File No. S370-36
Order No. GC20-1807-7

Systems

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

Release 6 PLC 17

This publication, intended for VM/370 system programmers, contains:

- Detailed descriptions of procedures, commands, and utility programs useful in debugging as well as guidelines for reading dumps.
- A description of CP and how it works and details of how to modify or better utilize CP.
- A description of CMS and how it works, as well as details of some special features of CMS.
- A description of the Remote Spooling Communications Subsytem (RSCS).

IBM

Eighth Edition (March 1979)

| This edition (GC20-1807-7) together with Technical Newsletters
| GN25-0492, dated August 1, 1979, and GN25-0829, dated April 1, 1981,
| applies to Release 6 PLC 17 (Program Level Change) of the IBM Virtual
Machine Facility/370, and to all subsequent releases unless otherwise
indicated in new editions or Technical Newsletters.

In Part 2 the entire section headed "Functional Information" has been deleted. This information now appears in the IBM Virtual Machine Facility/370: System Logic and Problem Determination Guide Volume 1, SY20-0886.

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

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

It is possible that this material may contain reference to, or information about, IBM products (machines and programs), programming, or services which are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such IBM products, programming, or services in your country.

Publications are not stocked at the address given below; 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 Programming Publications, Dept. G60, P.O. Box 6, Endicott, New York,
| U.S.A. 13760. IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

© Copyright International Business Machines Corporation 1972, 1973, 1974, 1975, 1976, 1977, 1979, 1981

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 programming personnel.

This publication consists of four parts and three 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 sections. The first section, "Introduction to Debugging", tells you how to identify a problem and lists guidelines to follow to find the cause. The second section, "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. 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.

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

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

- "2305" refers to IBM 2305 Fixed Head Storage, Models 1 and 2.
- "3262" refers to the IBM 3262 Printer, Models 1 and 11.
- "3270" refers to a series of display devices, namely, the IBM 3275, 3276 (referred to as a Controller Display Station), 3277, and 3278 Display Stations. A specific device type is used only when a distinction is required between device types.
  Information about display terminal usage also applies to the IBM 3138, 3148, and 3158 Display Consoles when used in display mode, unless otherwise noted.
  Any information pertaining to the IBM 3284 or 3286 Printer also pertains to the IBM 3287, 3288, and 3289 printers, unless otherwise noted.
- "3330" refers to the IBM 3330 Disk Storage, Models 1, 2, or 11; the IBM 3333 Disk Storage and Control, Models 1 or 11; and the 3350 Direct Access Storage operating in 3330/3333 Model 1 or 3330/3333 Model 11 compatibility mode.
- "3340" refers to the IBM 3340 Disk Storage, Models A2, B1, and B2, and the 3344 Direct Access Storage Model B2.
- "3350" refers to the IBM 3350 Direct Access Storage Models A2 and B2 in native mode.
- "370X" refers to IBM 3704 and 3705 Communications Controllers.
- "3705" refers to the 3705 I and the 3705 II unless otherwise noted.
- "2741" refers to the IBM 2741 and the IBM 3767, unless otherwise specified.

---

[1] Trademark of IBM

An expanded glossary is available in the IBM Virtual Machine Facility/370: Glossary and Master Index, GC20-1813.

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

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

Information on the new IBM 3262 Printer, Models 1 and 11, is for planning purposes only until the availability of the product.

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, GC20-1821.

COREQUISITE PUBLICATIONS

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

IBM Virtual Machine Facility/370:

- Planning and System Generation Guide, GC20-1801
- CP Command Reference for General Users, GC20-1820
- CMS Command and Macro Reference, GC20-1818
- CMS User's Guide, GC20-1819.
- Operator's Guide, GC20-1806
- Terminal User's Guide, GC20-1810
- OLTSEP and Error Recording Guide, GC20-1809.

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

- OS/VS Environmental Recording Editing and Printing (EREP) Program, GC28-0772

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

- Remote Spooling Communications Subsystem (RSCS) User's Guide, GC20-1816
- Data Areas and Control Blocks Logic, SY20-0884
- System Logic and Problem Determination, SBOF-3802
  - Volume 1 Control Program (CP), SY20-0886
  - Volume 2 Conversational Monitor System (CMS), SY20-0887
  - Volume 3 Remote Spooling Communication System (RSCS), SY20-0888

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

If the IBM 3850 Mass Storage System is attached to the VM/370 system, the following are corequisite publications:

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

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

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

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

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

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

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

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

SUPPLEMENTAL PUBLICATIONS

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

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

IBM 2821 Control Unit Component Description GA24-3312.

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

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

IBM OS/VS Linkage Editor and Loader, GC26-3813.

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

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

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

Contents

The entries in this Table of Contents are accumulative. They list additions to this publication by the following VM/370 System Control Program Products:

- VM/370 Basic System Extensions, Program Number 5748-XX8
- VM/370 System Extensions, Program Number 5748-XE1

However, the text within the publication is not accumulative; it only relates to the one SCP program product that is installed on your system. Therefore, there may be topics and references listed in this Table of Contents that are not contained in the body of this publication.

Summary of Amendments. xiii

PART 1. DEBUGGING WITH VM/370. 1

INTRODUCTION TO DEBUGGING. 3
How To Start Debugging 3
Does a Problem Exist? 4
Identifying the Problem. 7
Analyzing the Problem. 13
How To Use VM/370 Facilities To Debug. . 18
Abend. 18
CP Abend 18
CMS Abend. 19
Virtual Machine Abend (Other than CMS). 23
Unexpected Results 24
Unexpected Results in CP 24
Unexpected Results in a Virtual Machine 24
Loop 25
CP Disabled Loop 25
Wait 27
Summary of VM/370 Debugging Tools. . . . 32
Comparison of CP and CMS Facilities for Debugging 37
An Overview of VM/370 Commands That Can Be Used for Debugging 38
Commands that Display or Dump Virtual Machine Data. 38
Commands That Set and Query System Features, Conditions, and Events. . . 39
Commands To Collect and Analyze System Information 40
Commands That Trace Events in Virtual Machines. 40
Commands That Alter the Contents of Storage 41
Debugging CP on a Virtual Machine. . . . 42
CP Internal Trace Table. 42
Abend Dumps. 44
How To Print a CP Abend Dump From Tape 45
Reading CP ABEND Dumps 46
Reason for the Abend 46
Collect Information. 47
Register Usage 48
Save Area Conventions. 48
Virtual and Real Control Block Status. 50
Identifying and Locating a Pageable Module. 63
VMDUMP Records: Format and Content . . 63
Locating Logical Dump Records. . . . 64

DEBUGGING WITH CMS 64.2
CMS Debugging Commands 64.2
DEBUG. 65
Nucleus Load Map 66
Load Map 68
Reading CMS Abend Dumps. 68
Reason for the Abend 69
Collect Information. 70
Register Usage 72

PART 2. CONTROL PROGRAM (CP) 73

VM/370 75
Introduction to the VM/370 Control Program 75
Virtual Machine Time Management. . . . 76
Virtual Machine Storage Management . . 76
Virtual Storage Preservation (5748-XX8). 78.1
Virtual Storage Preservation (5748-XE1). 78.1
Virtual Machine I/O Management 79
Spooling Functions 80
Spool File Recovery. 81
CP Commands. 82

PROGRAM STATES 83

USING PROCESSOR RESOURCES. 84
Queue 1. 84
Queue 2. 84
Deadline Priority (5748-XX8) 85
Deadline Priority (5748-XE1) 85
Queue 3 (5748-XX8) 86
Queue 3 (5748-XE1) 86

INTERRUPTION HANDLING. 86
INTERRUPTION HANDLING (5748-XX8) . . . 86.1
INTERRUPTION HANDLING (5748-XE1) . . . 86.1
I/O Interrupts 86
I/O Interrupts (5748-XX8). 86.1
I/O Interrupts (5748-XE1). 86.1
Program Interrupt. 86
Program Interrupt (5748-XX8) 86.1
Program Interrupt (5748-XE1) 86.1
Machine Check Interrupt. 86

Machine Check Interrupt (5748-XX8) . . 86.1
Machine Check Interrupt (5748-XE1) . . 86.1
SVC Interrupt. 87
External Interrupt 87
Synchronous Interrupts In An Attached Processor System. 87
Real I/O Interrupts. 87

PERFORMANCE GUIDELINES 88
General Information. 88
Virtual Machine I/O. 89
Paging Considerations. 90
Locked Pages Option. 91
Reserved Page Frames Option. 92
Virtual=Real Option. 92
VM/370 Performance Options 93
Favored Execution. 93
Priority 95
User Priority (5748-XX8) 95
User Priority (5748-XE1) 95
Reserved Page Frames 95
Virtual=Real 95
Affinity 97
Multiple Shadow Table Support (5748-XE1). 98
Shadow Table Bypass (5748-XE1). . . . 98.1
Virtual Machine Assist Feature 98
(5748-XE1). 98.2
Virtual Machine Assist Feature. 98
Using the Virtual Machine Assist Feature 99
Restricted Use of the Virtual Machine Assist Feature. 99
VM/370 Extended Control-Program Support (ECPS). 99
Using the VM/370 Extended Control-Program Support 101
The Virtual Block Multiplexer Channel Option. 102
Alternate Path Support 102
MVS/System Extensions Support (5748-XE1). 102.1
Low Address Protection Facility (5748-XE1). 102.1
Common Segment Facility (5748-XE1) .102.2
Special MVS Instruction Operation Handling Facilities (5748-XE1). . .102.2
Enabling MVS/System Extensions Support (5748-XE1). 102.2
Single Processor Mode (5748-XE1). . 102.2
Dynamic SCP Transition to or From Native Mode (5748-XE1) 102.3

PERFORMANCE OBSERVATION AND ANALYSIS . .103
Load Indicators. 103
The Indicate Command 103
The Class G INDICATE Command 104
The Class E INDICATE Command 106
The Class E INDICATE FAVORED Command (5748-XX8). 109
The Class E INDICATE FAVORED Command (5748-XE1). 109
The MIGRATE Command (5748-XE1) 109
Querying and Setting the System Resource Management Variables (5748-XX8). 109
Querying and Setting the System Resource Management Variables (5748-XE1). 110
Querying and Setting the Paging Variable (5748-XX8) 110
Querying and Setting the Paging Variable (5748-XE1) 110
The MONITOR Command. 110
The MONITOR Command (5748-XX8) 110.1
The MONITOR Command (5748-XE1) 110.1
Implemented Classes. 119
VM/370 Monitor Response to Unusual Tape Conditions 121
VM/370 Monitor Considerations. 122
VM/370 Monitor Data Volume and Overhead. 123
Load Environments of VM/370. 124

ACCOUNTING RECORDS 127
Accounting Records for Virtual Machine Resource Usage. 127
Accounting Records for Dedicated Devices and Temporary Disk Space. . . .128
Accounting Records for Dedicated Devices and Temporary Disk Space (5748-XX8). 127
Accounting Records for Dedicated Devices and Temporary Disk Space (5748-XE1). 127
Accounting Records for LOGON, AUTOLOG, and LINK Journaling 128
Accounting Records Created by the User .129
Operational Notes. 130
User Accounting Options. 130

GENERATING SAVED SYSTEMS 132
The NAMESYS Macro For Saved Systems. . .132
Using the SAVESYS Command. 134
Shared Segments. 135
Special Considerations for Shared Segments. 135
Discontiguous Saved Segments 135
User Requirements. 136
The NAMESYS Macro for Discontiguous Saved Segments. 137
Loading and Saving Discontiguous Shared Segments 138
How The Interface Works. 139
Shared Segment Protection. 140
Virtual Machine Operation. 142

THE VIRTUAL MACHINE COMMUNICATION FACILITY. 143
Using the Virtual Machine Communication Facility. 145
VMCF Applications. 145
Security and Data Integrity. 146
Performance Considerations 147
General Considerations 148
VMCF Protocol. 148
The SEND Protocol. 148
The SEND/RECV Protocol 150
The SENDX Protocol 151
The IDENTIFY Protocol. 152
Descriptions of VMCF Subfunctions. . . .153
The Control Subfunctions 153
The Data Transfer Functions. 155

Invoking VMCF Subfunctions 159
Diagnose Code X'68' 159
The VMCPARM Parameter List 159
External Interrupt Code X'4001' 164
VMCF User Doubleword 166
DIAGNOSE X'68' Return Codes. 167
Data Transfer Error Codes. 170

SPECIAL MESSAGE FACILITY 171

VM/370 USE OF THE IBM 3850 MSS 172
VM/370 Access to the MASS Storage Control 172
Asynchronous MSS Mount Processing. . . .173
VM/370 Processing of MSS Cylinder Faults. 173
Backup and Recovery of MSS Volumes . . .174
Logical Device Support Facility (5748-XX8). 174
Logical Device Support Facility (5748-XE1). 174

TIMERS IN A VIRTUAL MACHINE. 175
Interval Timer 175
Processor Timer. 176
TOD Clock. 176
Clock Comparator 177
Pseudo Timer 177
Pseudo Timer Start I/O 177
Pseudo Timer DIAGNOSE. 178
CP In Attached Processor Mode. 178
PSA. 178
I/O Handling 179
Signaling. 180
Locking. 180
Affinity 181

DIAGNOSE INSTRUCTION IN A VIRTUAL MACHINE 182
DIAGNOSE Code X'00' -- Store Extended-Identification Code. 183
DIAGNOSE Code X'04' -- Examine Real Storage 184
DIAGNOSE Code X'08' -- Virtual Console Function. 184
DIAGNOSE Code X'08' -- Virtual Console Function (5748-XX8) 184.1
DIAGNOSE Code X'08' -- Virtual Console Function (5748-XE1) 184.1
DIAGNOSE Code X'0C' -- Pseudo Timer. . .186
DIAGNOSE Code X'10' -- Release Pages . .186
DIAGNOSE Code X'14' -- Input Spool File Manipulation. 187
Subcode X'0000' 187
Subcode X'0004' 188
Subcode X'0008' 188
Subcode X'000C' 188
Subcode X'0010' 188
Subcode X'0014' 189
Subcode X'0018' 189
Subcode X'001C'. 189
Subcode X'0020'. 189
Subcode X'0FFF' 190
DIAGNOSE Code X'18' -- Standard DASD I/O 190
DIAGNOSE Code X'1C' -- Clear Error Recording Cylinders 191
DIAGNOSE Code X'20' -- General I/O . . .191
DIAGNOSE Code X'24' -- Device Type and Features. 192
DIAGNOSE Code X'28' -- Channel Program Modification. 194
DIAGNOSE Code X'2C' -- Return DASD Start of LOGREC 195
DIAGNOSE Code X'30' -- Read One Page of LOGREC Data 196
DIAGNOSE Code X'34' -- Read System Dump Spool File. 196
DIAGNOSE Code X'38' -- Read System Symbol Table. 197
DIAGNOSE Code X'3C' -- VM/370 Directory.197
DIAGNOSE Code X'40' -- Clean-Up After Virtual IPL by Device (5748-XX8) . . . 197
DIAGNOSE Code X'40' -- Clean-Up After Virtual IPL by Device (5748-XE1) . . . 197
DIAGNOSE Code X'48' -- Issue SVC 76 From a Second Level VM/370 Virtual Machine 197
DIAGNOSE Code X'4C' -- Generate Accounting Cards for the Virtual User. 197
DIAGNOSE Code X'4C' -- Generate Accounting Records for the Virtual User (5748-XX8) 198
DIAGNOSE Code X'4C' -- Generate Accounting Records for the Virtual User (5748-XE1) 198
DIAGNOSE Code X'50' -- Save the 370X Control Program Image 199
DIAGNOSE Code X'54' -- Control The Function of the PA2 Function Key. . . .199
DIAGNOSE Code X'58' -- 3270 Virtual Console Interface 199
Displaying Data (5748-XX8) 200
Displaying Data (5748-XE1) 200
Full Screen Mode (5748-XX8). 200.1
Full Screen Mode (5748-XE1). 200.1
DIAGNOSE Code X'5C' -- Error Message Editing 200
DIAGNOSE Code X'5C' -- Error Message Editing (5748-XX8). 200.4
DIAGNOSE Code X'5C' -- Error Message Editing (5748-XE1). 200.4
DIAGNOSE Code X'60' -- Determining the Virtual Machine Storage Size. 201
DIAGNOSE Code X'64' -- Finding, Loading, and Purging a Named Segment. .201
DIAGNOSE Code X'68' -- Virtual Machine Communication Facility (VMCF) 203
DIAGNOSE Code X'6C' -- Special Diagnose for Shadow Table Maintenance (5748-XE1). 204
DIAGNOSE Code X'70' -- Activating the Time-of-Day (TOD) Clock Accounting Interface (5748-XE1). 204
DIAGNOSE Code X'74' -- Saving or Loading a 3800 Named System 204
DIAGNOSE Code X'74' -- Saving or Loading a Named System (5748-XE1) . .204.1
DIAGNOSE Code X'78' -- MSS Communication 205
DIAGNOSE Code X'7C' -- Logical Device Support Facility (5748-XX8). 206

DIAGNOSE Code X'7C' -- Logical Device Support Facility (5748-XE1) 206
DIAGNOSE Code X'84' -- Directory Update In-Place. 206
DIAGNOSE Code X'7C' -- Return Codes and Condition Codes (5748-XX8). 206.1
DIAGNOSE Code X'7C' -- Return Codes and Condition Codes (5748-XE1). 206.1
Descriptions of Logical Device Support Facility Subfunctions (5748-XX8). 206.2
Descriptions of Logical Device Support Facility Subfunctions (5748-XE1). 206.2
Initiate: DIAGNOSE Code X'7C' Subfucntion X'0001' (5748-XX8). . .206.2
Initiate: DIAGNOSE Code X'7C' Subfucntion X'0001' (5748-XE1). . .206.2
Accept: DIAGNOSE Code X'7C' Subfucntion X'0002' (5748-XX8). . .206.2
Accept: DIAGNOSE Code X'7C' Subfucntion X'0002' (5748-XE1). . .206.2
Present: DIAGNOSE Code X'7C' Subfucntion X'0003' (5748-XX8). . .206.3
Present: DIAGNOSE Code X'7C' Subfucntion X'0003' (5748-XE1). . .206.3
Terminate: DIAGNOSE Code X'7C' Subfucntion X'0004' (5748-XX8). . .206.3
Terminate: DIAGNOSE Code X'7C' Subfucntion X'0004' (5748-XE1). . .206.3
Terminate (all): DIAGNOSE Code X'7C' Subfucntion X'0005' (5748-XX8). . .206.3
Terminate (all): DIAGNOSE Code X'7C' Subfucntion X'0005' (5748-XE1). . .206.3
External Interupt Code X'2402' (5748-XX8). 206.3
External Interupt Code X'2402' (5748-XE1). 206.3
Logical Device Restrictions (5748-XX8). 206.4
Logical Device Restrictions (5748-XE1). 206.4
DIAGNOSE Code X'84' -- Directory Update In-Place (5748-XX8) 206.4
DIAGNOSE Code X'84' -- Directory Update In-Place (5748-XE1) 206.4

