event Recording | Value depends on virtual machine. |
| 14 | 0 | Check-Stop Control | Machine-Check Handling | Value depends on |
| 14 | 1 | Synchronous MCEL[3] Control | Machine-Check Handling | machine check |
| 14 | 2 | I/O Extended Logout Control | Channel-Check Handling | handler for the |
| 14 | 4 | Recovery Report Mask | Machine-Check Handling | virtual machine. |
| 14 | 5 | Degradation Report Mask | Machine-Check Handling | |
| 14 | 6 | External Damage Report Mask | Machine-Check Handling | |
| 14 | 7 | Warning Mask | Machine-Check Handling | |
| 14 | 8 | Asynchronous MCEL Control | Machine-Check Handling | |
| 14 | 9 | Asynchronous Fixed Log Ctrl. | Machine-Check Handling | |
| 15 | 8-28 | MCEL Address | Machine-Check Handling | |

Explanation:
The fields not listed are unassigned.
The initial value of unassigned register positions is unpredictable.

[1] The initial value varies depending on whether virtual storage is supported in the virtual machine.
[2] PER means program-event recording.
[3] MCEL means machine-check extended logout.

Figure 53. Control Register Assignments




The fields of the PSW are:

| Bits | Contents |
|---|---|
| 0 | Must be zero. |
| 1 | PER (Program Event Recording) enabled. |
| 2–4 | Must be zero. |
| 5 | Address translation. |
| 6 | Summary I/O mask. |
| 7 | Summary extension. |
| 8–11 | The protection key determines if information can be stored or fetched from a particular location. |
| 12 | Extended control mode. |
| 13 | The machine check flag is set to 1 whenever a machine check occurs. |
| 14 | The wait state flag is set to 1 when the CPU is in the wait state. |
| 15 | The problem state flag is set to 1 when the CPU is operating in the problem rather than the supervisor state. |
| 16–17 | Must be zero. |
| 18–19 | The condition code reflects the result of a previous arithmetic, logical, or I/O operation. |
| 20–23 | The program mask indicates whether or not various program exceptions are allowed to cause program interrupts. |
| 24–32 | Must be zero. |
| 33–63 | The instruction address gives the location of the next instruction to be executed for program interrupts or of the instruction last executed for external interrupts. |

Figure 54. The Extended Control PSW (Program Status Word)

# Appendix B: MULTI-LEAVING

MULTI-LEAVING is a term that describes a computer-to-computer communication technique developed for use by the HASP system and used by the RSCS component of VM/370. MULTI-LEAVING can be defined as the fully synchronized, pseudo-simultaneous, bidirectional transmission of a variable number of data streams between two or more computers using binary synchronous communications facilities.

## MULTI-LEAVING IN VM/370

The following sections outline the specifications of a comprehensive, MULTI-LEAVING communications system (as is used in HASP/ASP). While the VM/370 support for programmable BSC remote stations is completely consistent with the MULTI-LEAVING design, it does not use certain of the features provided in MULTI-LEAVING:

- The transmission of record types other than print, punch, input, console, and control is not supported.
- The only general control record type used is the terminal sign-on control.
- Multiple data streams are not supported.
- Only SCB count units of 1 are used.
- No support is included for column binary cards.

## MULTI-LEAVING PHILOSOPHY

The basic element for multileaved transmission is the character string. One or more character strings are formed from the smallest external element of transmission, the physical record. These physical records are input to MULTI-LEAVING and may be any of the classic record types (card images, printed lines, tape records, etc.). For efficiency in transmission, each of these data records is reduced to a series of character strings of two basic types:

1. A variable-length nonidentical series of characters.
2. A variable number of identical characters.

An eight-bit control field, termed a String Control Byte (SCB), precedes each character string to identify the type and length of the string. Thus, a string of nonidentical characters (as in 1 above) is represented by an SCB followed by the nonduplicate characters. A string of consecutive, duplicate, nonblank characters (as in 2 above) can be represented by an SCB and a single character (the SCB indicates the duplication count, and the character following indicates the character to be duplicated). In the case of an all-blank character string, only an SCB is required to indicate both the type and the number of blank characters. A data record to be transmitted is segmented into the optimum number of character strings (to take full advantage of the identical character compression) by the transmitting program. A special SCB is used to indicate the grouping of character strings that compose the original physical record. The receiving program can then reconstruct the original record for processing.

| Control Characters | Usage |
|---|---|
| DLE | BSC Leader (SOH if no transparency feature) |
| STX | BSC Start-of Text |
| BCB | Block Control Byte |
| FCS | Function Control Sequence |
| FCS | Function Control Sequence |
| RCB | Record Control Byte for record 1 |
| SRCB | Sub-Record Control Byte for record 1 |
| SCB | String Control Byte for record 1 |
| DATA | Character String |
| SCB | String Control Byte for record 1 |
| DATA | Character String |
| SCB | Terminating SCB for record 1 |
| RCB | RCB for record 2 |
| SRCB | SRCB for record 2 |
| SCB | SCB for record 2 |
| DATA | Character String |
| SCB | Terminating SCB for record 2 |
| RCB | Transmission Block terminator |
| DLE | BSC Leader (SYN if no transparency feature) |
| ETB | BSC Ending Sequence |

Figure 55. A Typical MULTI-LEAVING Transmission Block

In order to allow multiple physical records of various types to be grouped together in a single transmission block (see Figure 55), an additional eight-bit control field precedes the group of character strings representing the original physical record. This field, the Record Control Byte (RCB), identifies the general type and function of the physical record (input stream, print stream, data set, etc.). A particular RCB type has been designated to allow the passage of control information between the various systems. Also, to provide for simultaneous transmission of similar functions (that is, multiple input streams, etc.), a stream identification code is included in the RCB. A second eight-bit control field, the Sub-Record Control Byte (SRCB), is also included immediately following the RCB. This field is used to supply additional information concerning the record to the receiving program. For example, in the transmission of data to be printed, the SRCB can be used for carriage control information.

For actual MULTI-LEAVING transmission, a variable number of records may be combined into a variable block size, as indicated previously (that is, RCB,SRCB,SCB1,SCB2, ...,SCBn, RCB,SRCB,SCB1,...,etc.). The MULTI-LEAVING design provides for two (or more) computers to exchange transmission blocks, containing multiple data streams as described above, in an interleaved fashion. To allow optimum use of this capability, however, a system must have the capability to control the flow of a particular data stream while continuing normal transmission of all others. This requirement becomes obvious if one considers the case of the simultaneous transmission of two data streams to a system for immediate transcription to physical I/O devices of different speeds (such as two print streams).

To provide for the metering of the flow of individual data streams, a Function Control Sequence (FCS) is added to each transmission block. The FCS is a sequence of bits, some of which represent a particular transmission stream. The receiver of several data streams can temporarily stop the transmission of a particular stream by setting the corresponding FCS bit off in the next transmission to the sender of that stream. The stream can subsequently be resumed by setting the bit on.

However, since only single data streams are supported, RSCS does not support this metering capability. If bit one of the FCS (wait-a-bit) is on, or if bits 4, 9, or 15 (print, console, punch stream identifiers) are off, transmission will be suspended. Thus the bit pattern of x'88C1' represents the minimum acceptable FCS configuration for transmission to be continued.

Finally, for error detection and correction purposes, a Block Control Byte (BCB) is added as the first character of each block transmitted. The BCB, in additional to control information, contains a hexadecimal block sequence count. This count is maintained and verified by both the sending and receiving systems to exercise a positive control over lost or duplicated transmission blocks.

In addition to the normal binary synchronous text control characters (STX, ETB, etc.) MULTI-LEAVING uses two of the BSC control characters, ACK0 and NAK. ACK0 is used as a "filler" by all systems to maintain communications when data is not available for transmission. NAK is used as the only negative response and indicates that the previous transmission was not successfully received.

## MULTI-LEAVING CONTROL SPECIFICATION

This section describes the bit-by-bit definitions of the various MULTI-LEAVING control fields and includes notes concerning their use.

```
 ________
 OIIITTTT
 0      7
```

Usage: To identify each record type within a transmission block

Bits:

| | | |
|---|---|---|
| OIIITTTT | 00000000 | End of transmission block |
| --or-- | | |
| 0 | 1 | Non-EOT RCB |
| III0000 | | III is control information: |
| III | 000 | Reserved |
| | 001 | Request to initiate a function transmission (prototype RCB for function in SRCB ) |
| | 010 | Permission to initiate a function Transmission (RCB for function contained in SRCB ) |
| | 011 | Reserved |
| | 100 | Reserved |
| | 101 | Available for location modification |
| | 111 | General control record (Type indicated in SRCB) |
| --or-- | | |
| 0 | 1 | Non-EOT RCB |
| IIITTTT | | III is used to identify streams of multiple identical functions (such as multiple print streams to a multiple printer terminal). TTTT is the record type identifier. |
| TTTT | 0001 | Operator message display request |
| | 0010 | Operator command |
| | 0011 | Normal input record |
| | 0100 | Print record |
| | 0101 | Punch record |
| | 0110 | Data set record |
| | 0111 | Terminal message routing request |
| | 1000-1100 | Reserved |
| | 1101-1111 | Available to user |

SUB-RECORD CONTROL BYTE (SRCB)

Usage: To provide supplemental information about a record

Bits: The contents of this control block depend upon the record type. Several types are shown below.

```
 ________
 ..CHAR..
 0      7
```

Usage: To identify the type of generalized control record

Bits:

| | | |
|---|---|---|
| CHARACTER | A | Initial terminal sign-on |
| | B | Final terminal sign-off |
| | C | Print initialization record |
| | D | Punch initialization record |
| | E | Input initialization record |
| | F | Data set transmission initialization |
| | G | System configuration status |
| | H | Diagnostic control record |
| | I-R | Reserved |
| | S-Z | Available to user |