CP CONVENTIONS 211
CP Coding Conventions. 211
CP Loadlist Requirements 214

HOW TO ADD A CONSOLE FUNCTION TO CP. . .215

PRINT BUFFERS AND FORMS CONTROL. 216
Adding New Print Buffer Images 217
UCS Buffer Images. 217
UCSB Images. 219
FOB Images (5748-XX8). 222
FOB Images (5748-XE1). 222
Forms Control Buffer 222
Forms Control Buffer (5748-XX8). . . .222.1
Forms Control Buffer (5748-XE1). . . .222.1
3203 Model 4 and 5 Printer Forms Control and Print Buffer. 223
3203 Model 4 and 5 Printer and 3262 Model 1 and 11 Printer Forms Control and Print Buffer (5748-XX8) . .223
3203 Model 4 and 5 Printer and 3262 Model 1 and 11 Printer Forms Control and Print Buffer (5748-XE1) . .223
UCC Buffer Images. 224

IBM 3800 PRINTING SUBSYSTEM. 226
Using the 3800 Printer as a Dedicated Device. 226
Using the 3800 Printer as a Real Spooling Device 227
Specifying Printer Options 227
Creating Control Tables. 227
Storing and Loading Control Tables . .228
Recovering from I/O Errors 228
Displaying Printer Control Information 228

JOURNALING LOGON, AUTOLOG, AND LINK COMMANDS. 229

SUPPRESSING PASSWORDS ENTERED ON THE COMMAND-LINE. 230

PART 3. CONVERSATIONAL MONITOR SYSTEM (CMS) 231

INTRODUCTION TO CMS. 233
The CMS Command Language 233
The File System. 234
Program Development. 235

INTERRUPT HANDLING IN CMS. 236
SVC Interruptions. 236
Internal Linkage SVCs. 236
Other SVCs. 236
Input/Output Interruptions 237
Terminal Interruptions 238
Reader/Punch/Printer Interruptions . . .238
User-Controlled Device Interruptions . .238
Program Interruptions. 238
External Interruptions 239
Machine Check Interruptions. 239

FUNCTIONAL INFORMATION 240
Register Usage 240
Structure of DMSNUC. 240
USERSECT (User Area) 241
DEVTAB (Device Table). 241
Structure of CMS Storage 241
Free Storage Management. 243
GETMAIN Free Storage Management. . . .244
DMSFREE Free Storage Management. . . .247
Releasing Allocated Storage. 252
DMSFREE Service Routines 252
Error Codes from DMSFRES, DMSFREE, and DMSFRET 254
CMS Handling of PSW Keys 255
CMS SVC Handling 256
SVC Types and Linkage Conventions. . .257
Search Hierarchy for SVC 202 259
User and Transient Program Areas . . .263
Called Routine Start-Up Table. 264
Returning to the Calling Routine . . .264
CMS Interface for Display Terminals. . .267

HOW TO ADD A COMMAND OR EXEC PROCEDURE TO CMS. 269

OS MACRO SIMULATION UNDER CMS. 270
OS Data Management Simulation. 270
Handling Files that Reside on CMS Disks 270
Handling Files that Reside on OS or DOS Disks 271
Simulation Notes 273
Access Method Support. 278
Reading OS Data Sets and DOS Files Using OS Macros 280

DOS/VS SUPPORT UNDER CMS 284
DOS/VSE SUPPORT UNDER CMS (5748-XX8) . .284
DOS/VSE SUPPORT UNDER CMS (5748-XE1) . .284
Hardware Devices Supported 285
CMS Support of DOS/VS Functions. 285
CMS Support of DOS/VSE Functions (5748-XX8). 285
CMS Support of DOS/VSE Functions (5748-XE1). 285
Logical Unit Assignment. 288
DOS/VS Supervisor and I/O Macros Supported by CMS/DOS. 289
DOS/VSE Supervisor and I/O Macros Supported by CMS/DOS (5748-XX8) 289
DOS/VSE Supervisor and I/O Macros Supported by CMS/DOS (5748-XE1) 289
Supervisor Macros. 290
Sequential Access Method -- Declarative Macros. 294
Sequential Access Method -- Imperative Macros 304
DOS/VS Transient Routines. 304
DOS/VSE Transient Routines (5748-XX8). .304
DOS/VSE Transient Routines (5748-XE1). .304
EXCP Support in CMS/DOS. 305
DOS/VS Supervisor Control Blocks Simulated by CMS/DOS. 306
DOS/VSE Supervisor Control Blocks Simulated by CMS/DOS (5748-XX8) 306
DOS/VSE Supervisor Control Blocks Simulated by CMS/DOS (5748-XE1) 306
User Considerations and Responsibilities. 306
DOS/VS System Generation and Updating Considerations. 306
DOS/VSE System Generation and Updating Considerations.(5748-XX8) 306
DOS/VSE System Generation and Updating Considerations.(5748-XE1) 306
VM/370 Directory Entries 307
CMS/DOS Storage Requirements 308
When the DOS/VS System Must Be Online. .308
When the DOS/VSE System Must Be Online (5748-XX8). 308
When the DOS/VSE System Must Be Online (5748-XE1). 308
Performance. 309
Execution Considerations and Restrictions. 309
Tape Label Processing in CMS/DOS (5748-XX8). 309
Tape Label Processing in CMS/DOS (5748-XE1). 309

CMS SUPPORT FOR OS AND DOS VSAM FUNCTIONS 310
Hardware devices Supported 310
DOS/VS Supervisor Macros and Logical Transients Support for VSAM 311
DOS/VSE Supervisor Macros and Logical Transients Support for VSAM (5748-XX8).311
DOS/VSE Supervisor Macros and Logical Transients Support for VSAM (5748-XE1).311
Storage Requirements 312
Data Set Compatibility Considerations. .312
Data Set Compatibility Considerations (5748-XX8). 312.1
Data Set Compatibility Considerations (5748-XE1). 312.1
ISAM Interface Program (IIP) 312
ISAM Interface Program (IIP) (5748-XX8). 312.1
ISAM Interface Program (IIP) (5748-XE1). 312.1

SAVING THE CMS SYSTEM. 313
The CMSSEG Discontiguous Saved Segment .313
CMSSEG Usage Options 313
Saved System Restrictions for CMS. . . .315

CMS BATCH FACILITY 316
Resetting the CMS Batch Facility System Limits. 316
Writing Routines To Handle Special Installation Input. 317
BATEXIT1: Processing User-Specified Control Language. 317
BATEXIT2: Processing the Batch Facility /JOB Control Card. 317
EXEC Procedures for the Batch Facility Virtual Machine 318
Data Security under the Batch Facility .318
Improved IPL Performance Using a Saved System. 318

AUXILIARY DIRECTORIES. 319
How To Add an Auxiliary Directory. . . .319
Generation of the Auxiliary Directory.319
Initializing the Auxiliary Directory .320
Establishing the Proper Linkage. . . .320
An Example of Creating an Auxiliary Directory 321

ASSEMBLER VIRTUAL STORAGE REQUIREMENTS .324
Overlay Structures 324
Prestructured Overlay. 324
Dynamic Load Overlay 326

PART 4. REMOTE SPOOLING COMMUNICATIONS SUBSYSTEM (RSCS). 327

INTRODUCTION TO RSCS 329
Locations And Links. 329
Remote Stations. 329
VM/370 Spool System Interface. 330
RSCS Command Language. 330

STRUCTURE OF RSCS VIRTUAL STORAGE. . . .332
RSCS Supervisor. 333
Supervisor Queue Extension 333
Free Storage 333
System Control Task. 334

Free Storage and Line Drivers. 334
Line Allocation Task 334
Spool File Access Task 334

FUNCTIONAL INFORMATION 335
Virtual Storage Management 335
  Page Allocation. 335
  Queue Element Management 335
File Management. 336
  Tag Slot Queues. 336
  Spool File Access. 337
Task-to-Task Communication 337
RSCS Command Processing. 337
RSCS Message Handling. 338
Interruption Handling. 338
  External Interruptions 338
  SVC Interruptions. 339
  I/O Interruptions. 339

LOGGING I/O ACTIVITY 340
The SML Log Record 341
The NPT Log Record 342

APPENDIXES 343

APPENDIX A: SYSTEM/370 INFORMATION . . .345
Control Registers. 345

APPENDIX B: MULTI-LEAVING. 349
MULTI-LEAVING in VM/370. 349
MULTI-LEAVING Philosophy 349
MULTI-LEAVING Control Specification. . .351
  Record Control Byte (RCB). 352
  Sub-Record Control Byte (SRCB) 353
  String Control Byte (SCB). 355
  Block Control Byte (BCB) 355
  Function Control Sequence (FCS). . . .356

APPENDIX C: VM/370 MONITOR TAPE FORMAT
 AND CONTENT 357
Header Record. 357
Data Records 358

INDEX. 367

FIGURES

Figure 1. Abend Messages.....4
Figure 2. VM/370 Problem Types.....8
Figure 3. Does a Problem Exist?.....15
Figure 4. Debug Procedures for Waits and Loops.....16
Figure 5. Debug Procedures for Unexpected Results and an abend.....17
Figure 6. Summary of VM/370 Debugging Tools.....32
Figure 7. Comparison of CP and CMS Facilities for Debugging...37
Figure 8. CP Trace Table Entries.....44
Figure 9. CP Control Block Relationships.....51
Figure 10. CP Device Classes, Types, Models, and Features.....60
Figure 10.1. VMDUMP Record Format.....64.1
Figure 11. Sample CMS Load Map.....67
Figure 12. CMS Control Blocks.....70
Figure 13. Storage in a Virtual=Real Machine.....96
Figure 13.1. Functions and Instructions that ECPS Supports.....100.1
Figure 14. Virtual Machine Communication Facility (VMCF) Subfunctions.....144
Figure 15. The SEND Protocol.....149
Figure 16. The SEND/RECV Protocol....150
Figure 17. The SENDX Protocol.....151
Figure 18. The IDENTIFY Protocol.....152
Figure 19. VMCF Subfunctions, Parameters, and Return Codes.....163
Figure 20. DIAGNOSE Code X'68' Return Codes.....167
Figure 21. DIAGNOSE Code X'68' Data Transfer Error Codes.170
Figure 21A. Logical Device Support Facility Subfunctions (5748-XX8).....174.1
Figure 21A. Logical Device Support Facility Subfunctions (5748-XE1).....174.1
Figure 22. Formats of Pseudo Timer Information.....177
Figure 23. Storage in a Virtual=Real Machine.....179
Figure 24. Addressable Storage Before and After a LOADSYS Function.....202
Figure 25. UCSB Associative Field Chart.....220
Figure 26. Devices Supported by a CMS Virtual Machine.....242
Figure 26. Devices Supported by a CMS Virtual Machine (5748-XX8).....242.1
Figure 26. Devices Supported by a CMS Virtual Machine (5748-XE1).....242.1
Figure 27. CMS Storage Mapp.....245
Figure 28. CMS Command (and Request) Processing.....261
Figure 29. PSW Fields When Called Routine Starts.....264
Figure 30. Register Contents When Called Routine Starts.....264
Figure 31. Simulated OS Supervisor Calls.....272
Figure 32. Summary of Changes to CMS Commands to Support CMS/DOS.....286
Figure 33. Physical IOCS Macros Supported by CMS/DOS.....290
Figure 34. DOS/VS Macros Supported Under CMS.....291
Figure 34. DOS/VSE Macros Supported Under CMS (5748-XX8).....291
Figure 34. DOS/VSE Macros Supported Under CMS (5748-XE1).....291
Figure 35. CMS/DOS Support of DTFCD Macro.....295
Figure 36. CMS/DOS Support of DTFCN Macro.....296
Figure 37. CMS/DOS Support of DTFDI Macro.....297
Figure 38. CMS/DOS Support of DTFMT Macro.....299
Figure 39. CMS/DOS Support of DTFPR Macro.....301
Figure 40. CMS/DOS Support of DTFSD Macro.....302
Figure 41. DOS/VS VSAM Macros Supported by CMS.....311
Figure 41. DOS/VSE VSAM Macros Supported by CMS (5748-XX8).....311
Figure 41. DOS/VSE VSAM Macros Supported by CMS (5748-XE1).....311
Figure 42. An Overlay Structure.....324
Figure 43. RSCS Command Summary.....331
Figure 44. RSCS Storage Allocation....332
Figure 45. Control Register Allocation.....345
Figure 46. Control Register Assignments.....346
Figure 47. The Extended Control PSW (Program Status Word).....348
Figure 48. A Typical MULTI-LEAVING Transmission Block.....350

Summary of Amendments
for GC20-1807-7
VM/370 Release 6 PLC 17

MISCELLANEOUS

Changed: Documentation Only

This Technical Newsletter incorporates minor technical and editorial changes.

Summary of Amendments
for GC20-1807-7
VM/370 Release 6 PLC 4

3031AP EXTENDED CONTROL PROGRAM SUPPORT

New: Program Feature

The 3031 and 3031AP now provide Extended Control Program Support for specific instructions and VM/370 functions.

CP DUMP SERVICES FOR VIRTUAL MACHINES

New: Program Feature

A new command, VMDUMP, dumps virtual storage to a specified reader spool file. The dump is in a format that is acceptable as input to the VM/Interactive Problem Control System Extensions program product. Installations that have not installed this program product may process the dump with a user-written program.

Summary of Amendments
for GC20-1807-7
VM/370 Release 6 PLC 1

3800 PRINTING SUBSYSTEM

New: Program Feature

VM/370 now supports the 3800 printing subsystem as a dedicated device or as a real spooling device. A new diagnose code (X'74') allows an installation to save or to load a 3800 named system. This support is described in Part 2.

SPECIAL MESSAGE FACILITY

New: Program Feature

The special message facility allows one virtual machine to send messages to another virtual machine by issuing a new command, SMSG. The special message facility is described in Part 2.

3850 MSS SUPPORT

New: Program Feature

Virtual machines may now access mass storage volumes that contain VM/370 minidisks, or they may access entire mass storage volumes. A new diagnose code (X'78') enables MSS to communicate with VM/370. This support is described in Part 2.

DIRECTORY UPDATE IN-PLACE

New: Program Feature

Now, a new diagnose code (X'84') enables a virtual machine to update the VM/370 directory. This support is described in Part 2.

LOGON, AUTOLOG, AND LINK JOURNALING

New: Program Feature

Now, VM/370 will optionally attempt to detect and record certain occurrences of the LOGON, AUTOLOG, and LINK commands. This support is described in Part 2. Also, the topic "Accounting Records", in Part 2, has been updated to include new record types used by this facility.

SUPPRESSING PASSWORDS ENTERED ON THE COMMAND-LINE

New: Program Feature

Installations may optionally request that VM/370 reject LOGON or LINK commands when the password is entered on the same line as the command. This support is described in Part 2.

MONITORING FACILITIES

New: Program Feature

The VM/370 Monitor command has been changed as follows:

- New operands have been added to the INTERVAL parameter and to the LIMIT parameter.
- A new parameter, SEEKS, is now supported.

These changes are described in Part 2.

Also, the content of the VM monitor tape has been changed. This change is described in Appendix C.

SHARED SEGMENT PROTECTION

New: Program Feature

Now, VM/370 allows an installation to optionally protect or not protect shared segments. A new parameter has been added to the NAMESYS macro for this support. Shared segment protection and the NAMESYS macro are discussed in Part 2.

EDITORIAL UPDATES

Changed: Documentation

Extensive editorial updates have been made throughout this publication.

April 1, 1981

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

Introduction to Debugging

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.

Information that explains how to debug CP or CMS is contained in this book; information explaining how to debug applications programs is in the VM/370 CMS User's Guide. For information that explains how to use the VM/370 Interactive Problem Control System (IPCS) for debugging, refer to the VM/370 Interactive Problem Control System (IPCS) User's Guide

If an IPCS problem is caused by a virtual machine operating system (other than CMS and RSCS), refer to the publications pertaining to that operating system for specific information. However, use the CP debugging facilities, such as the CP commands, to perform the recommended debugging procedures discussed in the other publication.

If it becomes necessary to apply a PTF (Program Temporary Fix) to a component of VM/370, 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) DMKDMP908I SYSTEM FAILURE CODE xxxxxx	CP abend, system dumps to disk. Restart is automatic.
Optional Messages:	
DMKDMP905W SYSTEM DUMP FAILURE; PROGRAM CHECK DMKDMP906W SYSTEM FAILURE; MACHINE CHECK, RUN SEREP 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 DMKCKP901W SYSTEM RECOVERY FAILURE; MACHINE CHECK, RUN SEREP DMKCKP902W SYSTEM RECOVERY FAILURE; FATAL I/O ERROR – NUCL CYL – WARM CYL DMKCKP922W SYSTEM RECOVERY FAILURE; INVALID SPOOLING DATA DMKCKP910W SYSTEM RECOVERY FAILURE; INVALID WARM START CYLINDER DMKCKP911W SYSTEM RECOVERY FAILURE; WARM START AREA FULL	If the checkpcint 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 DMKCKS912W SYSTEM RECOVERY FAILURE; VOLID xxxxxx NOT MOUNTED DMKCKS915E PERMANENT I/O ERROR ON CHECKPOINT CYLINDER DMKCKS916E ERROR ALLOCATING SPOOL FILE BUFFERS 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 DMKWRM903W SYSTEM RECOVERY FAILURE; VOLID xxxxxx ALLOCATION ERROR CYLINDER xxx DMKWRM904W SYSTEM RECOVERY FAILURE; INVALID WARM START DATA DMKWRM912W SYSTEM RECOVERY FAILURE; VOLID xxxxxx NOT MOUNTED 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 DMKCKP960I SYSTEM WARM START DATA SAVED DMKCKP961W SYSTEM SHUTDOWN COMPLETE	CP abend, system dumps to tape or printer. The system stops; the operator must IPL the system to start again.
Optional Messages	
DMKDMP905W SYSTEM DUMP FAILURE; PROGRAM CHECK DMKDMP906W SYSTEM DUMP FAILURE; MACHINE CHECK, RUN SEREP DMKDMP907W SYSTEM DUMP FAILURE; FATAL I/O ERROR	If the dump program encounters a program check, a machine check, or fatal I/O error, a message is issued indicating the error. CP enters the wait state with code 003 in the PSW. If the dump cannot find a defined dump device and if no printer is defined for the dump, CP enters a disabled wait state with code 004 in the PSW.
	CP termination with wait state.
DMKMCH610W MACHINE CHECK SUPERVISOR DAMAGE DMKMCT610W MACHINE CHECK SUPERVISOR DAMAGE	The machine check handler encountered an unrecoverable error with the VM/370 control program.
DMKMCH611W MACHINE CHECK SYSTEM INTEGRITY LOST DMKMCT611W 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
DMKMCH612W MACHINE CHECK; TIMING FACILITIES DAMAGE; RUN SEREP	An error has occurred in the timing facilities. Probable hard error.
DMKMCT620I MACHINE CHECK; ATTACHED PROCESSOR NOT BEING USED	A malfunction alert, clock clock error or instruction processing error occurred on the attached processor. The system continues to run in uniprocessor mode.
	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.
DMKMCH622W MACHINE CHECK; MULTIPLE CHANNEL ERRORS	There was a group error machine check from which the machine check handler could not recover. CP enters a wait state with code 001 in 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 Refer to OS and DOS publications 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 processor 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 DMKDMP908I SYSTEM FAILURE, CODE xxxxxx appears on the processor console. In this instance, the system dump device is a disk, so the system dumps to disk and automatically restarts. If an error occurs in the dump, checkpoint, or warmstart program, CP enters the wait state after issuing one or more of the following messages: DMKDMP905W SYSTEM DUMP FAILURE; PROGRAM CHECK DMKDMP906W SYSTEM DUMP FAILURE; MACHINE CHECK, RUN SEREP DMKDMP907W SYSTEM DUMP FAILURE; FATAL I/O ERROR DMKCKP900W SYSTEM RECOVERY FAILURE; PROGRAM CHECK DMKCKP901W SYSTEM RECOVERY FAILURE; MACHINE CHECK, RUN SEREP DMKCKP902W SYSTEM RECOVERY FAILURE; FATAL I/O ERROR DMKCKP922W SYSTEM RECOVERY FAILURE; INVALID SPOOLING DATA DMKCKP910W SYSTEM RECOVERY FAILURE; INVALID WARM START CYLINDER DMKCKP911W SYSTEM RECOVERY FAILURE; WARM START AREA FULL DMKCKS903W SYSTEM RECOVERY FAILURE; VOLID xxxxxx ALLOCATION ERROR CYLINDER xxx n DMKCKS912W SYSTEM RECOVERY FAILURE; VOLID xxxxxx NOT MOUNTED DMKCKS915E PERMANENT I/O ERROR ON CHECKPOINT CYLINDER DMKCKS917E CHECKPOINT CYLINDER INVALID; CLEAR STORAGE AND COLD START DMKWRM921W SYSTEM RECOVERY FAILURE; UNRECOVERABLE I/O ERROR DMKWRM903W SYSTEM RECOVERY FAILURE; VOLID xxxxxx ALLOCATION ERROR CYLINDER xxx DMKWRM904W SYSTEM RECOVERY FAILURE; INVALID WARM START DATA DMKWRM912W SYSTEM RECOVERY FAILURE; VOLID xxxxxx NOT MOUNTED
	CP abend	The following messages appear on the processor console: DMKDMP908I SYSTEM FAILURE, CODE xxxxxx DMKDMP960I SYSTEM WARM START DATA SAVED DMKDMP961W SYSTEM SHUTDOWN COMPLETE

Figure 2. VM/370 Problem Types (Part 1 of 6)

Problem Type	Where Abend Occurs	Distinguishing Characteristics
Abend (cont.)	CP abend (cont.)	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: DMKDMP905W SYSTEM DUMP FAILURE; PROGRAM CHECK DMKDMP906W SYSTEM DUMP FAILURE; MACHINE CHECK, RUN SEREP DMKDMP907W SYSTEM DUMP FAILURE; FATAL I/O ERROR DMKCKP900W SYSTEM RECOVERY FAILURE; PROGRAM CHECK DMKCKP901W SYSTEM RECOVERY FAILURE; PROGRAM CHECK RUN SEREP DMKCKP902W SYSTEM RECOVERY FAILURE; FATAL I/O ERROR DMKCKP910W SYSTEM RECOVERY FAILURE; INVALID WARM START CYLINDER DMKCKP911W SYSTEM RECOVERY FAILURE; WARM START AREA FULL
	CP termination with wait state	An unrecoverable machine check error has occurred. One of the following messages: DMKMCH610W MACHINE CHECK SUPERVISOR DAMAGE DMKMCT610W MACHINE CHECK SUPERVISOR DAMAGE DMKMCH611W MACHINE CHECK INTEGRITY LOST DMKMCT611W MACHINE CHECK INTEGRITY LOST DMKMCH612W MACHINE CHECK; TIMING FACILITIES DAMAGE; RUN SEREP DMKMCT612W MACHINE CHECK; TIMING FACILITIES DAMAGE; RUN SEREP appears on the processor console. The system enters a wait state. A machine check occurred on the attached processor. The message: DMKMCT620I MACHINE CHECK; ATTACHED PROCESSOR NOT BEING USED appears on the console. The system continues in uniprocessor mode.

Figure 2. VM/370 Problem Types (Part 2 of 6)

Problem Type	Where Abend Occurs	Distinguishing Characteristics
Abend (cont.)	CP termination without automatic restart	An unrecoverable channel check error has occurred. The message: DMKCCH603W CHANNEL ERROR, RUN SEREP, RESTART SYSTEM appears on the processor console, and CP enters wait state.
	Virtual machine abend (CMS)	The CMS message DMSABM148T SYSTEM ABEND xxx CALLED FROM xxxxxx appears on the terminal. The system stops and waits for a command to be entered on the terminal. 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. 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: DMKMCH616I MACHINE CHECK; USER userid TERMINATED DMKCCH604I CHANNEL ERROR; DEV xxx; USER userid; MACHINE RESET is sent to the system operator at the processor console. Also, the virtual user is notified by one of the following messages that his virtual machine was terminated or reset: DMKMCH619I MACHINE CHECK; OPERATION TERMINATED DMKCCH606I CHANNEL ERROR; OPERATOR TERMINATED
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.

Figure 2. VM/370 Problem Types (Part 3 of 6)

Problem Type	Where Abend Occurs	Distinguishing Characteristics
Wait	Disabled CP wait	The processor 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 DMKMCH610W MACHINE CHECK SUPERVISOR DAMAGE DMKMCT610W MACHINE CHECK SUPERVISOR DAMAGE DMKMCH611W MACHINE CHECK SYSTEM INTEGRITY LOST DMKMCH612W MACHINE CHECK; TIMING FACILITIES DAMAGE; RUN SEREP DMKMCH612W MACHINE CHECK; TIMING FACILITIES DAMAGE; RUN SEREP appears on the processor console, a machine check (probable hardware error) caused the CP disabled wait state. If the message DMKCCH603W CHANNEL ERROR, RUN SEREP, RESTART SYSTEM appears on the processor console, a channel check (probable hardware error) caused the CP disabled wait state. If the message DMKCPI955W INSUFFICIENT STORAGE FOR VM/370 appears on the processor console, the control program has entered a disabled wait state with code 00D in the PSW. Either the generated system is larger than the real machine size, or a hardware machine malfunction prevents VM/370 from using the necessary amount of storage. If the message DMKPAG415E CONTINUOUS PAGING ERRORS FROM DASD xxx appears on the processor 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. 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.

Figure 2. VM/370 Problem Types (Part 4 of 6)

Problem Type	Where Abend Occurs	Distinguishing Characteristics
Wait (cont.)	Enabled CP wait	The processor 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: DMKDSP450W CP ENTERED; DISABLED WAIT PSW DMKDSP451W CP ENTERED; INVALID PSW DMKDSP452W CP ENTERED; EXTERNAL INTERRUPT LOOP DMKDSP453W CP ENTERED; PROGRAM INTERRUPT LOOP
	Enabled virtual machine wait	A PSW enabled for I/O 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. 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 DMKDSP450W CP ENTERED; DISABLED WAIT PSW If, in addition, the message DMTINI402T IPL DEVICE READ I/O ERROR appears on the RSCS console, an unrecoverable error has occurred while reading the RSCS nucleus from DASD storage. RSCS enters a disabled wait state with a code of 011 in the PSW. If a program check occurs before the program check handler is activated, RSCS enters a disabled wait state with a code of 007 in the PSW.

Figure 2. VM/370 Problem Types (Part 5 of 6)

Problem Type	Where Abend Occurs	Distinguishing Characteristics
Wait (cont.)	Disabled RSCS wait (cont.)	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: DMTREX090T PROGRAM CHECK IN SUPERVISOR — RSCS SHUTDOWN DMTREX091T INITIALIZATION FAILURE — RSCS SHUTDOWN
	Enabled RSCS wait	RSCS has no task ready for execution. A PSW, enabled for external and I/O interrupts, is loaded with a wait code of all zeros.
Loop	CP disabled loop	The processor console wait light is off. The problem state bit of the real PSW is off. No I/O interrupts are accepted.
	Virtual machine disabled loop	The program is taking longer to execute than anticipated. Signaling attention from the 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 6 of 6)

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.

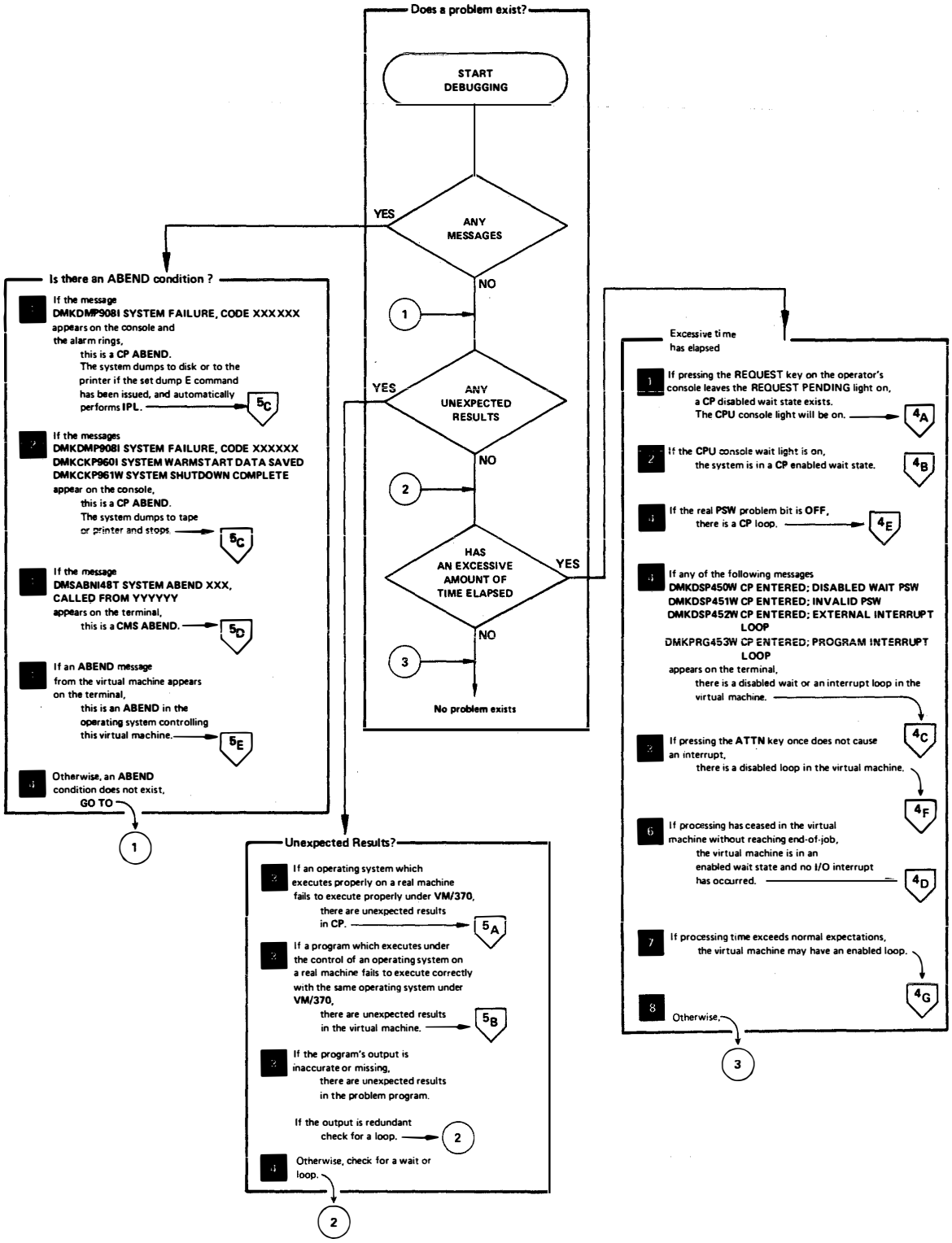


Figure 3. Does a Problem Exist?

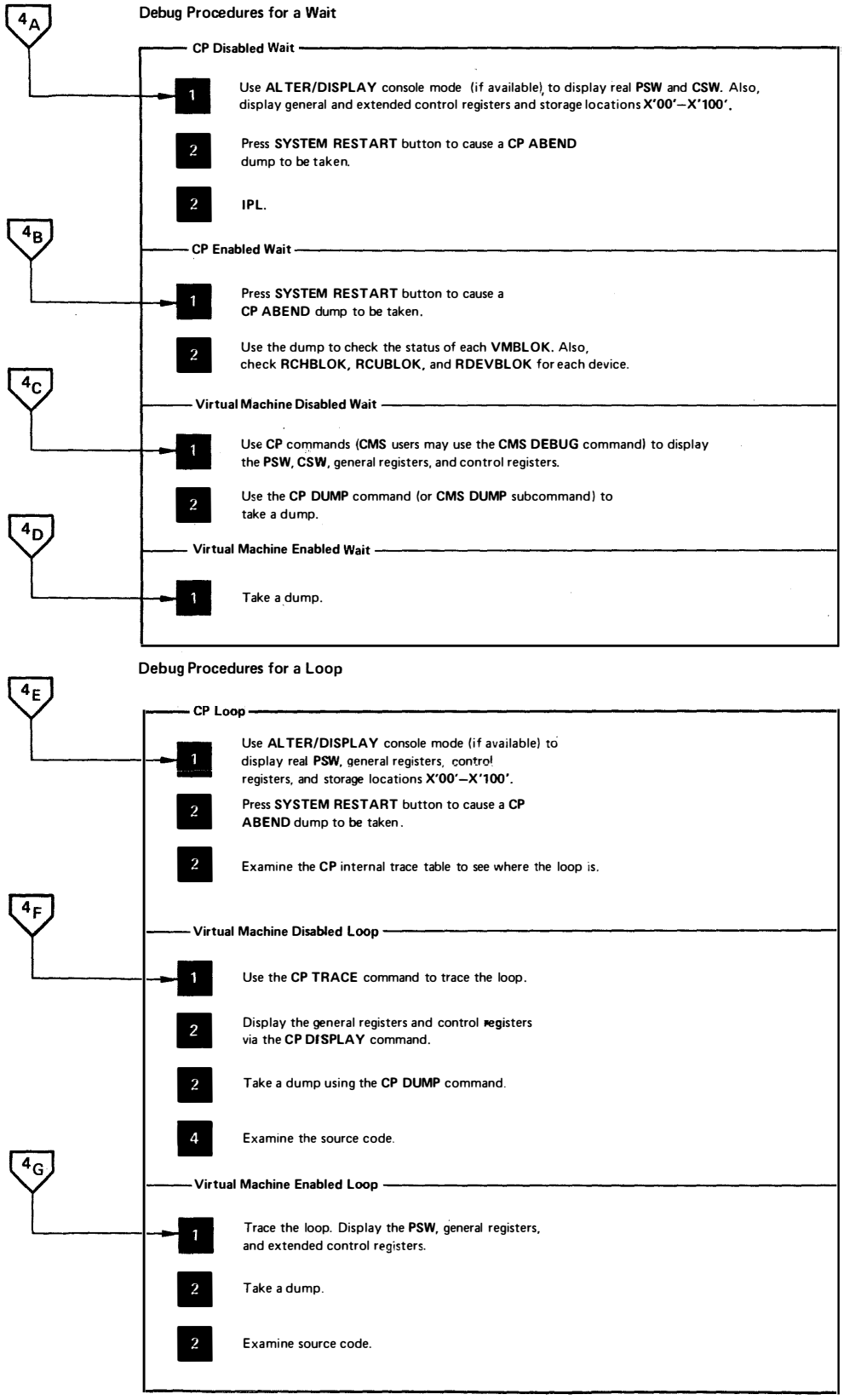
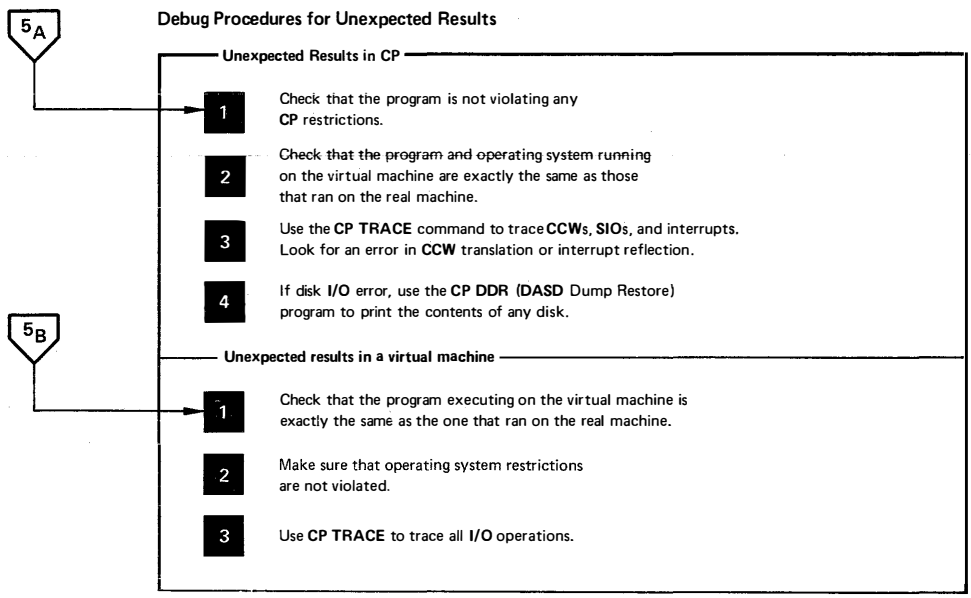


Figure 4. Debug Procedures for Waits and Loops



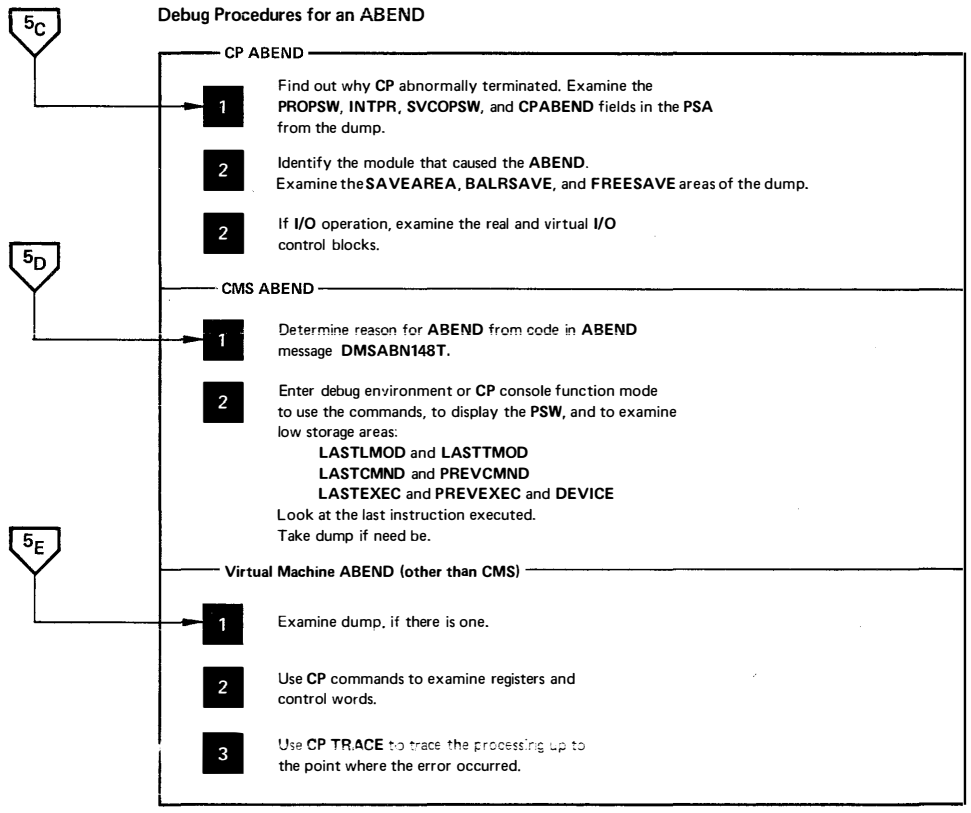


Figure 5. Debug Procedures for Unexpected Results and an Abend

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 a loop condition, you will want to examine the PSW. For a CP loop, you have to use the operator's console to display the PSW, but for a virtual machine loop you can display the PSW via the CP DISPLAY command.