SRCB For Print Records

```
 ________
 OMCCCCCC
 0      7
```

Usage: To provide carriage control information for print records

Bits:

| | | |
|---|---|---|
| O | 1 | |
| M | 0 | Normal carriage control |
| | 1 | Reserved |
| CCCCCC | 000000 | Suppress space |
| | 0000NN | Space nn lines after print |
| | 01NNNN | Skip to channel nnnn after print |
| | 1000NN | Space immediate nn spaces |
| | 11NNNN | Skip immediate to channel nnnn |

## SRCB for Punch Records

```
 ________
 OMMBRRSS
 0      7
```

Usage: To provide additional information for punch records

Bits:

| | | |
|---|---|---|
| 0 | 1 | |
| MM | 00 | SCB count units = 1 |
| | 01 | SCB count units = 2 |
| | 10 | SCB count units = 4 |
| | 11 | Reserved |
| B | 0 | EBCDIC card image |
| | 1 | Column binary card image |
| RR | 00 | Reserved |
| SS | NN | Stacker select information |

## SRCB for Input Record

```
 ________
 OMMBRRRR
 0      7
```

Usage: To provide additional information for input records

Bits:

| | | |
|---|---|---|
| 0 | 1 | |
| MM | 00 | SCB count units = 1 |
| | 01 | SCB count units = 2 |
| | 10 | SCB count units = 4 |
| | 11 | Reserved |
| B | 0 | EBCDIC card image |
| | 1 | Column binary image |
| RRRR | 0000 | Reserved |

## SRCB for Terminal Message Routine Record

```
 ________
 OTTTTTTT
 0      7
```

Usage: To indicate the destination of a terminal message

Bits:

| | | |
|---|---|---|
| 0 | 1 | |
| TTTTTTT | 0000000 | Broadcase to all remote systems |
| | NNNNNNN | Remote system number (1-99) or remote system group (100-127) |

STRING CONTROL BYTE (SCB)

```
 ________
 OKLJJJJJ
 0      7
```

Usage: Control field for data character strings

Bits:

| | | |
|---|---|---|
| OKLJJJJJ | 00000000 | End of record |
| --or-- | | |
| OKLJJJJJ | 10000000 | Record is continued in next transmission block |
| --or-- | | |
| O | 1 | Non-EOR SCB |
| K | 0 | Duplicate character string |
| L | 0 | Duplicate character is blank |
| | 1 | Duplicate character is nonblank and follows SCB |
| JJJJJ | NNNNN | Duplicate count |
| --or-- | | |
| O | 1 | Non-EOR SCB |
| K | 1 | Nonduplicate character string |
| LJJJJJ | NNNNN | Character string length |

Note: Count units are normally 1 but may be in any other units. the units used may be indicated at function control sign-on or dynamically in the SRCB.

BLOCK CONTROL BYTE (BCB)

```
 ________
 OXXXCCCC
 0      7
```

Usage: transmission block status and sequence count

Bits:

| | | |
|---|---|---|
| O | 1 | |
| XXX | 000 | Reserved |
| | 001 | Bypass sequence count validation |
| | 010 | Reset expected block sequence count to CCCC |
| | 011 | Reserved |
| | 100 | Reserved |
| | 101 | Available to user |
| | 110 | Available to user |
| | 111 | Reserved |
| CCCC | NNNN | Module 16 block sequence count |

```
 ________________
 OSRRAXXXOTRRXXXB
 0       78       15
```

Usage: To control the flow of function streams

Bits:

| | | |
|---|---|---|
| 0...0 | 1...1 | Reserved (must be 1s) |
| S | 0 | Normal processing |
| | 1 | Suspend all stream transmission (wait-a-bit) |
| RR...RR | 00...00 | Reserved |
| A | 1 | Print stream identifier |
| XXX...XXX | X | Reserved stream identifiers |
| T | 1 | Console stream identifiers |
| B | 1 | Punch stream identifiers |

Note: Function stream identifiers are meaningful only to the receiver of the sequence. If bit A, T, or B is off, transmission is suspended; transmission continues when all three bits are on.

# Appendix C: VM Monitor Tape Format and Content

Each time a monitor call interrupt occurs, VM Monitor receives control and collects data appropriate for the particular class and code of MONITOR CALL. (Or, for USER, PERFORM or DASTAP classes, VM Monitor gets control at periodic intervals to collect data.) The data is formatted into records which are collected sequentially in the order that each interrupt occurred. The tape data format is standard Variable Blocked (VB) format. Data is written at the default tape drive density. Maximum block and record lengths are 4096 bytes. The formats and contents of all the kinds of data records for the currently implemented classes and codes of MONITOR CALL are listed below.

All values described in the following records are binary unless otherwise noted.

[1]Indicates that the field is EBCDIC.

[2]Indicates that the field is in special timer format described below.

[3]See CP_PLM for field format definition.

## HEADER RECORD

Every data record is preceded by the following 12 byte header:

| Data Item | Number of Bytes | DSECT Variable Name |
|---|---|---|
| Total bytes in record | 2 | MNHRECSZ |
| Zeros (standard V format record) | 2 | |
| MONITOR CALL class number | 1 | MNHCLASS |
| MONITOR CALL code number | 2 | MNHCODE |
| Time of Day | 5 | MNHTOD |

Note: Time of day occupies 2 full words in storage, with the right hand 12 bits zeros. The right hand 2 bytes and the leftmost byte are ignored giving 16 microsecond accuracy instead of 1 microsecond.

The first 4 bytes of this header are the standard variable-format record-length field.

## Class Zero - Codes for Tape Header, Trailer, and Data Suspension Records

| Monitor Code | Data Item | Number of Bytes | CP Variable Name | DSECT Variable Name |
|---|---|---|---|---|
| 97 | Tape header record | | | |
| | CPU serial/model number | 8 | CPUID | MN097CPU |
| | Software version number[1] | 8 | DMKCPEID | MN097LEV |
| | Date of data collection session[1] | 8 | tod clock | MN097DAT |
| | Time of data collection session[1] | 8 | tod clock | MN097TIM |
| | USERID of monitor controller[1] | 8 | VMUSER | MN097UID |
| | CR8 mask of enabled classes | 4 | DMKPRGC8 | MN097CR8 |
| 98 | Tape trailer record | | | |
| | USERID of user shutting down monitor[1] | 8 | VMUSER | MN098UID |
| 99 | Tape write suspension record | | | |
| | TOD at suspension[2] | 5 | --- | MN099TOD |
| | Count of write suspensions | 4 | --- | MN099CNT |

## Class Zero - PERFORM

| Monitor Code | Data Item | Number of Bytes | CP Variable Name | DSECT Variable Name |
|---|---|---|---|---|
| 00 | Interval statistics | | | |
| | Total system idle time[3] | 8 | IDLEWAIT | MN000WID |
| | Total system page wait[3] | 8 | PAGEWAIT | MN000WPG |
| | Total system time I/O wait[3] | 8 | IONTWAIT | MN000WIO |
| | Total system problem time[3] | 8 | PROBTIME | MN000PRB |
| | Total paging start I/O's | 4 | DMKPAGPS | MN000PSI |
| | Total page I/O requests | 4 | DMKPAGCC | MN000CPA |
| | Current page frames on free list | 4 | DMKPTRFN | MN000NFL |
| | Pages being written, due for free list | 4 | DMKPTRSW | MN000PSN |
| | Total pages flushed, but reclaimed | 4 | DMKPTRPR | MN000PRC |
| | Number of reserved pages | 4 | DMKPTRRC | MN000RPC |
| | Number of shared system pages | 4 | DMKPTRSC | MN000SPC |
| | Total number of times free list empty | 4 | DMKPTRF0 | MN000FLF |
| | Total number of calls to DMKPTRFR | 4 | DMKPTRFC | MN000CPT |
| | Total pages stolen from in Q users | 4 | DMKPTRSS | MN000SS |
| | Number of pages examined in stealing pages | 4 | DMKPTRFF | MN000PFF |
| | Number of pages swapped from the flush list | 4 | DMKPTRRF | MN000PRF |
| | Number of full scans done in stealing pages | 4 | DMKPTRCS | MN000PCS |
| | Total real external interrupts | 4 | DMKPSANX | MN000NXR |
| | Total calls to DMKPRVLG | 4 | DMKPRVCT | MN000CPR |
| | Total calls to DMKVIOEX | 4 | DMKVIOCT | MN000CVI |
| | Total calls to CCWTRANS from DMKVIO | 4 | DMKVIOCW | MN000CCW |
| | Total Virt Interval Timer Ints reflected | 4 | DMKDSPIT | MN000ITI |
| | Total Virt CPU Timer Ints reflected | 4 | DMKDSPPT | MN000PTI |