The following sections describe specific debugging procedures for the various error conditions. The procedures 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

Three types of abnormal terminations (ABEND) can occur on VM/370: CP abends, CMS abends, or virtual machine abends. The following description provides guidelines for debugging each type of ABEND.

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 determine why the control program terminated and then determine how to correct the condition. See the "Reading CP Abend Dumps" discussion for detailed information on reading a CP abend dump.

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

1. Program Check in CP

   Examine the PROPSW and INTPR fields in the prefix storage area (PSA) to determine the failing module.

2. Module Issuing an SVC 0

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

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

3. Operator Pressing SYSTEM RESTART Button on Processor 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.

   Note: The conditions that cause an abnormal termination of an attached processor configuration are the same as those that cause a uniprocessor configuration to abnormally terminate.

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

Examine the real and virtual control blocks to find the status of I/O operations. Figure 9 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.

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 through F). The possible programming exceptions are:

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

2. ABEND Macro

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

3. Halt Execution command (HX)

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

4. System Abend

   A CMS system routine can abnormally terminate by issuing the DMSABN macro. The first three hexadecimal digits of the system abend code 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 through FFF) that appears in the DMSABN148T 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 with 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 amont 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, use the DEBUG subcommands or CP commands to examine the PSW and specific areas of low storage. For instructions on how to use the CMS debug commands, see "CMS Debugging Commands" in this section. For instructions on how to use the CP commands, see an "An Overview of VM/370 Commands that can be Used for Debugging" in this section. See 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 with the current CMS load map trying to determine the module that caused the abend. The CMS storage-resident nucleus routines reside in fixed storage locations.

   Also check the interruption code in the PSW.

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

Field	Contents
LASTLMOD	Contains the name of the last module loaded into storage via the LOADMOD command.
LASTTMOD	Contains the name of the last module loaded into the transient area.
LASTCMND	Contains the name of the last command issued.
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 termination of the operating system on a real machine. Refer to publications for that operating system for debugging information. However, all of the CP debugging facilities may be used to help you gather the information you need. Because certain operating systems (OS/VS1, OS/VS2, and DOS/VS) manage their virtual storage themselves, CP commands that examine or alter virtual storage locations should be used only in virtual=real storage space with OS/VS1, OS/VS2, and DOS/VS.

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

The VMDUMP command dumps virtual storage to a specified virtual machine's reader spool file. Installations that have installed the VM/Interactive Problem Control System (IPCS) Extensions program product may use it to process the dump. Other installations may process the dump with a user-written program.

If you choose to run a 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 may 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 non-recoverable machine check occurs in that virtual machine. Hardware errors usually cause this type of virtual machine termination. The following message:

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

appears on the processor console.

If the message:

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

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

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

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 VM/370 Planning and System Generation Guide for a list of the restrictions.

Next, be sure that the program and operating system running on the virtual machine are the same as those that ran on the real machine. Check for:

- The same job stream
- The same copy of the operating system (and program)
- The same libraries

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

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

RESTORE -- transfers data from tapes created by DDR DUMP to a direct access device. The direct access device to which the data is being restored must be the same type of device as the direct access device originally containing that data.

COPY -- copies data from one device to another device of the same type. Data may be reordered, by cylinder, when copied from disk to disk. In order to copy one tape to another, the original tape must have been created by the DDR DUMP function.

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

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

CMS users should refer to the "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 processor operator should perform the following sequence when gathering information to find the cause of a disabled loop.

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

   On an attached processor system, you must add the prefix value for the PSA of each processor to display, dump, or alter low core storage for each processor, or use the M or N operand prefixes described under the DCP, DMCP, and STCP commands.

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 processor 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 nct.

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 that exist only on the virtual machine's paging device.

Virtual Machine Enabled Loop

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

1. Use the CP TRACE 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 cn 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 Contrcl Program is in a wait state, its processing ceases.

CP Disabled Wait

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

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

Three types of severe machine checks can cause the VM/370 Control Program to terminate or cause a CP disabled wait state.

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

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

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

```
DMKCCH603 CHANNEL ERROR, RUN SEREP, RESTART SYSTEM

DMKMCH612W MACHINE CHECK TIMING FACILITIES DAMAGE; RUN SEREP

DMKMCT612W MACHINE CHECK TIMING FACILITIES DAMAGE; RUN SEREP
```

If an unrecoverable machine check occurs in the control program, the message

DMKMCH610W MACHINE CHECK SUPERVISOR DAMAGE

--or--

DMKMCT610W MACHINE CHECK SUPERVISOR DAMAGE

appears on the processor console. The control program is terminated and enters a wait state 001 or wait state 013.

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

DMKMCH611W MACHINE CHECK SYSTEM INTEGRITY LOST

--or--

DMKMCT611W MACHINE CHECK SYSTEM INTEGRITY LOST

appears on the processor console. The control program is terminated and enters wait state 001 or wait state 013.

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

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

- The generated system is larger than the real machine size

--or--

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

The message

DMKCPI955W INSUFFICIENT STORAGE FOR VM/370

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

If more than one paging device is available, disable the device on which the hardware errors are occurring and IPL the system again. If the VM/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 processor operator to record the needed information.

1. Using the alter/display mode of the processcr 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 the 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

DMKPRG453W   CP ENTERED; PROGRAM INTERRUPT LOOP
```

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

- PSW
- CSW
- General registers
- Control registers

Then use the CP DUMP 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 decreased while the virtual machine is in a wait state, it cannot cause the external interrupt. A "real timer" runs in both the problem state and wait state and can cause an external interrupt which will allow processing to resume. The clock comparator can also cause an external interrupt.

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. It may also be the result of an incorrect virtual DASD 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 and that the real DASD 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 zeros. 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 for debugging programs in a virtual machine. The CP and CMS commands are classified by the function they perform.

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

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

Function	Comments	CP Command	CMS Command
Dump data	VMDUMP provides the same information that DUMP provides but in a different format; the format is compatible with the VM/Interactive Problem control System Extensions program product.	VMDUMP [hexloc1 | 0] [-|:] [hexloc2 | END] {.} [bytecount | END] [TO * | TO userid | SYSTEM] [FORMAT vmtype] [DSS] [*dumpid]	

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

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

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 CAW
	Display contents of CSW	DISPLAY CSW	DEBUG CSW
Store data	Store specified information into consecutive storage locations without alignment	STORE Shexloc hexdata...	DEBUG STORE {symbol / hexloc} hexinfo[hexinfo[hexinfo]]
	Store specified words of information into consecutive fullword storage locations	STORE {hexloc / Lhexloc} {hexword1[hexword2...]}	
	Store specified words of information into consecutive general registers	STORE Greg hexword1 [hexword2...]	DEBUG SET GPR reg hexinfo[hexinfo]
	Store specified words of information into consecutive floating-point registers	STORE Yreg hexword1 [hexword2...]	

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 SET PSW hexinfo [hexinfo]
	Store information in CSW		DEBUG SET CSW hexinfo [hexinfo]
	Store information in CAW		DEBUG 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 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 storage contents to the printer	The dump is printed in hexadecimal format with EBCDIC translation. The storage address of the first byte of each line is identified at the left. The control blocks are formatted.	The dump is printed in hexadecimal format. The storage address of the first byte of each line is identified at the left. The contents of general and floating-point registers are printed at the beginning of the dump.
Displaying the contents of storage and control registers at the terminal	The display is typed in hexadecimal format with EBCDIC translation. The CP command displays storage keys, floating-point registers and control registers.	The display is typed in hexadecimal format. The CMS commands do not display storage keys, floating-point registers, or control registers, as the CP command does.
Storing information	The amount of information stored by the CP command is limited only by the length of the input line. The information can be fullword aligned when stored. CP stores data in the PSW, but not in the CAW or CSW. However, data can be stored in the CSW or CAW by specifying the hardware address in the STORE command. CP also stores the status of the virtual machine in the extended 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: • All interrupts, instructions, and branches • SVC interrupts • I/O interrupts • Program interrupts • External interrupts • Privileged instructions • All user I/O operations • Virtual and real CCW's • All instructions The CP trace is interactive. You can stop and display other fields.	CMS traces all SVC interrupts. CMS displays the 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

An Overview of VM/370 Commands that Can Be Used for Debugging

The VM/370 Control Program provides interactive commands that control the VM/370 system and enable the user to control his virtual machine and associated control program facilities. The virtual machine operator using these commands can gather much the same information about his virtual machine as the operator of a real machine gathers using the processor console.

Several of these commands (for example, STORE or DISPLAY) examine or alter virtual storage locations. When CP is in complete control of virtual storage (as in the case of DOS, MFT, MVT, PCP, CMS, and RSCS) these commands execute as expected. However, when the operating system in the virtual machine itself manipulates virtual storage (as in the case of OS/VS1, OS/VS2, or DOS/VS) these CP commands should not be used.

This section presents an overview of the VM/370 commands used for debugging. It supplements the preceding section which discussed debugging procedures and techniques. Instructions for using the commands discussed in this section are in the following publications:

- VM/370 CP Command Reference for General Users
- VM/370 Operator's Guide
- VM/370 CMS Command and Macro Reference

The following categories of commands are discussed:

- Commands that display VM/370 control information
- Commands that set and query system features, conditions, and events
- Commands that collect and analyze system information
- Commands that trace events in virtual machines
- Commands that alter the contents of storage

COMMANDS THAT DISPLAY OR DUMP VIRTUAL MACHINE DATA

Commands that display or dump virtual machine data are: DUMP, VMDUMP, DISPLAY, DCP, and DMCP.

The DUMP and DISPLAY commands of CP are privilege class G commands and are used to display control information describing the status of virtual machines.

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

- Virtual program status word (PSW)
- General registers
- Floating-point registers
- Control registers (if your VM/370 directory has the ECMODE option)
- Storage keys
- Virtual storage locations (first-level storage only)

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

- Virtual storage locations (first-level storage only)
- Storage keys
- General registers
- Floating-point registers

- Control registers
- Program status word (PSW)
- Channel address word (CAW)
- Channel status word (CSW)

The DCP and DMCP commands of CP are privilege class C and E commands and are used to display real storage locations. The DMCP command spools the contents of real storage to your virtual printer. The DCP command displays at your terminal the contents of real storage locations.

The class G VMDUMP command dumps virtual storage to a specified reader spool file. VMDUMP provides the same dump information that the DUMP command provides but in a different format. For example, if a byte of storage contains X'00', DUMP records it in printable format, X'F0F0'; VMDUMP records it as it appears in storage, X'00'. The VM/Interactive Problem Control System Extensions program product can process records written by VMDUMP. For a description of the format and contents of the VMDUMP records, see "VMDUMP Records: Format and Content" in this section.

COMMANDS THAT SET AND QUERY SYSTEM FEATURES, CONDITIONS, AND EVENTS

The SYSTEM and SET commands set system-controlled functions and events; the QUERY command allows you to determine the status of those settings.

The SYSTEM command is a privilege class G command that simulates the RESET and RESTART buttons on the real computer console. It can also be used to clear storage.

The functions of the SET command are described in detail in the VM/370 CP Command Reference for General Users. For debugging, the SET command provides the MSG, WNG, and EMSG operands. These provide messages that may be useful while you are debugging.

The SET MSG function determines whether you receive messages set by other users via the MSG command. Also, the MSG operand determines whether you receive messages from CP when other users spool reader, printer, or punch files to your virtual machine.

The SET SMSG command turns on or off a virtual machines special message flag. If the virtual machine has issued DIAGNOSE Code X'68' (Authorize), this flag determines whether the virtual machine accepts or rejects messages sent via the SMSG command -- when the flag is on, messages are accepted.

The SET WNG function determines whether you receive warning messages from the system operator.

The SET EMSG function controls error message handling. The EMSG operand gives you the ability to specify that you want message code, message text, or both to be displayed at your terminal. You can also specify that no messages be displayed (except in the case where you have spooled your console output).

When you are debugging, it is useful to have all messages displayed at your terminal.

The QUERY command displays the status of features and conditions set by the SET command for your virtual machine. ON is the default for the MSG, WNG, and EMSG operands of the SET command; OFF is the default for the SMSG operand. To verify these settings, use the QUERY command.

COMMANDS TO COLLECT AND ANALYZE SYSTEM INFORMATION

This section discusses five commands to collect and analyze system information when you are debugging. These are the ADSTOP and BEGIN commands and the LOCATE, MONITOR, and TRACE commands.

Stopping Virtual Machine Execution at a Specific Address

The ADSTOP command stops the execution of a virtual machine at a specific address; BEGIN causes the virtual machine to resume execution.

Execution halts when the instruction at the address specified in the ADSTOP command is reached. At this point, you may invoke other CP debugging commands.

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

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

To successfully set an address stop, the virtual instruction address must be in real storage at the time the ADSTOP command is issued.

Locating CP Control Blocks in Storage

Use the LOCATE command to find the address of CP control blocks associated with a particular user, a user's device, or a real system device. The control blocks and their functions are described in the VM/370 Data Areas and Control Block Logic.

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

COMMANDS THAT TRACE EVENTS IN VIRTUAL MACHINES

Use the TRACE command to trace the following virtual machine events:

- SVC interruption
- I/O interruption
- Program interruption
- External interruption

- Non-I/O privileged instructions
- SIO, SIOF, TIO, CLRIO, HIO, HDV, and TCH instructions.
- Branch instructions
- CCW and CSW instructions

The results collected by the TRACE command are spooled to your virtual printer and to your terminal and/or real printer.

COMMANDS THAT ALTER THE CONTENTS OF STORAGE

You can use the STORE, STCP, and ZAP commands to alter the contents of storage.

Altering the Contents of Virtual Machine Storage

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

- Virtual machine storage locations (first-level virtual storage only)
- General registers
- Floating-point registers
- Control registers (if available)
- Program Status Word

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

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

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

Altering the Contents of Real Storage

Use the STCP command to alter the contents of real storage. The STCP command cannot alter the real PSW or real registers.

Modifying or Dumping CMS MODULE, LOADLIB, or TXTLIB Files

Use the ZAP command to modify or dump MODULE, LOADLIB, or TXTLIB files. ZAP can be used to modify either fixed- or variable-length MODULE files. It is for use by system support personnel only.

ZAP makes use of control records to control processing. These records can be submitted either from the terminal or from a disk file. Using the VER and REP control records, you can verify and replace records or instructions in a control section (CSECT). Using the DUMP control record, you can dump all or part of a CSECT, an entire member of a LOADLIB or TXTLIB file, or an entire MODULE file.

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 VM/370 Operating Systems in a Virtual Machine for directions on how to set up the virtual machine.

CP Internal Trace Table

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

- External interruptions
- SVC interruptions
- Program interruptions
- Machine check interruptions
- I/O interruptions
- Free storage requests
- Release of free storage
- Entry into scheduler
- Queue drop
- Run user requests
- Start I/O
- Unstack I/O interruptions
- Storing a virtual CSW
- Test I/O
- Halt Device
- Unstack IOBLOK or TRQBLOK
- NCP BTU (Network Control Program Basic Transmission Unit)
- Spinning on a lock (attached processor environment)
- SIGP (X'13')
- Clear Channel instruction

An installation may optionally specify the size of the CP trace table. To do so, use the SYSCOR macro instruction in module DMKSYS. Information on using this macro instruction is in the VM/370 Planning and System Generation Guide.

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

For each 256K bytes (or part thereof) of real storage available at IPL time, one page (4096 bytes) is allocated to the CP trace table. Each entry in the CP trace table is 16 bytes long. There are trace table entries for each type of event recorded. The first byte of each

trace table entry, the identification code, identifies the type of event being recorded. Figure 8 describes the format of each type of trace table entry.

Some trace table entries are generated by ECPS:VM. The first bit of these entries is set to 1 to indicate the entry was generated by the hardware assist. For example, a trace table entry of type X'86' (FREE) is the same as an entry of type X'06'. The only difference is that the first entry was generated by the hardware assist.

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

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

The CP internal trace table is initialized during IPL. If you do not wish to record events in the trace table, issue the MONITOR STOP command to suppress recording. 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.

your virtual machine cannot continue, it terminates and, in some cases, attempts to issue a dump. In the VM/370 environment, the problem program dump always goes to the virtual printer. Depending on installation operating procedures, the virtual machine operating system dump may also go to the virtual printer. A CLOSE must be issued to the virtual printer to have either dump print on the real printer.

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

When the CP abend dump is sent to a disk, use the CMS VMFDUMP command to print the dump on the real printer.

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 131 characters long, unblocked, and contain carriage control characters.

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

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

FILEDEF ddname2 {TAP2} (DEN 1600 RECFM U LRECL 132)
                {TAP1}
```

Then use the MOVEFILE command to print the tape:

```
MOVEFILE ddname2 ddname1
```

An extended form of the VMFDUMP command may be used via the facilities of IPCS (Interactive Problem Control System) by Field Engineering Program Systems Representatives, and by 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 upon 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
- Processor Timer
- Storage
- If in attached processor mode, formats and prints both PSAs' storage

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

If the CP SET DUMP command directed the dump to tape or the printer, the printed format of the dump is the same as with VMFDUMP, except that the control blocks are not formatted and printed. If the system was an attached processor, all of the registers, etc., are printed for the abending processor. Also, each PSA is printed before printing main storage.

When the Control Program can no longer continue and abnormally terminates, you must first determine the condition that caused the abend, and then find the cause of that condition. You should know the structure and function of the Control Program. "Part 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 Block Logic for a description of the CP control blocks. Figure 9 shows the CP control block relationships. Also, you will need the current load map for CP to be able to identify the modules from their locations.

REASON FOR THE ABEND

Determine the immediate reason for the abend. You need to examine several fields in the PSA (Prefix Storage Area), to find the reason for the abend. In a uniprocessor system, the PSA is in locations 0 to 4095. In an attached processor system, each processor has its own PSA.

1. Examine the program old PSW and program interrupt code to find whether or not a program check occurred in CP. The program old PSW (PROPSW) is located at X'28' and the program interrupt code (INTPR) is at X'8E'. If a program check has occurred in supervisor mode, use the CP system load map to identify the module. If you cannot find the module using the load map, refer to "Identifying a Pageable Module." Figure 47 in "Appendix A: System/370 Information" describes the format of an Extended Control PSW.

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

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

   Use the CP system load map to identify the module issuing the SVC 0. If you cannot find the module using the CP system load map, refer to "Identifying a Pageable Module". Figure 47 in Appendix A describes the format of an Extended Control PSW.

3. Examine the old PSW at X'08'. If an abnormal termination occurs because the operator pressed the system restart button, the old PSW at location X'08' points to the instruction that was executing when CP recognized the abnormal termination. Figure 47 in Appendix A describes the format of an Extended Control PSW.

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

COLLECT INFORMATION

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

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
X'08'	CP is running in supervisor state

2. Current User

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

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

The TRACSTRT field (location X'0C') contains the address of the start of the trace table. The TRACEND field (location X'10') contains the address of the byte following the end of the trace table. The address of the next available trace table entry is found in the TRACCURR field (location X'14'). To find the last recorded trace table entry, subtract X'10' from the value at location X'14'. The result is the address of the last recorded entry. Figure 8, earlier in this section, describes the format of each type of trace table entry.

Note: If the system was in attached processor mode, the trace table pointers are in absolute page zero.

REGISTER USAGE

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

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

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

The following general registers usually contain the same information.

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

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

SAVE AREA CONVENTIONS

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

Save areas used by attached processor support are DUMPSAVE, SIGSAVE, LOKSAVE, MFASAVE, SWTHSAVE, LOCKSAVE, and SVCREGS. These save areas are all in the PSA. All except LOCKSAVE and and SVCREGS are 16 words in size.

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

1. SAVEAREA

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

4. DUMPSAVE

   All the general registers at the time of the error are saved in DUMPSAVE (displacement X'500') before DMKDMP is called. They are saved by DMKPSA after a restart, by DMKSVC after an SVC 0, and by DMKPRG. The registers are stored in DUMPSAVE in the order GR0 through GR15. GR12 usually contains the base register for the module executing at the time of the error.

5. SIGSAVE

   SIGSAVE (displacement X'540') is used as a save/work area by DMKEXT, an attached processor-only module that handles all signaling requests. When a signal request is issued, DMKEXTSP is called. On entry, DMKEXTSP stores GR12 through GR15, and GR0 through GR6. GR7 through GR11 are not saved. The remainder of SIGSAVE is used as a work area. GR14 contains the caller's return address.

6. LOKSAVE

   All the general registers are stored in LOKSAVE (displacement X'580') before DMKLOK executes. DMKLOK is an attached processor-only module that manipulates certain attached processor-only locks. The registers are stored in the order GR0 through GR15. GR14 contains the caller's return address.

7. MFASAVE

   All the general registers are stored in MFASAVE (displacement X'5C0') before DMKMCTMA executes. DMKMCTMA is the entry into DMKMCT, an attached processor-only module, that handles malfunction alert interrupts. The registers are stored space in the order GR0 through GR15. GR14 and GR15 contain the caller's return address.

8. SWTHSAVE

   All the general registers are stored in SWTHSAVE (displacement X'600') by DMKSTK and DMKVMASW. DMKVMASW is an entry that is used only in an attached processor system to switch a user's page table pointers. The registers are stored in the order GR0 through GR15. GR14 contains the caller's return address. All entries to DMKSTK store registers GR0 through GR15 in SWTHSAVE.

9. LOCKSAVE

   LOCKSAVE (displacement X'640') is a four-word save area used by the LOCK macro to save GR14, GR15, GR0, and GR1 if the SAVE option of the LOCK macro is specified.

10. SVCREGS

    SVCREGS (displacement X'650') is a four-word save area used to save GR12 through GR15 at the time of an SVC interrupt.

VIRTUAL AND REAL CONTROL BLOCK STATUS

Examine the virtual and real control blocks for more information on the status of the CP system. Figure 9 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:

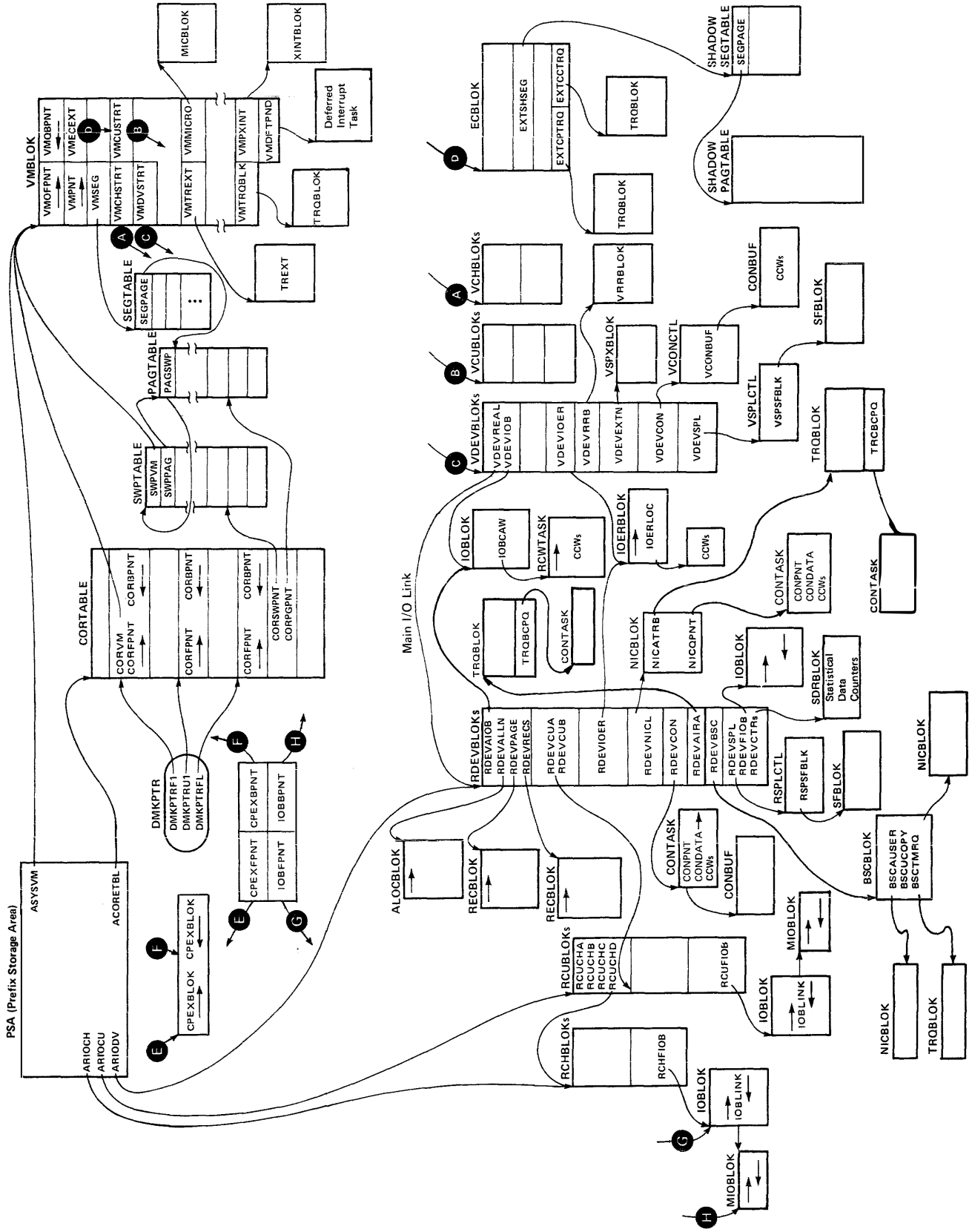


Figure 9. CP Control Block Relationships

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
X'02'	Virtual machine in eligible list
X'01'	Reflect an external interrupt to a virtual machine

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'80'	Deferred task waiting for system lock (attached processor mode)
X'40'	Virtual PER (Program Event Recording) interrupt pending
X'20'	Virtual program interrupt deferred
X'10'	Virtual SVC interrupt deferred
X'08'	Virtual pseudo page fault pending
X'02'	Virtual I/O interrupt pending
X'01'	Virtual external interrupt pending

b. VMFSTAT (displacement X'68') contains the virtual machine features.

Value of VMFSTAT	Comments
X'80'	Virtual block multiplexer channles
X'40'	Autopoll handshake option in use
X'20'	User requested virtual timer request

Value of VMMLVL2	Comments
X'80'	Receiving all informational messages

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

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
X'04'	Virtual control unit active

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'08'	Console -- delay spooling
X'04'	Virtual device is being attached
X'02'	RESERVE/RELEASE are valid CCW operation codes
X'02'	Present attention with a single interrupt
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 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.

10. The value of the VDEVFLG2 field (displacement X'38') describes the Reserve/Release flags.

Value of VDEVFLG2	Comments
X'80'	Process virtual Reserve/Release commands
X'40'	Minidisk reserved by VDEVUSER
X'20'	VDEVBLOK to get device end when minidisk released
X'10'	Virtual I/O waiting for release on minidisk

11. For Reserve/Release minidisks, VDEVRRB (displacement X'3C') contains the address of the VRRBLOK.

RCHBLOK

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

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

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

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'08'	RCUCHA to RCHBLOK path not available
X'04'	RCUCHB to RCHBLOK path not available
X'02'	RCUCHC to RCHBLOK path not available
X'01'	RCUCHD to RCHBLOK path not available

3. RCUCHA (displacement X'10') points to the Primary RCHBLOK.

4. RCUCHB (displacement X'14') points to the first alternate RCHBLOK.

5. RCUCHC (displacement X'18') points to the second alternate RCHBLOK.

6. RCUCHD (displacement X'1C') points to the third alternate RCHBLOK.

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

Value of RCUTYPE	Comments
X'80'	This control unit can attach to only one subchannel
X'40'	Subordinate control unit
X'01'	TCU is a 2701
X'02'	TCU is a 2702
X'03'	TCU is a 2703

8. The RCUFIOB field (displacement X'08') points to the first IOBLCK 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'02'	Graf-IOBLOK pending; queue requests
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
X'10'	Logdrop/loghold indicated

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

Value of RDEVFLAG	Comments
X'02'	Console -- device is to be disabled
X'01'	Console -- 370X 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 -- 370X 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 10 for the list of possible device type class and device type values.

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

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

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

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

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

10. For real 370X Communications Controllers, several pointer fields are defined. The RDEVEPDV field (displacement X'1C') points to the start of the free RDEVBLOK list for EP lines. The RDEVNICL field (displacement X'38') points to the network control list and the RDEVCKPT field (displacement X'3C') points to the CKPBLOK for re-enable. Also, the RDEVMAX field (displacement X'2E') is the highest valid NCP resource name and the RDEVNCP field (displacement X'30') is the reference name of the active 3705 NCP.

11. For terminal devices, additional flags are defined. The value of the RDEVTFLG field (displacement X'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'	ASCII -- 8 level keyboard
X'0C'	APL correspondence keyboard
X'08'	APL PTTC/EBCD keyboard
X'04'	Correspondence keyboard
X'00'	PTTC/EBCD keyboard

DEVICE CLASS CODES

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

DEVICE TYPE CODES

- For Terminal Device Class

Code	Device Type
X'80'	Binary Synchronous Line for Remote
X'40'	2700 Binary Synchronous Line
X'40'	2955 Communication Line
X'20'	Telegraph Terminal Control Type II
X'20'	Teletype Terminal
X'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'01'	Dial Feature
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 3138 System Console
X'04'	IBM 3148 System Console
X'04'	IBM 3158 System Console
X'04'	IBM 3275 Display Station
X'04'	IBM 3276 Display Station
X'04'	IBM 3277 Display Station
X'01'	IBM 3278 Display Station
X'02'	IBM 3284 Printer
X'02'	IBM 3286 Printer
X'02'	IBM 3287 Printer
X'02'	IBM 3288 Printer
X'02'	IBM 3289 Printer

Figure 10. 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'43'	IBM 3203 Printer
X'44'	IBM 1443 Printer
X'45'	IBM 3800 Printing Subsystem
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 10. 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'	370X Programmable Communications Controller
X'20'	3851 Mass Storage Controller
X'04'	SRF (7443) device
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'	Reserve/Release are valid CCW operation codes
X'01'	3330V Virtual MSS volume

- For special devices

Code	Feature
X'10'	Type I channel adapter for 370X
X'20'	Type II channel adapter for 370X

Figure 10. CP Device Classes, Types, Models, and Features (Part 3 of 3)

IDENTIFYING AND LOCATING A PAGEABLE MODULE

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

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

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

VMDUMP RECORDS: FORMAT AND CONTENT

When a user issues the VMDUMP command, CP dumps virtual storage of the user's virtual machine. CP stores this dump on the reader spool file of a virtual machine that the user specified as an operand on the VMDUMP command.

CP writes the storage dump to the spool file as a series of logical records. Each spool file record and each logical dump record is 4096-bytes long. However, because each spool file record contains a header, one logical dump record does not fit into one spool file record. For this reason, CP splits a logical dump record into two parts. CP writes one part to one spool file record and the other part to an adjacent spool file record. The size of each part varies depending upon the amount of space remaining in the spool file record that CP is currently using. Thus, each logical dump record spans two spool file records. Figure 10.1 shows the format of spool file records, the format of logical dump records, and how logical dump records span spool file records.

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

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

hexadecimal offset	length	content
0	4 bytes	the DASD location (DCHR) of the next page buffer
4	4 bytes	the DASD location (DCHR) of the previous page buffer
8	4 bytes	binary zeros
C	4 bytes	the number of data records in the buffer

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

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

CP records the storage dump sequentially starting with the lowest address dumped and ending with the highest address dumped. CP records each byte as an untranslated 8-bit binary value.

For a description of the format and contents of DMPINREC, see VM/370 Data Areas and Control Block Logic. For a description of DMPKYREC1 and DMPKYREC2, see DMPKYREC also in the Data Areas and Control Block Logic publication.

Locating Logical Dump Records

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

$$loc = \frac{240+16n+4096n}{4096}$$

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

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

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

$$loc = \frac{240+(16x3)+(4096x3)}{4096}$$

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

quotient = 3

remainder = 288

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

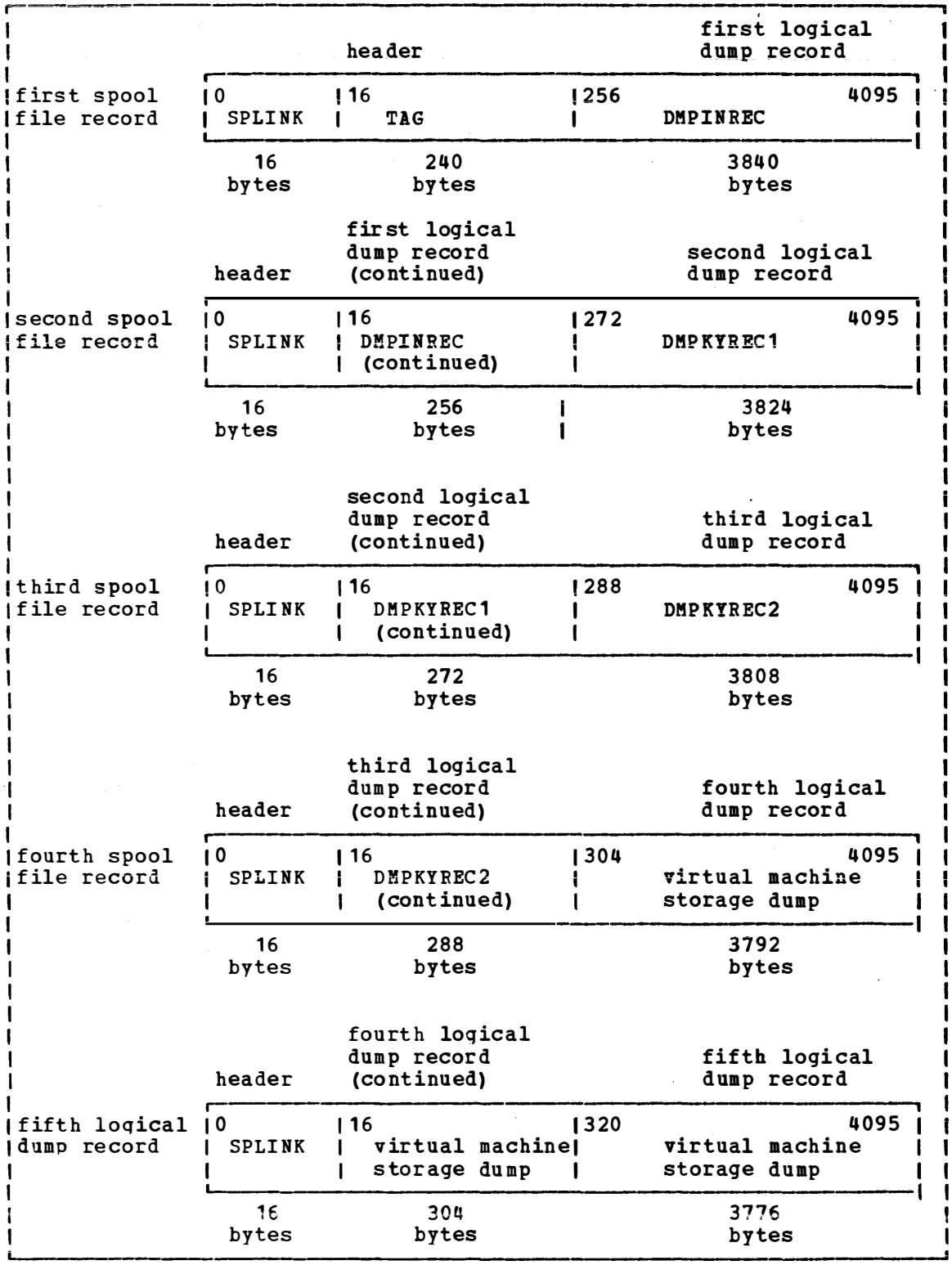


Figure 10.1. VMDUMP Record Format

Debugging with CMS

This section describes the debug tools that CMS provides. These tools can be used to help you debug CMS or a problem program. In addition, a CMS user can use the CP commands to debug. Information that is often useful in debugging is also included. The following topics are discussed in this section:

- CMS debugging commands
- Load maps
- Reading CMS dumps
- Control block summary

CMS Debugging Commands

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

The debug environment is entered whenever:

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

CMS will not accept other commands while in the debug environment. However, while in the debug environment, subcommands of the DEBUG command can be used to:

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

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

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

DEBUG

The DEBUG command provides support for debugging programs at a terminal. The virtual machine operator can stop the program at a specified location in order to examine and alter virtual storage, registers, and various control words. Once CMS is in the debug environment, the virtual machine operator can issue the various DEBUG subcommands. However, in the debug environment, all of the other CMS commands are considered invalid.