| Monitor Code | Data Item | Number of Bytes | CP Variable Name | DSECT Variable Name |
|---|---|---|---|---|
| | Total Virt Clock Comparator Ints reflected | 4 | DMKDSPCK | MN000CKI |
| | Total virtual SVC interrupts simulated | 4 | PSASVCCT | MN000CSV |
| | Total virtual program interrupts handled | 4 | DMKPRGCT | MN000CPG |
| | Total I/O interrupts handled | 4 | DMKIOSCT | MN000CIO |
| | Total calls to dispatch (main) | 4 | DMKDSPCC | MN000CDS |
| | Total fast reflects in dispatch | 4 | DMKDSPAC | MN000CDA |
| | Total dispatches for new PSW's | 4 | DMKDSPBC | MN000CDB |
| | Total calls to schedule | 4 | DMKSCHCT | MN000CSC |
| | Count of virtual machine SSK's simulated | 4 | DMKPRVEK | MN000EK |
| | Count of virtual machine ISK's simulated | 4 | DMKPRVIK | MN000IK |
| | Count of virtual machine SSM's simulated | 4 | DMKPRVMS | MN000MS |
| | Count of virtual machine LPSW's simulated | 4 | DMKPRVLP | MN000LP |
| | Count of virtual machine diagnose inst | 4 | DMKPRVDI | MN000DI |
| | Count of virtual machine SIO's simulated | 4 | DMKVIOSI | MN000SI |
| | Count of virtual machine SIOF's simulated | 4 | DMKVIOSF | MN000SF |
| | Count of virtual machine TIO's simulated | 4 | DMKVIOTI | MN000TI |
| | Count of virtual machine CLRIO's simulated | 4 | DMKVIOCI | MN000CI |
| | Count of virtual machine HIO's simulated | 4 | DMKVIOHI | MN000HI |
| | Count of virtual machine HDV's simulated | 4 | DMKVIOHD | MN000HD |
| | Count of virtual machine TCH's simulated | 4 | DMKVIOTC | MN000TC |
| | Count of virtual machine STNSM's simulated | 4 | DMKPRVMN | MN000MN |
| | Count of virtual machine STOSM's simulated | 4 | DMKPRVMO | MN000MO |
| | Count of virtual machine LRA's simulated | 4 | DMKPRVLR | MN000LR |
| | Count of virtual machine STIDP's simulated | 4 | DMKPRVCP | MN000CP |
| | Count of virtual machine STIDC's simulated | 4 | DMKPRVCH | MN000CH |
| | Count of virtual machine SCK's simulated | 4 | DMKPRVTE | MN000TE |
| | Count of virtual machine SCKC's simulated | 4 | DMKPRVCE | MN000CE |
| | Count of virtual machine STCKC's simulated | 4 | DMKPRVCT | MN000CT |
| | Count of virtual machine SPT's simulated | 4 | DMKPRVPE | MN000PE |
| | Count of virtual machine STPT's simulated | 4 | DMKPRVPT | MN000PT |
| | Count of virtual machine SPKA's simulated | 4 | DMKPRVEP | MN000EP |
| | Count of virtual machine IPK's simulated | 4 | DMKPRVIP | MN000IP |
| | Count of virtual machine PTLB's simulated | 4 | DMKPRVPB | MN000PB |
| | Count of virtual machine RRB's simulated | 4 | DMKPRVRR | MN000RR |
| | Count of virtual machine STCTL's simulated | 4 | DMKPRVTC | MN000TCL |
| | Count of virtual machine LCTL's simulated | 4 | DMKPRVLC | MN000LCL |
| | Count of virtual machine CS's simulated | 4 | DMKPRVCS | MN000CS |
| | Count of virtual machine CDS's simulated | 4 | DMKPRVCD | MN000CD |
| | Count of virt machine diagnose disk I/O's | 4 | DMKHVCDI | MN000HDI |
| | Number of users dialed to virtual machines | 4 | DMKSYSND | MN000NDU |
| | Number of users logged on | 4 | DMKSYSNM | MN000NAU |
| | Number of page reads | 4 | PGREAD | MN000PRD |
| | Number of page writes | 4 | PGWRITE | MN000PWR |
| | Number of system pagable pages | 4 | DMKDSPNP | MN000NPP |
| | Sum of working sets of in-Q users | 4 | DMKSCNPU | MN000SWS |
| | Number of users in interactive queue (Q1) | 4 | DMKSCHN1 | MN000Q1N |
| | No. of users in compute bound queue (Q2) | 4 | DMKSCHN2 | MN000Q2N |
| | Number of users eligible to enter Q1 | 2 | DMKSCHW1 | MN000Q1E |
| | Number of users eligible to enter Q2 | 2 | DMKSCHW2 | MN000Q2E |
| | Monitor sampling interval (seconds) | 2 | DMKPRGTI | MN000INT |
| | Count of cylinders allocated on primary paging device | 2 | ALOCUSED | MN000PPA |
| | Cylinder capacity of primary paging device | 2 | ALOCMAX | MN000PPC |

Class One - RESPONSE

| Monitor Code | Data Item | Number of Bytes | CP Variable Name | DSECT Variable Name |
|---|---|---|---|---|
| 00 | Read command sent to terminal | | | |
| | USERID | 8 | VMUSER | MN10XUID |
| | Line address | 2 | | MN10XADD |
| 01 | Terminal output line | | | |
| | USERID | 8 | VMUSER | MN10XUID |
| | Line address | 2 | | MN10YADD |
| | Byte count | 1 | | MN10YCNT |
| | Line of data | Variable | | MN10YIO |
| 02 | Edited terminal input line | | | |
| | USERID | 8 | VMUSER | MN10XUID |
| | Line address | 2 | | MN10XADD |
| | Byte count | 1 | | MN10YCNT |
| | Line of data[1] | Variable | | MN10YIO |

Note that the line addresses for the 370X in NCP mode appear as the base address.

These records are created at the time that DMKQCN handles the console I/O request. This may reflect a slightly different time than that of the SIO or the I/O interrupt. If DMKQCN is called to write a line that is longer than Terminal line size, more than one MC is issued resulting in more than 1 record. Input and output terminal data collected is limited to 128 bytes. Longer lines are truncated.

Class Two - SCHEDULE

| Monitor Code | Data Item | Number of Bytes | CP Variable Name | DSECT Variable Name |
|---|---|---|---|---|
| 02 | User dropped from dispatch queue | | | |
| | USERID[1] | 8 | VMUSER | MN20XUID |
| | Number of system pageable pages | 4 | DMKDSPNP | MN20XNPP |
| | Sum of working sets of in queue users | 4 | DMKSCHPU | MN20XSWS |
| | Number of users in interactive queue Q1 | 4 | DMKSCHN1 | MN20XQ1N |
| | No. of users in compute bound queue (Q2) | 4 | DMKSCHN2 | MN20XQ2N |
| | Number of users eligible for Q1 | 2 | DMKSCHW1 | MN20XQ1E |
| | Number of users eligible for Q2 | 2 | DMKSCHW2 | MN20XQ2E |
| | User new projected working set size | 2 | VMWSPROJ | MN20XWSS |
| | Queue being dropped from (1 or 2) | 1 | Q1DROP | MN20XQNM |
| | Reserved | 1 | --- | MN2RSV1 |
| | Accumulated user CP simulation time[3] | 8 | VMTTIME | MN20YTTI |
| | Accumulated user virtual time[3] | 8 | VMVTIME | MN20YVTI |
| | Externally assigned dispatch priority | 4 | VMQPRIOR | MN20YPRI |

| Monitor Code | Data Item | Number of Bytes | CP Variable Name | DSECT Variable Name |
|---|---|---|---|---|
| | Pages read while in queue | 2 | VMPGREAD | MN202PGR |
| | Sum of pages resident at all reads | 2 | VMPGRINQ | MN202APR |
| | Number of pages referenced while in Q | 2 | gen reg 4 | MN202REF |
| | Current number of pages resident | 2 | VMPAGES | MN202RES |
| | Number of pages stolen while in queue | 2 | VMSTEALS | MN202PST |
| | User total virt non-spool device SIO count | 4 | VMIOCNT | MN202IOC |
| | User total virtual cards punched | 4 | VMPNCH | MN202PNC |
| | User total virtual lines printed | 4 | VMLINS | MN202LIN |
| | User total virtual cards read | 4 | VMCRDS | MN202CRD |
| 03 | User added to dispatch queue | | | |
| | USERID | 8 | VMUSER | MN20XUID |
| | Number of system pageable pages | 4 | DMKDSPNP | MN20XNPP |
| | Sum of working sets of in queue users | 4 | DMKSCHPU | MN20XSWS |
| | Number of users in interactive queue (Q1) | 4 | DMKSCHN1 | MN20XQ1N |
| | No of users in compute bound queue (Q2) | 4 | DMKSCHN2 | MN20XQ2N |
| | Number of users eligible for Q1 | 2 | DMKSCHW1 | MN20XQ1E |
| | Number of users eligable for Q2 | 2 | DMKSCHW2 | MN20XQ2E |
| | User's projected working set size | 2 | VMWSPROJ | MN20XWSS |
| | Queue being added to | 1 | gen reg 15 | MN20XQNM |
| | Reserved | 1 | --- | MN2RSV1 |
| 04 | User added to eligible list | | | |
| | USERID | 8 | VMUSER | MN20XUID |
| | Number of system pageable pages | 4 | DMKDSPNP | MN20XNPP |
| | Sum of working sets of in queue users | 4 | DMKSCHPU | MN20XSWS |
| | Number of users in interactive queue (Q1) | 4 | DMKSCHN1 | MN20XQ1N |
| | No. of users in compute bound queue (Q2) | 4 | DMKSCHN2 | MN20XQ2N |
| | Number of users eligible for Q1 | 2 | DMKSCHW1 | MN20XQ1E |
| | Number of users eligible for Q2 | 2 | DMKSCHW2 | MN20XQ3E |
| | Users projected working set size | 2 | VMWSPROJ | MN20XWSS |
| | Queue being added to | 1 | VMQ1 | MN20XQNM |
| | Reserved | 1 | --- | MN2RSV1 |
| | Accumulated users CP sumulation time | 8 | VMTTIME | MN20YTTI |
| | Accumulated users virtual time | 8 | VMVTIME | MN20YVTI |
| | Eligible list priority | 2 | VMEPRIOR | MN20YPRI |

Class Four - USER

| Monitor Code | Data Item | Number of Bytes | CP Variable Name | DSECT Variable Name |
|---|---|---|---|---|
| 00 | Interval user resource utilization statistics | | | |
| | USERID[1] | 8 | VMUSER | MN400UID |
| | Accumulated user CP simulation time | 8 | VMTTIME | MN400TTI |
| | Accumulated user virtual time | 8 | VMVTIME | MN400VTI |
| | Total page reads | 4 | VMPGREAD | MN400PGR |
| | Total page writes | 4 | VMPGWRIT | MN400PGW |
| | Total non-spooled I/O requests | 4 | VMIOCNT | MN400IOC |
| | Total cards punched | 4 | VMPNCH | MN400PNC |
| | Total lines printed | 4 | VMLINS | MN400LIN |
| | Total cards read | 4 | VMCRDS | MN400CRD |
| | User running status | 1 | VMRSTAT | MN400RST |
| | User dispatch status | 1 | VMDSTAT | MN400DST |

| Monitor Code | Data Item | Number of Bytes | CP Variable Name | DSECT Variable Name |
|---|---|---|---|---|
| | User operating status | 1 | VMOSTAT | MN400OST |
| | User queuing status | 1 | VMQSTAT | MN400QST |
| | User processing status | 1 | VMPSTAT | MN400PST |
| | User control status | 1 | VMESTAT | MN400EST |
| | User tracing control | 1 | VMTRCTL | MN400TST |
| | User message level | 1 | VMMLEVEL | MN400MLV |
| | User queue level | 1 | VMQLEVEL | MN400QLV |
| | User command level | 1 | VMCLEVEL | MN400CLV |
| | User timer level | 1 | VMTLEVEL | MN400TLV |
| | Interrupt pending summary | 1 | VMPEND | MN400PND |
| | User's externally assigned priority | 1 | VMUPRIOR | MN400UPR |
| | Reserved | 1 | --- | MN4RSV1 |
| | Current number of pages resident | 2 | VMPAGES | MN400RES |
| | Current working set size estimate | 2 | VMWSPROJ | MN400WSS |
| | Page frames allocated on drum | 2 | VMPDRUM | MN400PDR |
| | Page frames allocated on disk | 2 | VMPDISK | MN400PDK |
| | Monitor sampling interval (seconds) | 2 | DMKPRGTI | MN400INT |

Class Five - INSTSIM

| Monitor Code | Data Item | Number of Bytes | CP Variable Name | DSECT Variable Name |
|---|---|---|---|---|
| 00 | Start of PRIVOP simulation | | | |
| | USERID[1] | 8 | VMUSER | MN500UID |
| | The privileged instruction | 4 | VMINST | MN500INS |
| | Virtual storage address of PRIVOP | 4 | VMPSW | MN500VAD |
| | Total user CP simulation time at start of simulation | 8 | cpu timer | MN5000VH |

Class Six - DASTAP

| Monitor Code | Data Item | Number of Bytes | CP Variable Name | DSECT Variable Name |
|---|---|---|---|---|
| 00,01 | Device activity data for all Tape and DASD devices | | | |
| | Number of device blocks recorded | 2 | | MN600NUM |
| | For each device - | | | |
| | Device address | | RDEVADDR+ RCUADDR+ | |
| | | 2 | RCHADDR | MN600ADD |
| | VM/370 type codes | 2 | RDEVTYPC | MN600TY |
| | Volume serial number[1] | 6 | RDEVSER | MN600SER |
| | Device accumulated I/O count | 4 | RDEVIOCT | MN600CNT |

Note: The monitor code zero record is collected when the MONITOR START TAPE command is entered. Thereafter, all DASTAP records are collected with a monitor code of one.

Class Seven - SEEKS

| Monitor Code | Data Item | Number of Bytes | CP Variable Name | DSECT Variable Name |
|---|---|---|---|---|
| 00 | DASD I/O request record | | | |
| | USERID[1] | 8 | VMUSER | MN700UID |
| | Device address | | RDEVADDR+ RCUADDR+ | |
| | | 2 | RCHADDR | MN700ADD |
| | Seek cylinder address | 2 | IOBCYL | MN700CYL |
| | Current arm position | 2 | RDEVCYL | MN700CCY |
| | Number of queued I/O tasks on device | 1 | RDEVQCNT | MN700QDV |
| | Number of queued I/O tasks on control unit | 1 | RCUQCNT | MN700QCU |
| | Number of queued I/O tasks on channel | 1 | RCHQCNT | MN700QCH |
| | Current seek direction | 1 | RDEVFLAG | MN700DIR |

Note: Current seek direction value is

X'00' seeking to lower cyl addr
X'01' seeking to higher cyl addr

Class Eight - SYSPROF additional data for system profile class

| Monitor Code | Data Item | Number of Bytes | CP Variable Name | DSECT Variable Name |
|---|---|---|---|---|
| 02 | Additional data at add Q drop Q times | | | |
| | Number of 4 byte device block counts which follow | 2 | --- | MN802NUM |
| | For each device ...count of I/O's | 4 | RDEVIOCT | --- |
| | After device counts ... | | | |
| | Current number of users logged on | 4 | DMKSYSNM | MN802NAU |
| | Total system page reads | 4 | PGREAD | MN802PGR |
| | Total system page writes | 4 | PGWRITE | MN802PGW |
| | Current number of pageable pages | 4 | DMKDSPNP | MN802NPP |
| | Total system idle time | 8 | IDLEWAIT | MN802WID |
| | Total system page wait time | 8 | PAGEWAIT | MN802WPG |
| | Total system I/O wait time | 8 | IONTWAIT | MN802WIO |
| | Total system problem time | 8 | PROBTIME | MN802PRB |

# Index


$$BCLOSE transient 356
$$BDUMP transient 356
$$BOPEN transient 355
$$BOPENR transient 356
$$BOPNLD transient 356
$$BOPNR2 transient 356
$$BOPNR3 transient 356

A
ABEND (see abnormal termination (ABEND))
ABEND macro 327
abnormal termination (ABEND) 14
(see also problem types)
CMS ABEND
action for 175-177
codes 175-177
module name 175-177
reason for 173
recovery 30,175-177
collect information 126,178
CP ABEND
action for 111-125
codes 111-125
reason for 109
recovery 109
CP dump 106
dump 106,108,173
(see also CMS (Conversational Monitor System), dump and CP (Control Program), dump)
in CMS 16
in CP 14
in DOS 16
in OS 16
internal trace table 126
messages 14,15
of a system routine 29
program check in CP 26
program interrupt 194
reason for 26,28,173
register usage 126
save area conventions 127
SVC 0 26,109
system 29
SYSTEM RESTART button 26
ACCESS command, accessing OS data sets 332
access method, OS, support of 329
accounting
ACCTOFF routine 240
ACCTON routine 239
cards, generating 267
records
created by the user 239
format for dedicated devices 238
format for virtual machines 237
when to punch 239
user options 239
Active Disk Table (ADT) 179,370
Active File Table (AFT) 179
address
translation
example 207
virtual-to-real 207
ADSTOP command
format 46
summary 39
usage 47
ADT (see Active Disk Table (ADT))
AFT (see Active File Table (AFT))
allocating, storage 306
ASSGN command 338
assigning, dedicated channels to a virtual machine 187
assist feature, virtual machine 47
ASSIST operand, of CP SET command (CP) 61
ATTACH macro 327
attaching, virtual devices 187
auxiliary directories
creating 369
example 371
error handling 371
establishing linkage 370
generating 369
initializing 369
saving resources 369

B
BALRSAVE (BAL register save area) 27,127
BASE control record, ZAP program 392
batch, facility (see CMS Batch Facility)
BATEXIT1 367
BATEXIT2 367
BATLIMIT 366
BDAM
restrictions on 331
support of 329
BEGIN command
format 48
summary 39
usage 48
BLDL macro 326
blocks
control
CMS 174
CP 128
BPAM, support of 329
BREAK
subcommand
error messages 144
format 142
usage 143
BSAM/QSAM, support of 329
BSP macro 328
buffers
forms control 279
print 279

C
calculating, dispatching priority 192
CAW
  operand, of DISPLAY command 41
  subcommand
    error messages 145
    format 145
    of DEBUG command 41
    usage 145
CAW (Channel Address Word)
  format 145
  virtual machine, displaying 49
Channel Address Word (see CAW (Channel Address Word))
channel program, modification 265
Channel Status Word (CSW)
  format 146
  virtual machine, displaying 49
CHAP macro 327
CHECK macro 328
CHKPT macro 328
class
  device 137
  privilege 189
clock, comparator 255
CLOSE
  command
    usage 31,106
CLOSE/TCLOSE macros 327
CMNDLINE (command line) 179
CMS (Conversational Monitor System)
  (see also virtual machines)
  ABEND macro 29
  abnormal termination 19,25
    codes 175-177
    collect information 178
    messages 16
    procedure 28,30,173
    reason for 173
    recovery 30
  auxiliary directories 369
  Batch Facility (see CMS Batch Facility)
  called routine table 317
  command language 287
  command processing 316
  commands (see CMS commands)
  control blocks, relationships 174
  devices supported 299
  DEVTAB (Device Table) 298
  display the PSW 30
  DMSABN macro description 29
  DMSFREE 298
    free storage management 302
    macro description 303
    service routines 308
  DMSFRES macro description 308
  DMSFRET macro description 307
  DMSFST macro description 369
  DMSITS 312,318
  DMSNUC 298
  dump
    at abnormal termination 173
    examine low storage 173
    format 173
    message 173
    register usage 180
  examine low storage 30
  file system 288,292
  free storage management 300
    DMSFREE 302
    GETMAIN 300
  function table 321
    reserved names 321
  functional information 297
  Halt Execution (HX) 29
  how to approach a problem 13
  how to save it 364
  interface with display terminals 320
  interrupt handling 293
  introduction 287
  language processors 105
  load map 30,171
  loader tables 300
  low storage 30
  nucleus 299
  nucleus load map 171
  program
    development facilities 289
    exception 28
  register usage 180,297
  restrictions 104
  returning to calling routine 317
  sample load map 172
  saved system restrictions 365
  simulation of DOS/VS functions 335
  storage
    dump 31,173
    map 301
    structure 298
  structure of DMSNUC 297
  SVC handling 312,318
  symbol references 297
  system, ABEND 29
  system save area modification 317
  transient area 299,315
  user
    area 315
    program area 300
  USERSECT (User Area) 298
CMS Batch Facility
  BATEXIT1 367
  BATEXIT2 367
  BATLIMIT MACRO file 366
  data security 368
  EXEC procedures 367
  installation input 366
  /JOB control card 367
  remote input 366
  system limits 366
    resetting 366
  user control cards 367
CMS commands 140
  BREAK subcommand 142
  CAW subcommand 145
  CSW subcommand 146
  DDR 170
  DEBUG 140
  DEFINE subcommand 148
  DUMP subcommand 150
  FILEDEF 108,332
  GO subcommand 152
  GPR subcommand 154
  how to add one 321
  HX subcommand 155
  LISTF 171
  MODMAP 171