Any DEBUG subcommand may be entered if CMS is in the debug environment 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 the debug environment.

Whenever a breakpoint is encountered, a program check occurs. The message

```
DMSDBG728I DEBUG ENTERED
BREAKPOINT YY AT XXXXX
```

appears on the terminal. Follow the same procedure to enter subcommands and resume processing as with a regular program check.

An external interrupt, which occurs when the CP EXTERNAL command is issued, causes CMS to enter the debug environment. The message

```
DMSDBG728I DEBUG ENTERED
EXTERNAL INTERRUPT
```

appears on the console. Any of the DEBUG subcommands may be issued. To exit from the debug environment after an external interrupt, use GO.

While CMS is in the debug environment, the control words and low storage locations contain the debug program values. The debug program saves the control words and low storage contents (X'00' through X'100') of the interrupted routine at location X'C0'.

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

```
LISTFILE * NUCMAP *
```

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

```
PRINT filename NUCMAP filemode
```

Figure 11 shows a sample CMS load map. Notice that the DEBUG work area (DBGSECT) and DMSINM module have been located.

```
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 11. Sample CMS Load Map

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 two ways to get a load map.

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

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

```
| MODmap    | filename                                                    |
```

where:

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

Reading CMS Abend Dumps

| If an abend dump is desired when CMS abnormally terminates, the terminal | operator must enter the DEBUG command and then the DUMP subcommand. The | dump formats and prints:

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

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

When CMS can no longer continue, it abnormally terminates. To debug CMS, first determine the condition that caused the abend and then find why the condition occurred. In order to find the cause of a CMS problem, you must be familiar with the structure and functions of CMS. Refer to "Part 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 Block Logic for details on CMS control blocks. Figure 12 shows the CMS control block relationships. 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. The VM/370 System Messages manual 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 47 in "Appendix A: System/370 Information" for a description of the PSW.

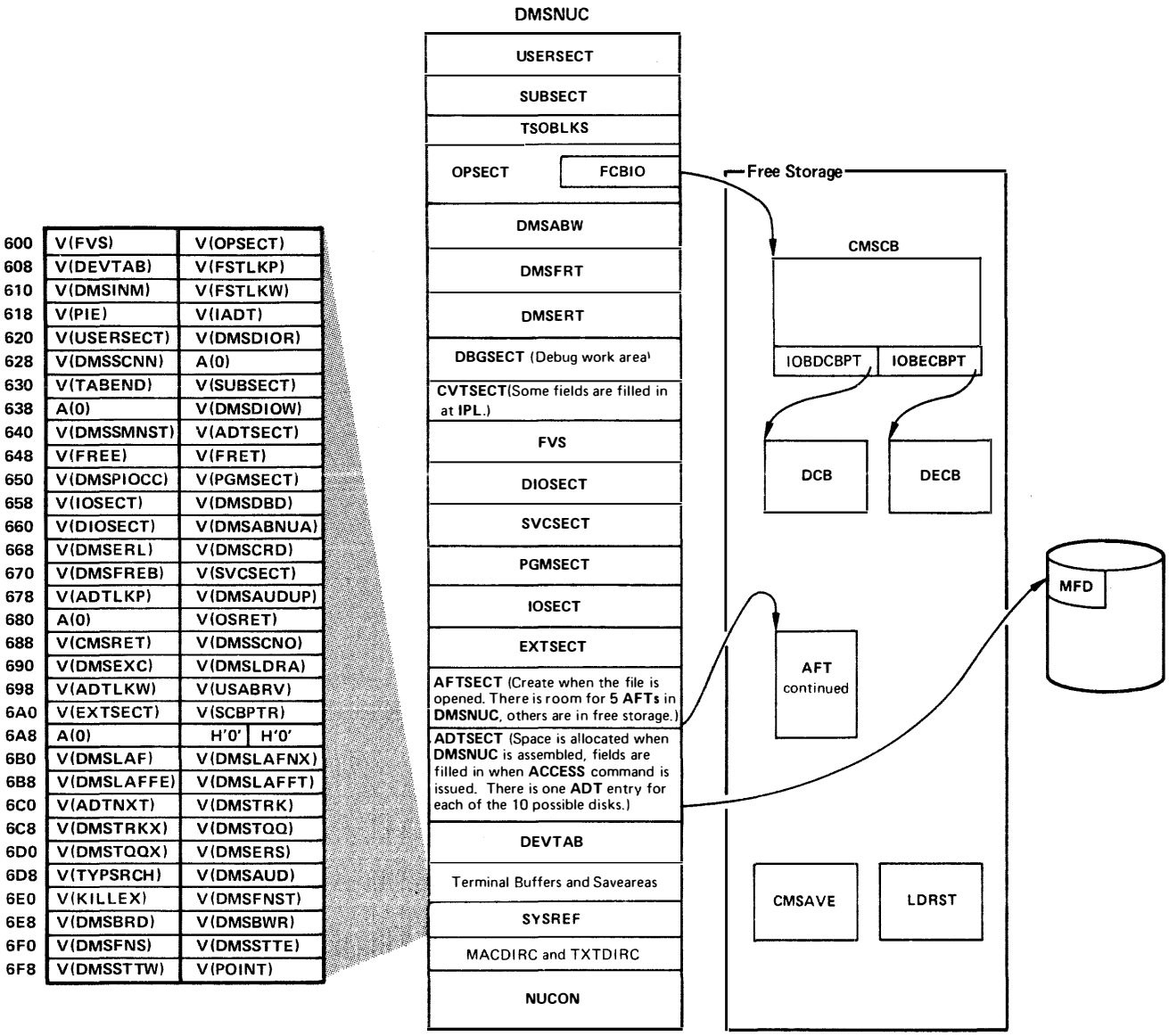


Figure 12. CMS Control Blocks

COLLECT INFORMATION

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

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

- Pointer to CMSCB

  The pointer to the CMSCB is in the FCBTAB field located at X'5C0'. CMSCB contains the simulated OS control blocks. These simulated OS control blocks are in free storage. The CMSCB contains a PLIST for CMS I/O functions, a simulated Job File Control Block (JFCB), a simulated Data Event Block (DEB), and the first in a chain of I/O Blocks (IOBs).

- The Last Command

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

- External Interrupt Work Area

  EXTSECT (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 reentry 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 37 in "Part 3. 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 Processor Resources
- Interruption Handling
- Functional Information
- Performance Guidelines
- Virtual Machine Assist Feature
- VM/370 Extended Control-Program Support
- Performance Observation and Analysis
- Accounting Information
- Generating Named Systems and Saving Systems
- The Virtual Machine Communication Facility
- VM/VS Handshaking
- 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 processor
- Virtual I/O devices

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

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

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

VM/370 performs some functions differently when running in attached processor mode. For a description of the additional processing performed when in attached processor mode, see VM/370 System Logic and Problem Determination Guide.

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

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

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

VIRTUAL MACHINE STORAGE MANAGEMENT

The normal and maximum storage sizes of a virtual machine are defined as part of the virtual machine configuration in the VM/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 relccation 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 operaticns 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 and the page and segment tables created for the virtual machine at logon time to build shadow page tables for the virtual machine. These shadow tables map the virtual storage created by the virtual machine operating system to the storage of the real computing system. The tables created by the virtual machine operating system may describe any page and segment size permissible in the IBM System/370.

Storage Protection

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

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

To use fetch protection, a virtual machine must execute the Set Storage Key (SSK) instruction referring to the data areas to be protected, with the fetch protect bit set on in the key. VM/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. In order to protect the CMS nucleus in the shared segment, user programs and disk-resident CMS commands run with a different key than the nucleus code.

Storage and Processor Utilization

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

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

The virtual=real option improves performance in the selected virtual machine since it removes the need for CP paging operations for the selected virtual machine. The virtual=real option is necessary whenever programs that contain 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.

Virtual Reserve/Release support can be used to further enhance device integrity for data on shared minidisks. Reserve/Release operation codes are simulated on a virtual basis for minidisks, including full-extent minidisks. For details on Reserve/Release support, refer to the VM/370 System Logic and Problem Determination Guide.

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.

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

In 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.

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

Dedicated Channels

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

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

SPOOLING FUNCTIONS

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

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

Spooling operations cease if the direct access storage space assigned to spooling is 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 command

```
SPOOL CONSOLE START
```

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

```
SPOOL CONSOLE STOP
```

SPOOL FILE RECOVERY

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

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

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

Warm Start

After a system failure, the warm start procedure copies spool file, accounting, and system message data to 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 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 (class A), system resource operator (class B), system programmer (class C), spooling operator (class D), system analyst (class E), service representative (class F), and general user (class G). Commands in the system analyst class may be used to inspect real storage locations, but may not be used to make modifications to real storage. Commands in the operator class provide real resource control capabilities. System operator commands include all commands related to virtual machine performance options, such as assigning a set of reserved page frames to a selected virtual machine. For descriptions of all the CP commands, see the VM/370 CP Command Reference for General Users and the VM/370 Operator's Guide.

Program States

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

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

Using Processor Resources

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

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

Virtual machines are dispatched from one of two queues, called Queue 1 and Queue 2. In order 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:

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

Queue 1

Virtual machines in Queue 1 (Q1) are considered conversational or interactive users, and enter this queue when an interrupt from a terminal is reflected to the virtual machine. Users are considered for dispatching from this queue on a first-in-first-out (FIFO) basis. When a virtual machine uses more than a certain amount of processor time without entering a virtual wait state, that user is placed in Queue 2.

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

Usually, 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, then 75 percent of the working set for that virtual machine is calculated. If the pages required for 75 percent of the working set are available, the virtual machine is placed on Q2. Otherwise, the virtual machine remains on the eligible list until there are no other users on Q1 or Q2.

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 processor time used while in the queue to elapsed time in the queue. Infrequent processor users are placed at the top of the list and are followed by more frequent processor users. When a nonexecutable user becomes executable, he is placed on the queue based on his dispatching priority.

When a virtual machine completes its time slice of processor 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:

- The virtual machine is on the "eligible list," waiting to be put on Q2

-- or --

- 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 that 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 processor 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 processor 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.

Interruption Handling

I/O Interrupts

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

Program Interrupt

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

Machine Check Interrupt

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

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

- System termination (CP disabled wait state)
- Attached processor disabled (system continues in uniprocessor mode)
- Selective virtual user termination
- Selective virtual machine reset
- Refreshing of damaged information with no effect on system configuration
- Refreshing of damaged information with the defective storage page removed from further system use
- Error recording only for certain soft machine checks

The system operator is informed of all actions taken by the RMS routines. When a machine check occurs during VM/370 startup (before the system is sufficiently initialized to permit RMS to operate successfully), the processor goes into a disabled wait state and places a completion code of X'00B' in the leftmost bytes of the current PSW.

SVC Interrupt

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

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

External Interrupt

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

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

Synchronous Interrupts in an Attached Processor System

Generally, when synchronous interrupts (such as program and SVC interrupts) occur in an attached processor system, the first-level interrupt handler (FLIH) attempts to gain the system lock before proceeding. If it is already in use, the interrupt status is stacked and deferred. The interrupted processor then attempts to run a user.

Real I/O Interrupts

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

Performance Guidelines

General Information

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

- The System/370 model used.
- The total number of virtual machines executing.
- The type of work being done by each virtual machine.
- The speed, capacity, and number of the paging devices.
- The amount of real storage available.
- The degree of channel and control unit contention, as well as arm contention, affecting the paging device.
- The type and number of VM/370 performance options in use by one or more virtual machines.
- The degree of MSS 3330 volume use.

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. If the real device is a 3330V, append an MSS cylinder fault prefix to the CCW prefix to prevent the channel from doing channel command retry.
7. Schedule the I/O request.
8. Present the SIO condition code to the virtual machine.
9. Recognize an MSS cylinder fault, queue the I/O request, and reschedule the request when the subsequent interruption is received (indicating staging is complete).
10. Intercept, retranslate, and present the channel end and device end interrupts to the appropriate virtual machine, where they must then be processed by the virtual machine operating system.

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

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

- Use of large blocking factors (up to 4096 bytes) for user data sets to reduce the total number of SIOs needed.
- Use of preallocated data sets.
- Use of virtual machine operating system options (such as chained scheduling in OS) that reduce the number of SIO instructions.

- Substitution of a faster resource (virtual storage) for I/O operations, by building small temporary data sets in virtual storage rather than using an I/O device.

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

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

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

2. The virtual machine assist feature reduces the real supervisor state time used by VM/370. For a detailed description of the feature, see "Virtual Machine Assist Feature" later in this section. For a list of processors on which the feature is available, see the VM/370 Planning and System Generation Guide.

3. VM/370 Extended Control-Program Support (ECPS) further reduces the real supervisor state time used by VM/370. For a detailed description of ECPS, see "VM/370 Extended Control-Program Support (ECPS)" later in this section. For a list of processors on which ECPS is available, see the VM/370 Planning and System Generation Guide.

Assignment and use of these options is discussed in "VM/370 Performance Options."

Paging Considerations

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

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

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

Virtual machines that reduce their paging activity by controlling their use of addressable space improve resource management for that virtual machine, the VM/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.

Additional dynamic paging storage may be gained by controlling free storage allocation. The amount of free storage allocated at VM/370 initialization time can be controlled by the installation. When the System is being generated, the FREE operand of the SYSCOR macro statement may be used to specify the number of free storage pages to be allocated at system load time.

If, at IPL time, the amount of storage that these pages represent is greater than 25 percent of the VM/370 storage size (not including the V=R area, if any), a default number of pages is used. The default value is 3 pages for the first 256K bytes of storage plus 1 page for each additional 64K bytes (not including the V=R size, if any).

The SYSCOR macro definition can be found in VM/370 Planning and System Generation Guide.

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

LOCKED PAGES OPTION

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

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

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

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

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

running the teleprocessing program on a real machine. Further response time is variable, depending upon the number of paging operations that must occur.

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

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

Note: In attached processor mode, no shared pages are locked. If the system operator attempts to lock a shared page or an address range containing one or more shared pages, he will receive the message

DMKCPV165I PAGE (hexloc) NOT LOCKED, SHARED PAGE

for each of the shared pages within the range.

RESERVED PAGE FRAMES OPTION

A more flexible approach than locked pages is the reserved page frames option. This option provides a specified virtual machine with an essentially private set of real page frames, the number of frames being designated by the system operator when he issues the CP SET RESERVE command 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 "VM/370 Performance Options."

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

VIRTUAL=REAL OPTION

The VM/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 "VM/370 Performance Options."

Since the entire address space required by the virtual machine is locked, these page frames are not available for use by other virtual machines except when the virtual=real machine is not logged on. This option often increases the paging activity for other virtual machine users, and in some cases for VM/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.

VM/370 Performance Options

VM/370 provides a number of options an installation may use to improve the performance of virtual machines and VM/370. Several options improve the performance of installation specified virtual machines; other options improve the performance of all virtual machines and VM/370. The options, described in the following discussion are:

- Favored execution
- User priority
- Reserved page frames
- Virtual=real
- Affinity
- Virtual machine assist
- Extended Control-Program Support

Specifying a performance option may mean making a performance trade-off; improving the performance of one virtual machine at the expense of VM/370 and other virtual machines. For example, after an operator specifies favored execution for a virtual machine, that virtual machine receives more processor time than other virtual machines. Therefore, before specifying any performance option, identify the option's performance trade-offs and assess their impact on system performance.

FAVORED EXECUTION

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

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

The basic favored execution option means that the virtual machine so designated is 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 processor with many other tasks of the same type, an installation should define the processor allocation to be made. In this case, the favored execution percentage option can be selected 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 processor time if it can use it. To select the favored execution option, specify the FAVORED operand on the class A, B, or F SET command. After the option is invoked, VM/370 provides processor time for the selected virtual machine as follows:

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

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

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

For a description of the SET command, see the VM/370 Operator's Guide.

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 processor 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 so doing, 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). For information about generating a virtual=real system, see the VM/370 Planning and System Generation Guide.

Figure 13 is an example of a real storage layout with the virtual=real option. The V=R area is 128K and real storage is 512K.

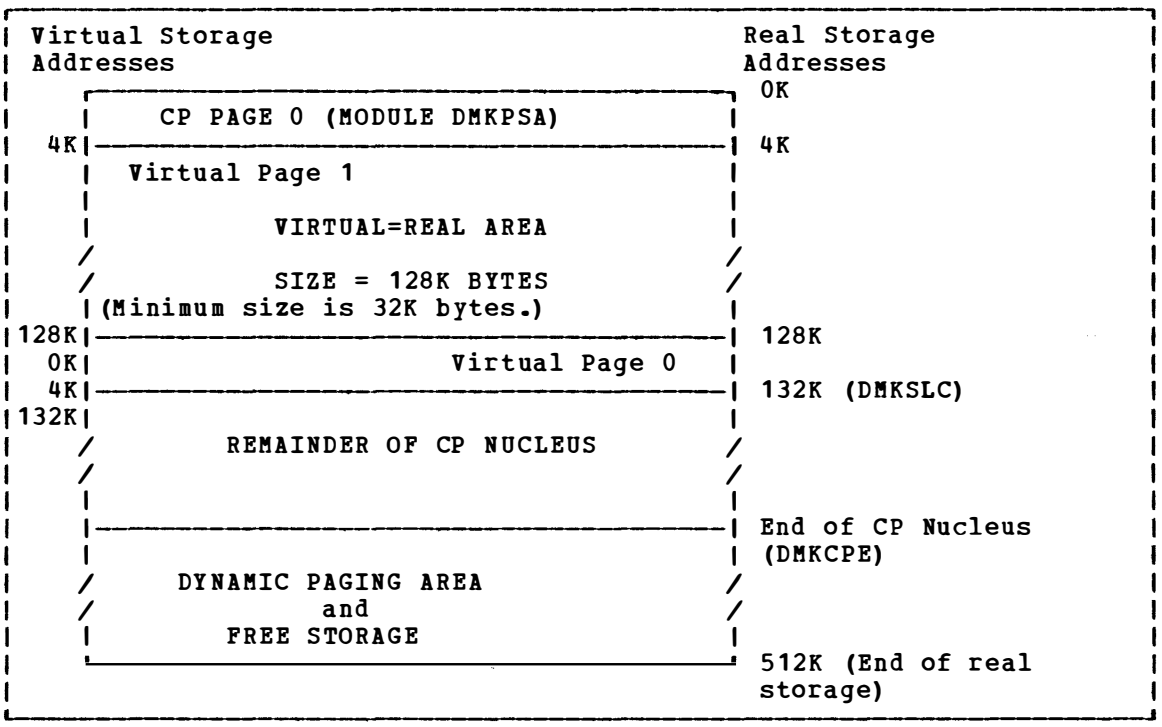


Figure 13. Storage Layout in a Virtual=Real Machine

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

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

2. The storage reserved for the virtual=real machine can only be used by a virtual machine with that option specified in the VM/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.

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

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

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

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

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

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

AFFINITY

This option allows virtual machines that operate on attached processor systems to select the processor of their choice for program execution. To select the affinity option, use the directory OPTION statement, or specify the AFFINITY operand on the class A, B, F, or G SET command. The directory OPTION statement is described in the VM/370 Planning and System Generation Guide. The class A, B, and F SET commands are described in the VM/370 Operator's Guide, and the class G SET command is described in the VM/370 Command Reference for General Users.

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

In attached processor systems, all real I/O operations and associated interrupts are handled by the main processor. Virtual I/O initiated on the attached processor that is mapped to real devices must transfer control to the main processor for real I/O execution. Therefore, benefits may be realized in a virtual machine "mix" by relegating those virtual machines that have a high I/O-to-compute ratio to the main processor, and those virtual machines that have a high compute-to-I/O ratio to the attached processor. Such decisions should be carefully weighed as every virtual machine is in contention with other virtual machines for resources of the system.

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

VIRTUAL MACHINE ASSIST FEATURE

The virtual machine assist feature is a processor hardware feature that 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 that VM/370 must handle. When the virtual machine assist feature is used, many of these interrupts are intercepted and handled by the processor. Consequently, VM/370 performance is improved.

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

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

Although the assist feature was designed to improve the performance of VM/370, virtual machines may see a performance improvement because more resources are available for virtual machine users. For a list of processors on which the Virtual Machine Assist Feature is available, see the VM/370 Planning and System Generation Guide.

USING THE VIRTUAL MACHINE ASSIST FEATURE

Whenever you IPL VM/370 on a processor 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 for all users. If you do not know whether or not the virtual machine assist feature is available to VM/370, use the class A and E QUERY command. For a complete description of the Class A and E QUERY and SET commands, see the VM/370 Operator's Guide.

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 unless you are running a second-level VM/370 system in 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. You can use the class G QUERY SET command line to find whether you have full, partial, or none of the assist feature available. For a complete description of the Class G QUERY and SET commands, see the VM/370 CP Command Reference for General Users.

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 that:

- Has set an instruction address stop
- Is tracing 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 machine assist feature is not available to a second-level virtual machine, that is, a virtual machine that is running in a virtual machine.

VM/370 Extended Control-Program Support (ECPS)

VM/370 Extended Control-Program Support (ECPS) extends, for specific privileged instructions, the hardware assistance that the virtual machine assist feature provides. ECPS also provides hardware assistance

for frequently used VM/370 functions. The use of ECPS improves VM/370 performance beyond the performance gains that the virtual machine assist feature provides.

ECPS consists of three functions:

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

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

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

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

ECPS does not support the same functions and instructions on all processors. Figure 13.1 lists the processors on which ECPS is available, and identifies, by processor, the functions and instructions ECPS supports.

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

Figure 13.1. Functions and Instructions that ECPS Supports

VIRTUAL INTERVAL TIMER ASSIST

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

USING THE VM/370 EXTENDED CONTROL-PROGRAM SUPPORT

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

At the VM/370 system level, ECPS is automatically enabled when the system is loaded. The class A command:

```
set cpassist off
```

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

```
set sassist off
```

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

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

```
set assist off
```

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

```
set assist notmr
```

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

```
set sassist on
```

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

```
set assist on tmr
```

Restricted Use of ECPS

The restrictions on the use of ECPS are the same as those described for the virtual machine assist feature with one addition. Whenever a

virtual machine traces external interrupts, the virtual interval timer assist is automatically disabled. When external interrupt tracing is completed, virtual interval timer assist is reenabled.

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 throughput, particularly for those systems or programs that are designed to use the block multiplexer channels.

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

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

ALTERNATE PATH SUPPORT

Through the use of the Two-Channel Switch and Two-Channel Switch Additional Features, alternate path support for DASD or tape provides for up to four channels on one control unit to be attached to VM/370. In addition, one device may be attached to two logical control units, providing support for the String Switch feature. This allows the control program up to eight paths to a given device when the maximum number of alternate channels and alternate control units are specified.

When an I/O request is received for a device which has alternate paths defined, the VM/370 IOS supervisor searches for an available path beginning with the primary path to the device. If the primary path is unavailable, the search continues with the first alternate path. Successive alternate paths are examined if required until an available path is found.

In case where no available path to the device exists, alternate path I/O scheduling is implemented in such a way that the request is queued off multiple busy/scheduled paths and the first path to become available is the path that the I/O request is started on.

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

Note: There is no alternate path scheduling after the SIO if a control unit busy, channel busy, or not operational condition is encountered. The I/O request will be queued in the busy conditions on the busy control unit or channel block to wait for an interrupt which will cause the request to be restarted. The not operational condition is presented to the second level interrupt handlers as a fatal condition.

Performance Observation and Analysis

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

THE INDICATE COMMAND

The INDICATE command allows the general user and the system analyst to display 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 processor 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. Because of spooling considerations, the INDICATE command may produce unexpected results if entered while a program is issuing I/O requests.

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 processor 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:

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

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

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

$$\text{New smoothed value} = \frac{(15 \times \text{old smoothed value} + \text{last interval value})}{16}$$

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:

```
┌──────────────────────────────────────────────────────────────────────┐
│              │  ┌    ┐                                                │
│ INDicate     │  │LOAD│                                                │
│              │  │USER│                                                │
│              │  └    ┘                                                │
└──────────────────────────────────────────────────────────────────────┘
```

where:

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

CPU-nnn% APU-nnn% Q1-nn Q2-nn STORAGE-nnn% RATIO-nnn

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

The APU figure is the percentage of time the attached processor is running.

The contention for the processor 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:

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

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 processor 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 processor 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 processor time for the user.

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

SIO is the total number of nonspooled I/O requests issued by the user.

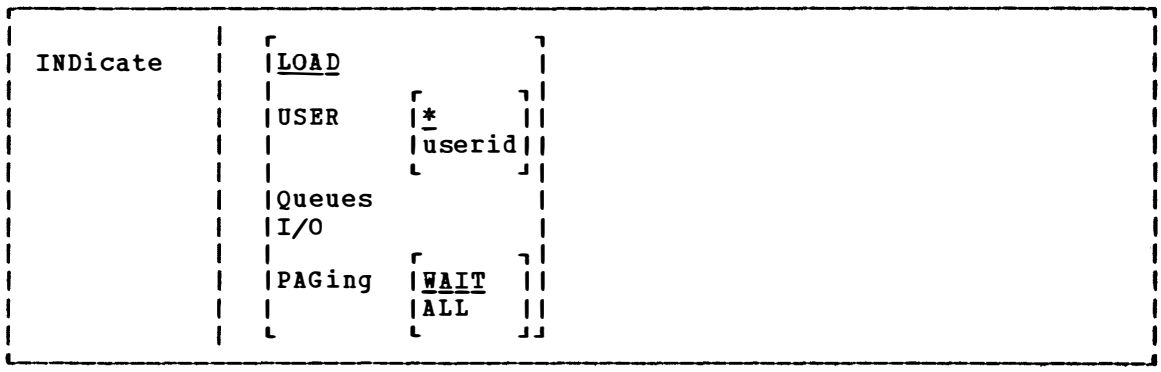
RDR is the total number of virtual cards read.

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 current runuser in uniprocessor mode. In attached processor configurations, the current runuser on the main processor.
- RA in attached processor configurations, the current runuser on the attached processor. (Not used in uniprocessor mode).
- DF in attached processor configurations, the processing of a synchronous (program and SVC) interrupt for this user has been deferred until the system lock is available (not used in uniprocessor mode).
- 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.

Note: The order of the users in the INDICATE QUEUES response is as follows:
1. Q1 and Q2 users in runlist priority order (that is, dispatching priority order).
2. Eligible list E1 users in scheduling priority order.
3. Eligible list E2 users in scheduling priority order.

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 devices. 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:

NO USERS IN QUEUE
is issued for the INDICATE QUEUES option when appropriate.

NO USERS IN I/O WAIT
is issued for the INDICATE I/O option when appropriate.

NO USERS IN PAGEWAIT
is issued for the INDICATE PAGING WAIT option when appropriate.

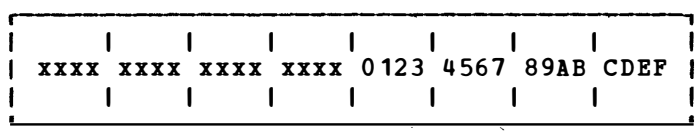
The MONITOR Command

VM/370 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. The format of control register 8 is as follows:



where:

x — indicates unassigned bits.

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

When a MONITOR CALL interruption occurs, the CP program interruption handler (DMKPRG) transfers control to the VM/370 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/370 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 as though it were obtained through a MONITOR CALL instruction. This simplifies the identification of the records.

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

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

---

[1]There is no class name for monitor class 3, but it is reserved.

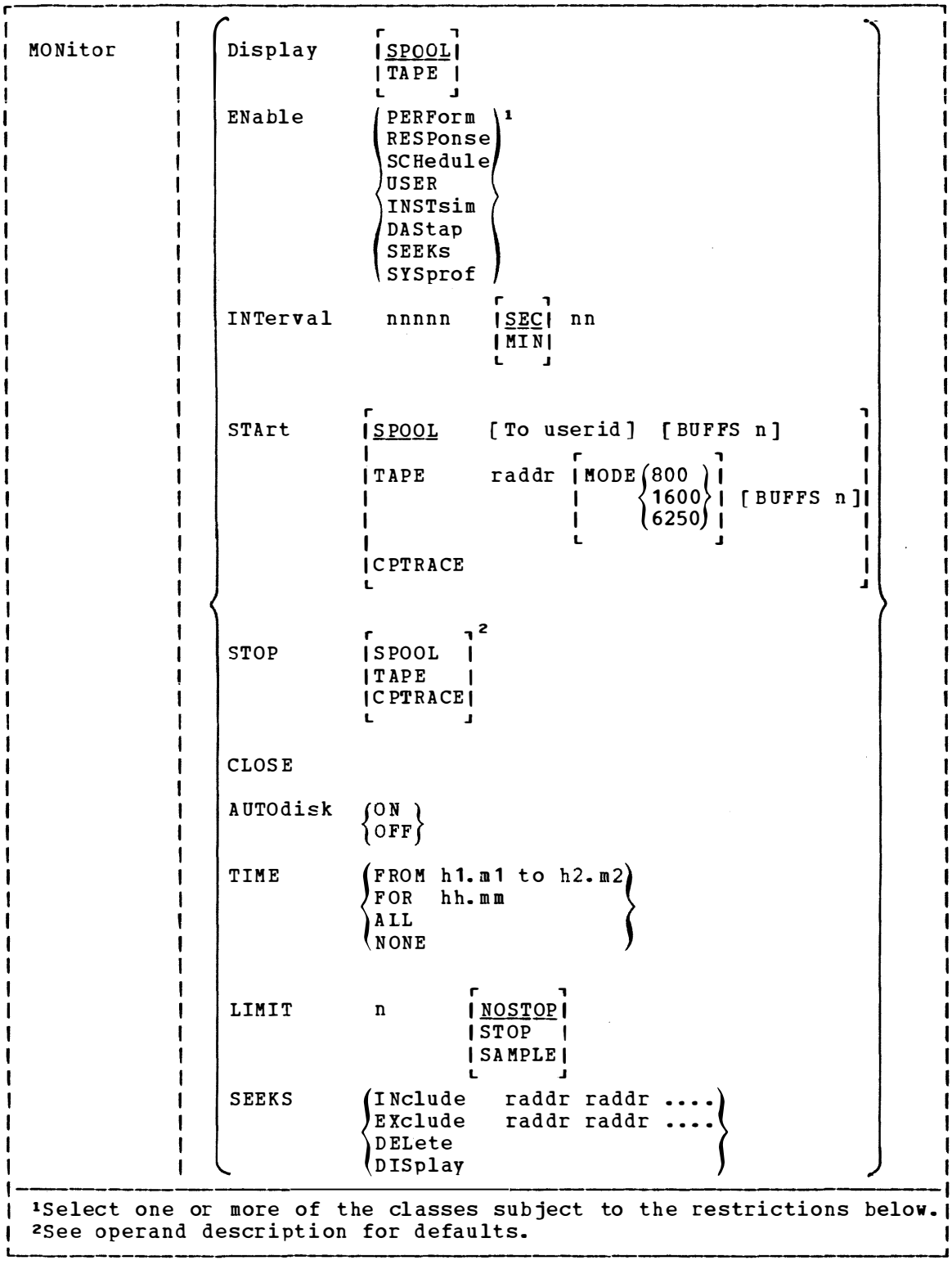
Another function, separate from the VM/370 Monitor, is also handled by the MONITOR command. The MONITOR command can stop and start CP internal trace table data collection, which is not initiated by MONITOR CALLs.

Note: The VM/370 Monitor record format and the contents of the record are shown in "Appendix C. Monitor Tape Format and Content."

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/370 Monitor data collection. Displays the specifications for automatic monitoring defined by the SYSMON macro in DMKSYS. In addition, it displays those specifications for automatic monitoring that are overridden by Monitor commands. It also displays whether the tape, or spool file is the recording medium.
- Starts and stops VM/370 data collection using tape or spool file. It also closes the spool file, if desired.
- Specifies VM/370 monitor classes of data collection enabled, number of buffers used, and time of data collection. It also specifies other options which override the specifications for automatic monitoring on the SYSMON macro contained in DMKSYS.
- Specifies the interval to be used for timer driven data collection.
- Specifies direct access devices that are to be included or excluded from a list of devices. The list defines direct access devices for which CP is to collect data for the SEEKs class.

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



where:

DISPLAY [SPOOL | TAPE | ALL]

displays the status of the applicable VM/370 Monitor variables and the status of the internal trace table. SPOOL is the default operand. Regardless of the SPOOL, TAPE, or ALL operand selected, each class of MONITOR CALL and its current enabled/disabled state is listed on the terminal.

If SPOOL is requested, the automatic monitoring specifications are listed on the terminal, including whether or not automatic monitoring has been requested, its start and stop times, the number of monitor buffers to be used, the userid of the virtual machine to receive the spool file, the spool file record limit and class, and which monitor classes are to be enabled.

If automatic monitoring is already in progress, the spool file number is given together with the number of monitor buffer records already written to it.

If the TAPE option is requested, only the status of monitor classes and the CPTRACE table is indicated.

If ALL is specified, a combination of SPOOL and TAPE responses are shown on the terminal.

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 upon whether data collection is active or inactive when the command is issued. If data collection is active (MONITOR START 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, the mask is saved until the MONITOR START command is issued. If a MONITOR START command is issued without a preceding MONITOR ENABLE, the SYSMON class specifications are used. Any mask stays in effect only until the next MONITOR STOP command.

MONITOR ENABLE Restrictions:

Restrictions exist on issuing the MONITOR ENABLE command while the VM/370 Monitor is collecting and recording data.

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 potential security exposure that 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 file to a 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] nn
| [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 is generated. 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.

nn specifies how frequently the VM/370 Monitor is to sample channel status, control unit status, and device status for the DASTAP class. The nn parameter specifies only seconds, has a minimum value of 1, a maximum value of 99, and a default value of 2. The amount of time specified by nn must be less than the amount of time specified by nnnnn.

The value of nn and the value of nnnnn work together in the following way. The VM/370 Monitor accumulates channel status, control unit status, and device status in a buffer. The value of nn determines the frequency of accumulation; the value of nnnnn determines how frequently the buffer is

written to tape. When the buffer is written to tape, it is written as a class 6 (DASTAP) record under monitor code 02.

The MONITOR interval is reset to the default of 60 seconds whenever any of the following occurs:

- The user issues MONITOR STOP, or the monitor stops automatically
- The system stops the MONITOR because of an unrecoverable I/O error
- The end of tape or spool record limit is reached

```
START SPOOL    [To userid]  [BUFFS n]
      TAPE     raddr  [MODE {800 } [BUFFS n]]
                              {1600}
                              {6250}
      CPTRACE
```

starts VM/370 Monitor data collection to the spool file or tape, or starts the CP internal trace table. If no optional parameter is provided, SPOOL is the default. MONITOR START SPOOL starts the VM/370 Monitor data collection using a spool file for storage.

When data collection is stopped and the spool is closed, the spool file is added to the chain of reader files destined for the virtual reader of the virtual machine defined by "userid." Userid may be an asterisk (*) if the recipient virtual machine is to be the one from which the START command is issued. If the TO USERID option is omitted, the userid specified in the SYSMON macro is used (See the VM/370 Planning and System Generation Guide. The TO USERID option overrides the SYSMON specification and stays in effect until the system is reinitialized or a new command is issued.

The monitor spool file is closed by a MONITOR STOP or MONITOR CLOSE command, or when the record count limit is reached (as specified in the SYSMON macro), or when a system restart or system shutdown occurs.

The filename and filetype is generated internally with the filetype identifying date and time of starting.

The class of spool file is specified in the SYSMON macro and defaults to "M". If no classes of data collection have been specified with an ENABLE command, then those specified with the SYSMON macro are used.

The number of monitor buffers used are as specified in the SYSMON macro, or as requested with the BUFFS option of the START command. The BUFFS option overrides the SYSMON specification only for the duration of the data collection session. Future monitoring sessions subsequently return to the SYSMON specification unless again overridden. If the number of buffers specified in the SYSMON macro has been defaulted, then the defaults described in the MONITOR START TAPE command are adopted.

The START TAPE command starts the data collection by VM/370 Monitor onto a tape mounted on a 9-track tape drive. 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.

Use BUFFS to specify the number of buffers to be used for monitoring, where "n" may be 1 to 10 and specifies the number of 4096-byte buffers to be used. If the option is omitted, VM/370 assigns a default. The value of the default depends upon the model number or type of processor as follows:

Model or Processor	Default
Model 135, 138, or 145	2 buffers
4331 Processor	2 buffers
Model 148, 155, or 158	3 buffers
Model 165 or 168	4 buffers
3031, 3032, 3033, or 4341 processor	4 buffers

It is valid to specify BUFFS 1 only if the PERFORM class of data collection is the only one enabled with the MONITOR ENABLE command. Once monitoring is in progress with just one buffer, it is not possible to issue a MONITOR ENABLE command with other than just the PERFORM class of data collection specified.

Single-buffer operation is useful for basic performance analysis in minimum main storage configurations.

When the VM/370 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 a density mode that the tape cannot handle, the control unit may not return an error condition. In this case, the mode setting is ignored, and the default control unit setting is used.

The START CPTRACE command 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.