MOVEFILE 108
ORIGIN subcommand 156
PRINT 171
PSW subcommand 158
RETURN subcommand 159
SET subcommand 160
STORE subcommand 162
SVCTRACE 140,166
VMFDUMP 107
X (Examine) subcommand 164
CMSCB (OS control blocks) 178
CMS/DOS
command summary 337-338
considerations for execution 360
control blocks used by 357
DOS/VS volumes needed 359
environment, defined 335
generating 357
library volume directory entries 358
performance 360
restrictions 360
storage requirements 358
support
DOS/VS macros under CMS 342-344
for declarative macros 345
for DTFDI macro 348
for DTFMT macro 350
for DTFPR macro 352
for DTFSD macro 353
for EXCP 356
for imperative macros 355
for the DTFCD macro 346
for the DTFCN macro 347
for transient routines 355
hardware devices 335
of DOS/VS functions 336
of DOS/VS supervisor and I/O macros 332
of physical IOCS macros 332
user responsibilities 357
CMSDOS discontiguous saved segment 245
CMSSEG
discontiguous saved segment 364
usage options 364
coding conventions
addressing 275
constants 274
CP 274
error messages 276
loadlist requirements 276
module names 275
register usage 274
cold start 197
column binary 103
command
language
CMS 287
RSCS 412
processing, RSCS 419
summary, RSCS 413
commands (see CMS commands, CP commands and RSCS commands)
comment control record, ZAP program 394
communication, between VM/370 and OS/VS1 251
COMND macro 278
compiler input/output assignments 339
completion code X'00B' 194
considerations
CMS/DOS 360
for virtual=real performance option 215
paging 210
VM Monitor 232
VSAM data set compatibility 363
console, function (see CP (Control Program))
control
blocks, used by CMS/DOS routines 357
records, for ZAP command 390
registers, displayed by DISPLAY command 40
Control Program (see CP (Control Program))
control program, for 3704/3705 Communications Controller 379
Conversational Monitor System (see CMS (Conversational Monitor System))
COPY, control statement, DDR program 91
CP (Control Program)
ABEND code, table 111-125
abnormal termination 25
messages 14,15
procedure 26,27,109
with automatic restart 19
without automatic restart 19
coding conventions 274
commands (see CP commands)
concurrent execution of virtual machines 183
console functions, how to add one 278
control block relationships 129
debugging CP on a virtual machine 96
disabled loop 22
procedure 33
disabled wait 20
procedure 24,35
dump
at abnormal termination 108
examine ABEND code 109
examine low storage 109
format 108
on disk 107
on printer 107
on tape 108
printing disk dump 107
printing tape dump 108
VMFDUMP command usage 107
enabled loop 22
enabled wait 20
procedure 24,36
enabled wait state 103
errors encountered by the warmstart program 15
examine low storage 27
how to approach a problem 13
identifying a pageable module 139
initialization 197
internal trace table 27,84,96,126 (see also CP trace table)
I/O management on virtual machine 187
load map 27
looping condition 24
low storage 27
machine check 27
page zero handling 186
privileged instruction simulation 183
problem state execution 183

program check 26
in the checkpoint program 14
in the dump program 14
PSA, Prefix Storage Area 27
real control blocks 27
I/O 198
register usage 126
restrictions 32,99
RMS (Recovery Management Support) 194
save areas 127
spooling 187
storage dump 26,108
SVC interrupt handling 200
SVC 0 26
SYSTEM RESTART button 26,36
trace table entries 98
(see also CP trace table)
unexpected results 20,25
procedure 32
virtual control blocks 27
I/O 199
virtual machine interrupt handling 183
wait state status messages 14
CP commands 45,189
ADSTOP 46
BEGIN 48
DCP 76
DISPLAY 49
DMCP 79
DUMP 55
for system programmers and system analysts 75
format 45
how to add a command 278
INDICATE 219
E privilege class 221
G privilege class 220
LOCATE 82
MONITOR 84,219,227
operands 45
privilege classes 45
QUERY 85
SAVENCP 87
SAVESYS 87
SET 58
STCP 88
STORE 65
SYSTEM 69
TRACE 70
CP trace table 27,84
allocation 96
entries 98
restarting tracing 97
size 96
terminating tracing 97
usage 97,126
CPABEND (ABEND Code) 109
CPEREP program 28
CPSTAT (CP running status) 126
CPU (Central Processing Unit)
resources 192
timer 254
utilization 192
creating, an NCPDUMP file 406
CSW (see Channel Status Word (CSW))
CSW
operand, of the DISPLAY command 41
subcommand
error messages 147
format 146
of DEBUG command 41
usage 146
CVTSECT (CMS Communicatcns Vector Table) 179

D
DASD DDR program (see DASD Dump Restore (DDR) program)
DASD Dump Restore (DDR) program 33,89
changing GMT to local time 93
COPY statement 91
DUMP statement 91
function statements 91
INPUT statement 90
invoking, as a standalone program 170
I/O definition statements 89
OUTPUT statement 90
PRINT statement 95
RESTORE statement 91
restrictions 92
sample output 94
standalone version 89
SYSPRINT statement 91
TYPE statement 95
usage 170
DASD I/O function 262
data
records, VM Monitor 438
security, batch 368
data set control block (DSCB) 329
data sets
OS
accessing 332
defining 332
reading 331
reading restrictions 333
VSAM, compatibility considerations 363
DCB macro 328
DCP command
format 76
responses 76
usage 76,78
DDR (see DASD Dump Restore (DDR) program)
DDR command
COPY function statement format 91
DUMP function statement format 91
INPUT control statement format 90
OUTPUT control statement format 90
PRINT function statement format 95
RESTORE function statement format 91
SYSPRINT control statement format 91
TYPE function statement format 95
usage 33
DEBUG command
BREAK subcommand 142
summary 39
CAW subcommand 145
summary 41
CSW subcommand 146
summary 41
DEFINE subcommand 148
DUMP subcommand 150
summary 39
usage 34

GO subcommand 152
summary 39
GPR subcommand 154
summary 40
HX subcommand 155
messages 141
ORIGIN subcommand 156
PSW subcommand 158
summary 40
usage 30
RETURN subcommand 159
rules for using 141
SET CAW subcommand, summary 42
SET CSW subcommand, summary 42
SET GPR subcommand, summary 41
SET PSW subcommand, summary 42
SET subcommand 160
STORE subcommand 162
summary 41
subcommands 142
usage 29
X (Examine) subcommand 164
summary 40
debugging
analyzing the problem 22
applying a PTF 13,22
comparison of CP and CMS facilities 44
how to start 13,23
identifying
a looping condition 23
a looping condition in the virtual machine 16
a wait 23
a wait state in the virtual machine 17
an abnormal termination 23
the problem 16
unexpected results 23
introduction 13
on a virtual machine 31
procedure
for abnormal termination 25
for CMS abnormal termination 28
for CP ABEND without dump 27
for CP abnormal termination 26
for CP disabled loop 33
for CP disabled wait 35
for CP enabled wait 36
for CP unexpected results 32
for looping condition 24
for RSCS disabled wait 37
for unexpected results 25
for virtual machine abnormal termination 30
for virtual machine disabled loop 34
for virtual machine disabled wait 36
for virtual machine enabled loop 34
for virtual machine enabled wait 37
for virtual machine unexpected results 32
for wait 24
recognizing a problem 14
summary of VM/370 debugging tools 39
unproductive processing time 17
with VM/370 facilities 26
declarative macros 345
dedicated
channel, assigning to a virtual machine 187
device 104
DEFINE subcommand
error messages 149
format 148
usage 148
DELAYED operand, of CP SET command 61
DELETE macro 326
demand paging 185
DEQ macro 327
DETACH, macro 328
detaching, virtual devices 187
determining, virtual machine storage size 271
DEVICE, command 31
devices
class codes 137
CMS supported 299
feature codes 139
model codes 139
sense information 194
supported, for VSAM under CMS 361
type codes 137
DEVTAB (Device Table) 298
DEVTYPE macro 327
DIAGNOSE instruction 257
channel program modification 265
clear I/O recording 263
DASD I/O function 262
define the function of the PA2 function key 269
determine virtual machine storage size 271
device type function 264
display data on the 3270 console screen 269
error message editing 270
examine real storage 258
find the address of a discontiguous saved segment 248
FINDSYS function 248,273
general I/O function 263
generate accounting cards 267
input spool file manipulation 260
load a discontiguous saved segment 248
LOADSYS function 248,271
page release function 259
pseudo timer 259
purge a discontiguous saved segment 248
PURGESYS function 248,273
read LOGREC DATA 266
read system dump spool file 266
read system symbol table 267
restrictions 102
save 3704/3705 control program 269
start of LOGREC area 266
to determine virtual machine size 248
update user directory 267
virtual console function 258
3270 virtual console interface 269
directory, entries for CMS/DOS library volumes 358
DISABLE, operand, of the NETWORK command 401
discontiguous saved segments 244
loading 271

purging 273
discontiguous shared segments
defined via the NAMESYS macro 245
user requirements 245
dispatching
interactive users 192
non interactive users 192
priority, calculating 192
scheme, for virtual machines 192
virtual machines
from queue 1 192
from queue 2 192
DISPLAY
command
format 49
responses 52
summary 40
usage 30,34,51
operand, of the NETWORK command 402
display terminals, CMS interface 320
displaying
data on the 3270 console screen 269
floating-point registers, DISPLAY command 40
general registers
DISPLAY command 40
GPR subcommand of DEBUG command 40
PSW
DISPLAY command 40
PSW subcommand of DEBUG command 40
storage
DISPLAY command 40
X subcommand of DEBUG command 40
DISPSW macro display terminals, DISPSW macro 320
DMCP command
format 79
responses 80
usage 80,81
DMKCFM (console function) support 278
DMKCKP 197
DMKCPI 197
DMKDDR (see DASD Dump Restore (DDR) program)
DMKSAV 197
DMKSNT (system name table) 241
DMSABN (ABEND routine) 178
DMSABN macro 29
operands 29
DMSEXS 311
DMSFREE 298
allocating nucleus free storage 306
allocating user free storage 306
error codes 310
operands 303
service routines 308
storage management 302
DMSFRES 308
error codes 310
operands 308
DMSFRET 307
error codes 310
operands 307
releasing storage 307
DMSINA 314
DMSINT 314
DMSIOW 295
DMSITE 296
DMSITI 294
DMSITP 295
DMSITS 293,312,318
DMSKEY 311
DMSLADAD, entry for auxiliary directory 370
DMSNUC 297,298
DOS (Disk Operating System)
abnormal termination
messages 16
procedure 31
DOS/VS
functions simulated by CMS 335
macros
supervisor 340
supported under CMS 342-344
DSCB 329
DTFCD macro 345
DTFCN macro 347
DTFDI macro 348
DTFMT macro 348
DTFPR macro 351
DTFSD macro 353
DUMP
command
define print limits 55
format 55
responses 57
summary 39
usage 34,37,56
control record, ZAP program 390
control statement, DDR program 91
dump
(see also CP (Control Program), dump and CMS (Conversational Monitor System), dump)
from 3704/3705
erasing 406
formatting 406
printing 406
DUMP
operand
of NCPDUMP command 406
of the NETWORK command 400
subcommand
error messages 151
format 150
usage 150
dump, used in problem determination 26
dumping
storage
at the printer 44
at the terminal 44
to a real printer 107
dynamic load overlay 375
dynamically modified program restrictions 99

E
EC (Extended Control) mode 34
EC (Extended Control) PSW 427
ECMODE option 255
ECRLOG (control registers) 178
editing, error messages 270
efficiency, of VM/370 performance options 208

Emulation Program (EP)
(see also 3704/3705 control program)
special considerations for loading 385
support under VM/370 380
3704/3705 control program 379
emulators
DOS 103
integrated 103
ENABLE, operand, of the NETWORK command 401
END, control record, ZAP program 395
enhancements, VM/VS Handshaking 253
ENQ macro 327
ENTRY, option, of SAVENCP command 383
environment, of VM/370, system load 234
EP (see Emulation Program (EP))
ERASE, option, of NCPDUMP command 406
erasing, 3704/3705 dump files 406
error codes 310
DMSFREE 310
DMSFRES 310
DMSFRET 310
error messages, editing 270
EXCP, CMS/DOS support for 356
Extended Control mode (see EC (Extended Control) mode)
extended control registers
virtual machine
displaying 49
printing 55
external interrupt
BLIP character 296
external console interrupt 195,201
HNDEXT macro 296
in CMS 296
in RSCS 420
interval timer 195,201
timer 296
TOD clock comparator 201
EXTOPSW (external old PSW) 173
EXTRACT macro 327
EXTSECT (external interrupt work area) 179

F
favored execution option 212
FCB (File Control Block) 297
FCBTAB (file control block table) 178
features, device 139
fetch storage protection 185
file
management
CMS 288,292
RSCS 418
File Status Table 369
FILEDEF command 332
AUXPROC option 333
defining OS data sets 332
usage 108
files, OS format, support of 329
FIND, macro 326
finding
saved systems 273
the address of a discontiguous saved segment 248
floating-point registers
virtual machine
displaying 49
printing 55
formatting, 3704/3705 dumps 406
forms control buffer
FCB 279
examples 285
macro 285
index feature 285
example 286
FPRLOG (floating-point registers) 178
free storage
management
CMS 300
RSCS 415
FREEDBUF macro 327
FREEMAIN macro 325
FREEPOOL macro 326
FREESAVE (DMSFRE register save area) 27,127

G
GENDIRT command
creating an auxiliary directory 370
format 370
general registers
virtual machine
displaying 49
printing 55
generating, CMS/DOS 357
GET macro 330
GETMAIN
free element chain 302
GETMAIN/FREEMAIN macros 326
macro 325
simulation 302
GETPOOL macro 326
GO subcommand
error messages 152
format 152
usage 152
GPR subcommand
error messages 154
format 154
usage 154
GPRLOG (general registers) 178

H
HALT, operand, of NETWORK command 398
handling
OS files
on CMS disks 322
on OS or DOS disks 323
header, record, VM Monitor 437
HX subcommand
error messages 155
format 155
usage 155

I
IDENTIFY macro 327
IIP (ISAM Interface Program) 363
imperative macros 355
INDICATE command 219
described 219

format
E privilege class 221
G privilege class 220
indicators, of system load 219
initialization 197
INPUT, control statement, DDR program 90
input/output (see I/O)
interrogating input/output assignments 340
interrupt, I/O, in CMS 294
interrupt handling 194
CMS 293
input/output interrupts 294
SVC interrupts 293
terminal interrupts 295
DMSITS 293
external interrupts 195,296
I/O interrupts 187
machine check interrupts 194,296
program interrupts 194,295
reader/punch/printer interrupts 295
RSCS 420
SVC interrupts 194
user controlled device interrupts 295
interval timer 254
INTSVC 312
I/O
assignments
compiler 339
interrogating 340
control blocks
real 198
relationship 198,199
virtual 199
function
DASD 262
general 263
interrupt
in CMS 294
in RSCS 421
logging, RSCS 422
management 187
overhead in CP, reducing 209
recording, clear 263
IOBLOK 27
IOSECT (I/O interrupt work area) 179
IPL (initial program load), NO CLEAR restriction 103

L
LASTCMND, last command 178
LASTCMND (last command) 31
LASTEXEC, last EXEC procedure 178
LASTEXEC (last EXEC procedure) 31
LASTLMOD
last module in free storage 178
last module loaded 30
LASTTMOD
last module in transient area 178
last transient loaded 30
LIBE option, of SAVENCP command 383
library volumes, CMS/DOS, directory entries for 358
LINK, macro 325
LIOCS routines supported by CMS/DOS 355
load
environments of VM/370 234
indicators 219
LOAD
macro 325
operand
of NETWORK command 384
of the NETWORK command 400
load library
applying PTFs to 388
updating 388
load map
CMS 171
how to print CMS load map 171
load operations, for a 3704/3705 control program 397
loader tables, (CMS) 300
loading
and saving discontiguous saved segments 247
discontiguous saved segments 248,271
3704/3705 control program
considerations 385
EP considerations 385
NCP considerations 386
PEP considerations 386
loadlist
requirements
CP 276
SPB card 276
LOCATE
command
format 82
responses 82
usage 82
locked pages option 210
LOG record
NPT 423
SML 422
logical units
assignment of 338
programmer 339
system 338
LOGREC area
getting starting address 266
reading 266
loop 33
(see also problem types)
disabled
CP 33
virtual machine 34
enabled, virtual machine 34
LOWSAVE (DEBUG save area) 178
LUB (Logical Unit Block) table 338

M
machine check
CP 27
during start-up 194
interrupt 194
in CMS 296
not diagnosed 27
unrecoverable 27
macros
declarative 345
imperative 355
OS (see OS (Operating System), macros)
VSAM, supported under CMS 262
MCKOPSW (CMS machine check old PSW) 173

minidisks 187
  maximum size for CMS 104
  restrictions 99
MNEMONIC option, of NCPDUMP command 406
model
  dependencies, restrictions 102
  device 139
MONITOR command 219
  CP internal trace table 84
  described 226
  format 84,227
  implemented classes 230
  responses 84
  summary 43
  usage 84
monitoring, recommendations 234
MOVE command, usage 108
MULTI-LEAVING
  block control byte (BCB) 435
  character string 429
  control fields
    record control byte (RCB) 432
    string control byte (SCB) 435
    sub-record control byte (SRCB) 433
  description of 429
  function control sequence (FCS) 436
  in VM/370 429
  transmission block 430
multiple path support restrictions 102

N
NAME
  control record, ZAP program 392
  option, of SAVENCP command 383
named systems
  allocating DASD space 241
  generating 241
    SPB card 241
    using the NAMESYS macro 241
  saved system 241
  SAVESYS command 243
  shared segments 243
  system name table (DMKSNT) 241
NAMESYS macro, for saved systems 241
NCP (see Network Control Program (NCP))
NCPDUMP
  command
    described 406
    DUMP operand 406
    usage 406
  file, creating 406
NETWORK command
  described 384,397
  DISABLE operand 401
  DISPLAY operand 402
  DUMP operand 400
  ENABLE operand 401
  execution described 384
  format
    class A 398
    class A and B 400
    class F 405
  HALT operand 398
  LOAD operand 384,400
  QUERY operand 402
  SHUTDOWN operand 398
  TRACE operand 405
  usage, for remote 3270 396
  VARY operand 403
Network Control Program (NCP)
  (see also 3704/3705 control program)
  special considerations for loading 386
  support under VM/370 380
  3704/3705 control program 379
NOFORM option, of NCPDUMP command 406
NOSVC operand, of CP SET command 61
NOTE macro 328
NPT LOG record 423
nucleus (CMS) 299
NUCON (nucleus constant area) 178

O
OPEN/OPENJ macros 326
operator, console, count control 103
options
  performance
    favored execution 212
    locked pages 210
    priority 214
    reserved page frames 211,214
    virtual=real 186,212,215
    virtual machine 212
ORIGIN
  subcommand
    error messages 156
    format 156
    usage 156
OS (Operating System)
  abnormal termination
    messages 16
    procedure 31
  data management simulation 322
  data sets, reading 331
  formatted files 329
  handling
    files on CMS disks 322
    files on OS or DOS disks 323
  macros
    ABEND 327
    ATTACH 327
    BLDL 326
    BSP 328
    CHAP 327
    CHECK 328
    CHKPT 328
    CLOSE/TCLOSE 327
    DCB 328
    DELETE 326
    DEQ 327
    descriptions of 324
    DETACH 328
    DEVTYPE 327
    ENQ 327
    EXTRACT 327
    FIND 326
    FREEDBUF 327
    FREEMAIN 325
    FREEPOOL 326
    GET 330
    GETMAIN 325
    GETMAIN/FREEMAIN 326
    GETPOOL 326

IDENTIFY 327
LINK 325
LOAD 325
NOTE 328
OPEN/OPENJ 326
POINT 328
POST 325
PUT 330
PUTX 330
RDJFCB 328
READ 330
SNAP 327
SPIE 326
STAE 328
STAX 328
STIMER 327
STOW 326
SYNADAF 328
SYNADRLS 328
TCLEARQ 328
TGET/TPUT 328
TIME 327
TTIMER 327
under CMS 322
WAIT 325
WRITE 330
WTO/WTOR 327
XCTL 325
XDAP 325
OS/VS1, running under control of VM/370 251
OUTPUT, control statement, DDR program 90
overhead, CP, reducing for I/O 209
overlay structures under CMS 373
overlaying
dynamic load 375
example 374
prestructured 373

P
page
allocation, RSCS 417
exceptions, effects of 210
frames 184
reserved 186,211
locking 210
release 259
selection 203
SPB (Set Page Boundary) card 276
table 184
zero
restrictions 102,186
pageable module, identifying 139
paging 184
address translation 203
by demand 185
considerations 210
lock page 203
page selection 203
paper tape 103
Partitioned Emulation Program (PEP)
(see also 3704/3705 control program)
special considerations for loading 386
support under VM/370 381
3704/3705 control program 379
PA2 program function key, defining the function of 269
PEP (see Partitioned Emulation Program (PEP))
performance 208
avoiding IPL 241
CMS/DOS 360
for mixed mode foreground/background systems 236
for time-shared multi-batch virtual machines 235
measurement 219
options
favored execution 212
locked pages 210
priority 214
reserved page frames 211,214
virtual=real 186,212,215
virtual machine 212
PFnn operand, of CP SET command 61
PGMOPSW (program old PSW) 173
PGMSECT (program check interrupt work area) 179
PLIST (parameter list) 297
POINT macro 328
POST macro 325
preferred virtual machine 212
Prefix Storage Area (see PSA (Prefix Storage Area))
prestructured overlays 373
PREVCMND (previous command) 31,178
PREVEXEC (previous EXEC procedure) 31,178
print buffers
adding new images 280
LOADBUF command 280
print chain image 280
UCB macro 282
UCBCCW macro 284
UCS
examples 281
macro 280
1403 279
UCSB
associative fields 283
examples 284
3211 279
UCSCCW macro 281
PRINT control statement, DDR program 95
printer, interruptions 295
printing, 3704/3705 dumps 406
priority
of execution 184
performance option 214
privilege classes 189
privileged instructions 208
problem
programs, unexpected results 25
types
abnormal termination 18
loop 22
unexpected results 20
wait 20
program
check
in the checkpoint program 14
in the dump program 14
interruption 202
problem state 194
supervisor state 194
states 191

program function keys 61
  delayed execution of 61
  immediate execution of 61
program interrupt, in CMS 295
Program Status Word (see PSW (Program Status Word))
programmer logical units 339
PROPSW (program old PSW) 109
protection keys 185
protection of shared segments 248
PSA (Prefix Storage Area) 27
  ARIOCH (address of first RCHBLOK) 133
  ARIOCU (address of first RCUBLOK) 134
  ARIODV (address of first RDEVBLOK) 134
pseudo timer 255,259
PSW (Program Status Word) 126
  interruption code 30
  keys, CMS 310
  virtual machine, displaying 49
PSW subcommand
  error messages 158
  format 158
  usage 158
PTFs (program temporary fixes)
  applying 13,22
    to 3704/3705 load library 388
PUB (Physical Unit Block) table 338
punch, interruptions 295
punch-feed-read 103
purging
  a discontiguous saved segment 248
  a discontiguous saved segments 273
PUT macro 330
PUTX macros 330

Q
QUERY
  command
    format 85
    operands 85
    responses 85
    usage 86
  operand, of the NETWORK command 402
queue 1, dispatching virtual machines from 192
queue 2, dispatching virtual machines from 192
Q1 (see queue 1)
Q2 (see queue 2)

R
RCHBLOK 133
  RCHADD (address) 133
  RCHFIOB (first IOBLCK pointer) 133
  RCHSTAT (status) 133
  RCHTYPE (type) 133
RCUBLOK 134
  RCUADD (address) 134
  RCUFIOB (first IOBLOK pointer) 134
  RCULIOB (last IOBLOK pointer) 134
  RCUSTAT (status) 134
  RCUTYPE (type) 134
RDEVBLOK 134
  RDEVADD (address) 134
  RDEVAIOB (IOBLOK pointer) 135
  RDEVATT (attached virtual address) 135
  RDEVCKPT (address of enable CKPBLOK) 135
  RDEVEPDV (address of EP free list) 135
  RDEVFLAG (device dependent flags) 135
  RDEVIOER (address of IOERBLOK) 135
  RDEVMAX (highest valid NCP name) 135
  RDEVNCP (reference name of active 3705 NCP) 135
  RDEVNICL (address of network control list) 135
  RDEVSPL (RSPLCTL pointer) 135
  RDEVSTAT (status) 134
  RDEVTFLG (flags) 136
  RDEVTMCD (terminal flags) 136
  RDEVTYPC (class) 135
  RDEVUSER (dedicated user) 135
RDEVICE macro, CPNAME operand 384
RDJFCB macro, 328
READ macro 330
reader, interruptions 295
reading, OS data sets 331
real
  address 207
  printer dumping to 107
  spooling 205
real storage
  examine 258
  optimizing use of 184
REALTIMER option 254
records
  accounting
    created by the user 239
    format for dedicated devices 238
    format for virtual machines 237
reduction
  of CP overhead, for virtual machine I/O 209
  of paging activity 210
  of SIO operation 209
reenterable code, usage 210
registers, usage, CMS 297
releasing
  allocated storage 308
  storage 307
Remote Spooling Communications Subsystem (see RSCS (Remote Spooling Communications Subsystem))
REP, control record, ZAP program 394
RESERVE, operand 186
reserved page frames 186
  performance option 211,214
resources, CPU 192
responses
  DCP command 76
  DISPLAY command 52
  DMCP command 80
  DUMP command 57
  LOCATE command 82
  MONITOR command 84
  QUERY command 85
  SAVESYS command 87
  STCP command 88
  STORE command 67
  SYSTEM command 69
  TRACE command 72

VM Monitor, to unusual tape conditions 232
responsibilities, user, for CMS/DOS 357
RESTORE, control statement, DDR program 91
restrictions
BDAM 331
CMS 104
minidisk 104
saved system 365
CMS/DOS 360
column binary 103
count control 103
CP 99
dedicated device 104
DIAGNOSE instruction 102
DOS
emulator 103
object programs 105
dynamically modified program 99
for reading OS data sets 333
integrated emulators 103
IPL command 103
with NOCLEAR option 103
language processors under CMS 105
minidisk 99
model dependencies 102
multiple path support 96
on use of DASD channel programs for certain devices 100
on use of IBCDASDI 100
on use of search arguments in performing DASD I/O 100
page zero 102
paper tape 103
punch-feed-read 103
SET CLOCK command 103
stacker selection 103
STORE CLOCK command 103
timing dependency 101
resume
execution
BEGIN command 39
GO subcommand of DEBUG command 39
RETURN
subcommand
error messages 159
format 159
usage 159
RSCS (Remote Spooling Communications Subsystem)
command language 412
command processing 419
command summary 413
disabled wait 21
procedure 37
X'001' 37
X'007' 38
X'011' 38
DMTMAP 415
DMTVEC 415
enabled wait 21,38
external interrupts 420
file management 418
free storage 415
functional information 417
interrupt handling 420
I/O
interrupts 421
logging activity 422
logging output 422
logging record 422
line allocation task 416
line driver storage 416
links 411
definition 411
table 411
locations 411
message handling 420
nonprogrammable remote terminals 411
page allocation 417
programmable remote stations 411
queue element management 417
remote stations 411
spool file
access 418
access task 416
storage
allocation 414
structure 414
supervisor 415
service routines 415
supervisor queue 415
extension 415
SVC interrupts 420
system control task 416
tag slot queues 418
task to task communications 419
virtual storage management 417
VM/370 spool system interface 412
RUNUSER (current user) 126

S
save area
BALRSAVE 27,127
CMS system 318
CMS system save area format 318
FREESAVE 27,127
SAVEAREA 27,127
user save area format 318
SAVEAREA (active save area) 27,127
saved systems
CMS 364
described 241
SAVESYS command 243
when to save a system 243
SAVENCP command 87
described 382
ENTRY option 383
execution described 383
LIBE option 383
NAME option 383
SAVESYS command 243
format 87
responses 87
usage 87
segment, shared (see shared segments)
segment table 184
service programs, ZAP 388
SET CLOCK command 103
SET command (CP)
ASSIST operand 61
format 58,107
NOSVC operand 61
operands 58

SVC operand 61
usage 63,107
SET subcommand
error messages 161
format 160
usage 160
SETKEY command, described 247
setting
address stops 44
program function keys 61
tabs on your terminal 62
shared segments 243
described 243
discontiguous 244
protection 248
special considerations 244
virtual machine operation 249
SHUTDOWN operand, of the NETWORK command 398
simulation 208
of DOS/VS functions by CMS 335
single instruction mode 189
SIO (see Start I/O (SIO) instruction)
SML LOG record 422
SNAP macro 327
spanned records, usage 330
SPB (Set Page Boundary) card 276
SPIE macro 326
spool file
access, RSCS 418
manipulation 260
recovery
after checkpoint start 189
after force start 189
after warm start 189
spooling
described 188
real 205
terminal input 189
terminal output 189
via RSCS 188
virtual 204
stacker selection 103
STAE macro 328
start
VM/370
cold 197
warm 197
Start I/O (SIO) instruction
handling 209
reducing 209
STAX macro 328
STCP command
format 88
responses 88
usage 88
STIMER macro 327
stop execution
ADSTOP command 39
BREAK subcommand of DEBUG command 39
stopping tracing
SVCTRACE command 43
TRACE command 43
storage
allocation 306
RSCS 414
CMS 301
dump
CMS 31
CP 26
keys, virtual machine, printing 55
locations
real machine, displaying 76
real machine, printing 79
virtual machine, displaying 49
virtual machine, printing 55
protection
fetch 185
storing, 185
releasing 307
requirements
assembler 373
for CMS support of VSAM 363
for CMS/DOS 358
STORE
subcommand
error messages 162
format 162
usage 162
STORE command
CLOCK operand 103
format 65
operands 65
responses 67
summary 41,42
usage 67
storing
data
into CAW, SET CAW subcommand of DEBUG command 42
into control registers, STORE command 42
into CSW, SET CSW subcommand of DEBUG command 42
into floating-point registers, STORE command 41
into general registers, SET GPR subcommand of DEBUG command 41
into general registers, STORE command 41
into PSW, SET PSW subcommand of DEBUG command 42
into PSW, STORE command 42
STORE command 41
STORE subcommand of DEBUG command 41
information 44
storage protection 185
STOW macro 326
STRINIT macro 300
structure, of RSCS storage 414
SVC
handling
by user 313
commands entered from the terminal 314
invalid SVCs 314
linkage 312
OS and DOS/VS SVC simulation 313
type of SVC 312
interrupt
CMS internal linkage SVCs 293
handling 200
other CMS SVCs 293
problem state 194,200
supervisor state 194,200

interrupts, RSCS 420
operand, of CP SET command 61
SVC 202 312
search hierarchy 314
SVC 203 313
SVCOPSW (SVC old PSW) 173
SVCSECT (SVC interrupt work area) 179
SVCTRACE command 140
format 166
FPRS output line 168
FPRSS output line 168
GPRS AFTER output line 167
GPRSB output line 167
GPRSS output line 168
interpreting the output 166
N/D output line 167
PARM output line 168
summary 42
of output 169
usage 37,166
SYNADAF macro 328
SYNADRLS macro 328
SYSPRINT, control statement, DDR program 91
system, ABEND 29
SYSTEM
command
format 69
responses 69
usage 69
system
dump spool file, reading 266
logical units 338
performance
for mixed mode foreground/background systems 236
measurement 219
routine, abnormal termination of 29
symbol table, reading 267
system name table (DMKSNT) 241
System/370
control registers
allocation 425
assignments 426
extended control (EC) PSW 427
information 425

T
TAB, operand, of CP SET command 62
tabs, setting for your terminal 62
TCLEARQ macro 328
terminal interruptions, in CMS 295
terminals, setting tabs on 62
TGET/TPUT macros 328
time, management 184
TIME macro 327
time slice 192
time-of-day (TOD) clock 255
timers
clock comparator 255
CPU timer 254
interval timer 254
pseudo timer 255
Time of Day (TOD) clock 255
timing restrictions 101
TRACCURR (current trace table entry) 126
TRACE
command
format 70
operands 70
responses 72
summary 42
usage 31,32,34,37,71
operand, of NETWORK command 405
TRACEND (end of trace table) 126
tracing
all user I/O operations, TRACE command 42
branches
TRACE command 42,43
CCWs, TRACE command 43
CP trace table 96
external interrupts, TRACE command 42
information 44
instructions
TRACE command 42,43
interrupts 96
TRACE command 42
I/O 96
interrupts, TRACE command 42
line activity, for a 3704/3705 control program 397
NCP BTU 96
privileged instructions, TRACE command 42
program interrupts, TRACE command 42
queue drop 96
real machine events, MONITOR command 43
run user requests 96
scheduling 96
storage management 96
SVC interrupts
SVCTRACE command 42
TRACE command 42
user operations, TRACE command 43
virtual 206
TRACSTRT (start of trace table) 126
transient area (CMS) 299
transient routines supported by CMS/DOS 355
TTIMER macro 327
TYPE, control statement, DDR program 95
type (device) 137

U
unexpected output 16
unexpected results
(see also problem types)
reason for 32
unit record, devices, sharing 188
unproductive processing time 16
user directory
reading 267
updating 267
user-controlled device interrupts 295
USERSECT (User Area) 298

V
VARY, operand, of the NETWORK command 403
VCHBLOK 131

VCHADD (virtual channel address) 131
VCHSTAT (status) 131
VCHTYPE (type) 131
VCUBLOK 131
VCUADD (virtual control unit address) 131
VCUSTAT (status) 131
VCUTYPE (type) 131
VDEVBLOK 132
VDEVADD (virtual device address) 132
VDEVCFLG (virtual console flags) 132
VDEVCSW (virtual CSW) 132
VDEVEXTN (virtual spool extension) 133
VDEVFLAG (device dependent information) 132
VDEVIOB (active IOBLOK pointer) 132
VDEVREAL (real device block address) 132
VDEVSFLG (virtual spooling flags) 133
VDEVSTAT (status) 132
VERIFY, control record, ZAP program 393
verifying the existence of saved systems 273
virtual
address 207
block multiplexer channel option 218
console functions, DIAGNOSE instruction 258
CPU 183
operator's console 183
spooling
card reader 204
printer 204
punch 204
tracing 206
virtual=real option 186,212,215
virtual console, operator 183
virtual devices, I/O 183
Virtual Machine Assist feature 47
described 216
querying status of 85
restrictions for use of 218
usage 61,217
used to reduce real supervisor state time 216
with TRACE command 71
Virtual Machine Facility/370 (VM/370)
CMS 287
control program 183
device types in 264
DIAGNOSE instruction in 257
directory 183
load environment 234
program states 191
RSCS 411
VM/VS Handshaking 251
virtual machines
ABEND dump 31
abnormal termination 19,25,31
CAW, displaying 49
creation 183
CSW, displaying 49
described 183
DIAGNOSE instruction usage 257
directory 183
disabled loop 22
procedure 34
disabled looping condition 24
disabled wait 20
procedure 24,36
dispatching scheme 192
enabled loop 22
procedure 34
enabled looping condition 24
enabled wait 21
procedure 24,37
with real timer option 37
without real timer option 37
extended control registers
displaying 49
printing 55
floating-point registers
displaying 49
printing 55
general registers
displaying 49
printing 55
interrupt, handled by CP 183
I/O management
dedicated devices 187
directory 186
shared devices 187
spooled devices 187
I/O operation 209
operating system 183
performance
for time-shared multi-batch machines 235
options 212
preferred 212
PSW 191
displaying 49
printing 55
shared segment operation 249
storage keys, printing 55
storage locations
displaying 49
printing 55
storage management
directory 184
virtual storage 184
time management
conversational user 184
nonconversational user 184
priority of execution 184
timers 254
unexpected results 20,25
procedure 32
virtual storage locations, printing 55
virtual storage 183
management
CP 184
RSCS 417
virtual-to-real address translation 207
VM Assist (see Virtual Machine Assist feature)
VM Monitor 226
collection mechanism 226
considerations 232
data records 438
data volume and overhead 233
header record 437
monitor classes 226
responses to unusual tape conditions 232
tape format and content 437

VMA (see Virtual Machine Assist feature)
VMBLOK 27,36,128
  VCUSTRT (address of VCUBLOK table) 131
  VMCHSTRT (address of VCHBLOK table) 131
  VMCOMND (last command) 128
  VMDSTAT (dispatching status) 128
  VMDVSTRT (address of VDEVBLOK table) 132
  VMEXTINT (external interrupts) 130
  VMIOACTV (active channel mask) 130
  VMICINT (I/O interrupts) 130
  VMPEND (interrupts pending) 130
  VMPSW (virtual PSW) 128
  VMRSTAT (running status) 128
VMFDUMP
  command 107
    usage 107
VM/VS Handshaking 251
  closing CP spool files 252
  miscellaneous enhancements 253
  non-paging mode 253
  pseudo page faults 252
VM/370 (see Virtual Machine Facility/370 (VM/370))
Volume Table of Contents (VTOC), support of 329
VSAM
  CMS support for 361
  data sets, compatibility considerations 363
  device support under CMS 361
  macros supported under CMS 262
  storage requirements for use under CMS 363
  support of 329

W
WAIT macro 325
wait state 35
  CP
    enabled wait 36,103
  CP disabled wait 35
  RSCS
    virtual machine disabled wait 37
    virtual machine enabled wait 38
  virtual machine
    disabled wait messages 36
    enabled wait procedure 37
warm start 197
WRITE macro 330
WTO/WTOR macros 327

X
X (Examine) subcommand
  error messages 164
  format 164
  usage 164
XCTL macro, 325
XDAP macro 325

Z
ZAP
  command (see ZAP command)
  control record
    BASE 392
    comment 395
    DUMP 390
    END 395
    NAME 392
    REP 394
  input control records 390
  service program 388
ZAP command
  described 389
  option output 389

3
3270
  remote, resource identification 396
  virtual console interface 269
3704/3705
  logging on 386
    after NCP has abnormally terminated 388
  sign on procedure for lines in NCP mode 387
3704/3705 Communications Controllers
  generating a VM/370 system to support 381
  introduction 379
  planning considerations 381
3704/3705 control program
  controlling resources of 397
  dumping 397
  Emulation Program (EP) 379
    loading considerations 385
  image saved on disk 382
  loading 382,397
    considerations 385
  NCPDUMP program 406
  Network Control Program (NCP) 379
    loading considerations 386
  Partitioned Emulation Program (PEP) 379
    loading considerations 386
  processing spool dump files 406
  saving 269
  support under VM/370 379
  testing 396

READER'S
COMMENT
FORM

**Title:** IBM Virtual Machine Facility/370:
System Programmer's Guide

**Order No.** GC20-1807-4

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.

Trim Along This Line

## YOUR COMMENTS PLEASE . . .

*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 reveiwed 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)**