```
      ┌       ┐[1]
STOP  |SPOOL  |
      |TAPE   |
      |CPTRACE|
      └       ┘
```

stops VM/370 Monitor data collection to a spool file or tape, or stops the CP internal trace table. If no option is specified and the VM/370 Monitor is active, then data collection is terminated, whether or not a spool file or tape is in use. Internal tracing can only be stopped by specific use of the CPTRACE option. If automatic VM/370 Monitor data collection is active when the MONITOR STOP SPOOL command is issued, monitoring ceases and will not start again (even if the current time is within the bounds of the TIME operand of the SYSMON macro) unless the system abnormally terminates or is shut down and reloaded.

The STOP TAPE command stops data collection by VM/370 Monitor onto 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.

The STOP CPTRACE command 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.

Note: The CPTRACE and TAPE operands of the MONITOR command have completely separate functions. Commands affecting the status of one function have no effect on the other.

CLOSE

may be used when VM/370 Monitor is collecting performance data using a spool file and it is desirable to reduce the data collected thus far. The command closes the current spool file (thereby making it available to the reader of the recipient virtual machine) and causes monitoring to continue uninterrupted with a new spool file.

```
AUTOdisk {ON }
         {OFF}
```

may be used to override the specification for automatic monitoring in the SYSMON macro. Its only use is to affect the automatic startup of monitoring. If automatic monitoring is already active, it may only be stopped manually by a MONITOR STOP command. Note that in general, any attempts to override the definitions of the SYSMON macro with commands are temporary. No monitor checkpointing is attempted, so that an IPL or abnormal termination causes full restoration of the initial automatic monitoring definitions.

---

[1] The default value is the active trace facility that is SPOOL or TAPE.

```
        ┌                    ┐
TIME    |FROM h1.m1 to h2.m2 |
        |FOR  hh.mm          |
        |ALL                 |
        |NONE                |
        └                    ┘
```

specifies that the automatic monitoring start and stop times defined by the SYSMON macro should be temporarily overridden (until the next IPL or the next MONITOR TIME command). The FROM, ALL, and NONE options are equivalent to their counterparts in the TIME operand of the SYSMON macro (see the VM/370 Planning and System Generation Guide. The FOR option is provided specifically to simplify data collection during benchmarking or testing. If AUTO ON is in effect, spool monitoring will start immediately and run for the specified period of time. Note that if automatic monitoring is imminent and the FOR option is specified, the period of monitoring defined by the SYSMON macro will be overridden.

```
        ┌       ┐
LIMIT n |NOSTOP|
        |STOP   |
        |SAMPLE |
        └       ┘
```

specifies that the LIMIT options of the SYSMON macro should be temporarily overridden (until the next system IPL or the next MONITOR LIMIT command). The maximum buffer count in each spool file may be changed with the n parameter (within the 10 to 50000 range of the SYSMON macro). If it is necessary to change whether or not automatic monitoring should continue after the limit has been reached and the spool file closed, then the STOP, NOSTOP options may be specified. If this should be done without changing the limit number, an asterisk (*) may be specified for "n".

When SAMPLE is specified, n defines how frequently the VM/370 Monitor is to close the monitor spool file and send it to the virtual reader of the data reduction virtual machine. To determine when to close the file, the VM/370 Monitor counts the number of data samples it takes for the PERFORM, USER, and DASTAP classes. When this count equals the value of n, the VM/370 Monitor closes the file. After the file is closed, monitoring continues using a new file.

The following example shows how the INTERVAL parameter and the LIMIT parameter work together. If the INTERVAL parameter specifies an interval of 30 seconds and the LIMIT parameter is coded as LIMIT 10 SAMPLE, the VM/370 Monitor closes the spool file every 300 seconds (30x10).

The LIMIT parameter may be specified for automatic monitoring by using the SYSMON macro instruction in module DMKSYS. Instructions for using this macro instruction are in the VM/370 Planning and System Generation Guide.

```
SEEKS  (INclude   raddr raddr ...)
       (Exclude   raddr raddr ...)
       {DELete                   }
       (DISplay                  )
```

allows an installation to establish and maintain a list of real device addresses for DASD devices. The VM/370 Monitor collects status information from the listed devices for the SEEKS class and writes the information to a class 7 monitor record. Appendix C defines the format and content of the class 7 record. The length of the list is limited only by the maximum number of device addresses that fit on one line of the terminal at which the command is entered.

Use the INclude option to create a list or to add entries to the existing list.

Use the EXclude option to exclude non-2305 devices from the list. The VM/370 Monitor collects information for all non-2305 devices not excluded.

Use the DELete option to delete the entire list. Deleting the list frees the storage the list occupied.

Use the DISplay option to display the list. To reduce the performance impact of collecting the status information, keep the list as short as possible. CP retains the list from one monitoring session to another. Therefore, periodically review the list to ensure that it does not contain unnecessary addresses.

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 for details of the MC instruction and the bits in control register 8.

Monitor Class	Keyword	Data Collection Function
0	PERFORM	Samples system resource usage data by accessing system counters of interest to system performance analysts.
1	RESPONSE	Collects data on terminal I/O. Simplifies analyses of command usage, user, and system response times. It can relate user activity to system performance. This class is invalid and no data can be collected for it unless the system programmer changes the LOCAL COPY file and reassembles DMKMCC.
2	SCHEDULE	Collects data about scheduler queue manipulation, monitors flow of work through the system, and indicates the resource allocation strategies of the scheduler.
3	-------	Reserved.
4	USER	Periodically scans the chain of VMBLOKs in the system, and extracts user resource utilization and status data.
5	INSTSIM	Records every virtual machine privileged instruction handled by the control program (CP). Because simulation of privileged instructions is a major source of overhead, this data may lead to methods of improving performance. 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.
6	DASTAP	Periodically samples device I/O activity counts (SIOs), for tape and DASD devices only. It is possible that the number of DASD and tape devices defined in DMKRIO may exceed 291 (the maximum number of MONITOR DASTAP records that fit in a MONITOR buffer). The following algorithm determines which devices are monitored: 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. 2. If the total number of online DASD devices is less than or equal to 291, all online DASD devices are monitored. 3. Otherwise, the first 291 online DASD devices are monitored.

Monitor Class	Keyword	Data Collection Function
7	SEEKS	Collects data for every I/O request to DASD. Reveals channel, control unit, or device contention and arm movement interference problems. Note: When NOTRANS is in effect for a virtual=real machine, no meaningful data is collected. 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. 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. In general, the data collected is the same except that in the first case there will be nonzero 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.
8	SYSPROF	Collects data complementary to the DASTAP and SCHEDULE classes in order to provide a more detailed "profile" of system performance through a closer examination of DASD utilization.

VM/370 MONITOR RESPONSE TO UNUSUAL TAPE CONDITIONS

Suspension

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

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/370 Monitor is stopped and the operator is informed of the action taken.

VM/370 MONITOR CONSIDERATIONS

System Generation

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

Initial Program Load

MONITOR START CPTRACE is active after real system IPL (manual or automatic). The VM/370 Monitor tape data collection is off after IPL. If automatic performance monitoring is specified in the SYSMON macro and IPL occurs within the range of the TIME operand of the SYSMON macro, VM/370 monitor data collection to a spool file is started.

System Shutdown

If the VM/370 Monitor data collection to a spool file is taking place, a system shutdown causes closing of the file and termination of monitoring. If data collection is to tape, a system shutdown implies a MONITOR STOP TAPE command. Normal command processing for the MONITOR STOP TAPE function is performed by the system.

System Failure

If the VM/370 system fails and data collection to a spool file is active, the spool file is closed and preserved, except for the last buffer. If the VM/370 system fails and data collection is active on tape, an attempt is made to write two tape marks, rewind, and unload the tape. If the tape drive fails to rewind and unload, be sure to write a tape mark before rewinding and unloading the tape. VM/370 Monitor data collection is terminated by the system failure.

I/O Devices

If VM/370 monitor data collection is active using tape, a supported tape drive must be dedicated to the system for the duration of the monitoring. For accounting purposes, all I/O is charged to the system.

VM/370 MONITOR DATA VOLUME AND OVERHEAD

Use of the VM/370 Monitor usually requires that three pages be locked in storage for the entire time the VM/370 Monitor is active; however, only two pages are required if the single buffer option is used with only the PERFORM class of data collection enabled. This reduces by three the number of page frames available for paging. This significantly affects the performance of the rest of the system when there is a limited number of page frames available for paging.

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

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

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

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

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

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

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

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

Performance for Time-Shared Multibatch Virtual Machines

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

Monitoring Recommendations

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

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

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

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

LOAD ENVIRONMENTS OF VM/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 processor.

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

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

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

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

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

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

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

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

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

Most of the conditions for good performance, established for the time-shared batch systems, apply equally well to mixed mode systems. However, two major factors make any determination more difficult to make. First, get evidence to show that, in all circumstances, priority is given to maintaining good interactive response, and that nontrivial tasks truly take place in the background. Second, background tasks, no matter how large, inefficient, or demanding should not be allowed to dominate the overall 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 Queue 1 execution time.

Accounting Records

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 as well as T-disk space assigned to virtual machine users, and accounting records generated as a result of user initiated DIAGNOSE X'4C' instruction. A CMS batch virtual machine creates an accounting record with the userid and account number of the user who sent his job to the batch machine. Accounting records are prepared as 80-character card images and 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 happens 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 Records 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 processor time used, including time for VM/370 supervisor functions
37-40	Milliseconds of virtual processor time used
41-44	Number of page reads
45-48	Number of page writes
49-52	Number of virtual machine SIO instructions for nonspooled I/O
53-56	Number of spool cards to virtual punch
57-60	Number of spool lines to virtual printer (this includes one line for each carriage control command)

61-64	Number of spool cards from virtual reader
65-78	Reserved
79-80	Accounting card identification code (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 10.

Accounting Records for LOGON, AUTOLOG, and LINK Journaling

When LOGON, AUTOLOG, and LINK journaling is on, VM/370 may write type 04, type 05, or type 06 records to the accounting data set. These records are written under the following circumstances:

- Type 04 records are written when VM/370 detects that a user has issued enough LOGON or AUTOLOG commands with an invalid password to reach or exceed an installation defined threshold value.
- Type 05 records are written when VM/370 detects that a user has successfully issued a LINK command to a protected minidisk not owned by that user.
- Type 06 records are written when VM/370 detects that a user has issued enough LINK commands with an invalid password to reach or exceed an installation defined threshold value.

These records have the following formats:

Type 04

column	contents
1- 8	USERID specified on the command
9-16	Reserved for IBM use
17-28	Date and time of accounting (mmddyyhhmmss)
29-32	Terminal address
33-40	Invalid password
41-48	USERID that issued the AUTOLOG command
49-51	Reserved for IBM use
52-53	Current invalid password count
54-55	Accounting record limit (JPSLOGAR)
56-78	Reserved for IBM use
79-80	Accounting card identification code (04)

Type 05

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

Type 06

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

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

The information on the accounting card is as follows:

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

For information on using DIAGNOSE code X'4C' see "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 SAVFR2. 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.

Generating Saved Systems

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

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

To generate a saved system:

- Assemble the NAMESYS macro instruction in module DMKSNT.
- Load a new control program nucleus.
- Load the system to be saved and then issue the SAVESYS command.

When allocating DASD space for named systems, provide an extra page for information purposes; do not overlay this area with subsequent named systems.

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=pppp,
      |         | SYSPGNM=(nn,nn,nn-nn,...),
      |         | SYSHRSG=(s,s,...),
      |         | PROTECT = {ON }
      |         |           {OFF}
```

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 (up 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=pppp
is the total number of pages (pppp) 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 four digits.

The STSPGCT operand is not required when assembling the NAMESYS macro using VM/370 Release 3 MACLIBs. The macro itself will calculate the number of pages to be saved.

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 that uses this name. All segments to be shared must be reenterable.

PROTECT = {ON / OFF}
specifies whether CP is to protect shared segments. The default is ON. To turn off segment protection, specify OFF.

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:

```
SAVESYS | systemname
```

where:

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

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

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

Shared Segments

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

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

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

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

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

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

Discontiguous Saved Segments

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

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

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

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

An example of a discontiguous saved segment is the segment of CMS that supports DOS program development and testing under CMS. This segment is reenterable and is named CMSDOS. The VM/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

In order to use discontiguous saved segments, you must:

- Allocate permanent space on a CP-owned volume to contain the saved segment.
- Assign a name to the segment and specify where it is to be stored on disk by defining an entry in the system name table (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=pppp,                         |
|       |         | SYSPGNM=(nn,nn,nn-nn,...),                           |
|       |         | SYSHRSG=(s,s,...),                                   |
|       |         | PROTECT = {ON }                                      |
|       |         |           {OFF}                                      |
```

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 (up 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
: 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. 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
: 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. 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 processing of the SAVE and LOAD commands, 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=pppp
: is the total number of pages (pppp) 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 four digits. The number of pages specified does not have to be a multiple of the number of pages in a segment, but can be some portion of a segment(s).

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 reenterable.

PROTECT = {ON | OFF}
: specifies whether CP is to protect shared segments. The default value is ON. To turn off segment protection, specify OFF.

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

LOADING AND SAVING DISCONTIGUOUS SHARED SEGMENTS

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

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

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

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

The format of the CMS SETKEY command is:

```
| SETKEY | key systemname [startadr] |
```

where:

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

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

startadr — is the starting address (in hexadecimal) at which the keys are to be assigned. The address must be within the address range defined for the saved system or discontiguous saved segments. Using the startadr operand, you can issue the SETKEY command several times and, thus, assign different keys to various portions of the saved system or segment.

HOW THE INTERFACE WORKS

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

Since the virtual machine is responsible for insuring that the discontiguous saved segment 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

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

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

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

The VM/370 default is to protect shared segments. To turn off segment protection, use the NAMESYS macro instruction. This macro instruction can also turn on segment protection. Instructions for using the NAMESYS macro instruction are in the section "The NAMESYS Macro for Saved System."

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

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

Shared segment protection supports:

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

CP's handling of storage keys includes the following:

- No distinction is made between shared and nonshared systems for storage key fetch instruction simulation, DISPLAY command execution, and page key handling.
- A mask in control register 6 prevents the ISK (insert storage key) and SSK (set storage key) instructions from being handled by the VMA feature. This is necessary because VMA updates the key on SSK instructions (including the SWPTABLE fields), but the new value is not detected by the hardware change bit monitoring.

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

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

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

VIRTUAL MACHINE OPERATION

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

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

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

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

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

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

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

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

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

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

```
DMKCDS004E INVALID HEXLOC - xxxxxx
```

The Virtual Machine Communication Facility

The Virtual Machine Communication Facility (VMCF) is part of the CP component of VM/370. VMCF provides virtual machines with the ability to send data to and receive data from any other virtual machine.

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

VMCF is implemented by means of subfunctions invoked using the DIAGNOSE instruction with a code of X'68' and a special 40-byte parameter list called VMCPARM. A VMCF subfunction is indicated by a particular subfunction code in the VMCPFUNC field in the parameter list. Note: Before you can use any other VMCF subfunction, you must use the AUTHORIZE subfunction for communications. Before you can communicate with another user, that user must also have used the AUTHORIZE subfunction.

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

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

Figure 14 lists the VMCF subfunctions and gives a brief description of each. The subfunctions are described in detail in the section "Descriptions of VMCF Subfunctions."

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

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

Function	Code[1]	Comments
AUTHORIZE	Control	Initializes VMCF for a given virtual machine. Once AUTHORIZE is executed, the virtual machine can execute other VMCF subfunctions and receive messages or requests from other users.
UNAUTHORIZE	Control	Terminates VMCF activity.
SEND	Data	Directs a message or block of data to another virtual machine.
SEND/RECV	Data	Directs a message or block of data to another virtual machine, and requests notification of a reply.
SENDX	Data	Directs data to another virtual machine on a faster but more restrictive protocol than the SEND subfunction.
RECEIVE	Data	Allows you to accept selective messages or data sent via a SEND or SEND/RECV subfunction.
CANCEL	Control	Cancels a message or data transfer directed to another user but not yet accepted by that user.
REPLY	Data	Allows you to direct data back to the originator of a SEND/RECV subfunction, simulating full duplex communication.
QUIESCE	Control	Temporarily rejects further SEND, SENDX, SEND/RECV, or IDENTIFY requests from other users.
RESUME	Control	Resets the status set by the QUIESCE subfunction and allows execution of subsequent requests from other users.
IDENTIFY	Control	Notifies another user that your virtual machine is available for VMCF communication.
REJECT	Control	Allows you to reject specific SEND or SEND/RECV requests pending for your virtual machine.

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

Figure 14. Virtual Machine Communication Facility (VMCF) Subfunctions

Using the Virtual Machine Communication Facility

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

VMCF APPLICATIONS

The VM/370 system with VMCF provides the user with the potential to apply new and different techniques to current applications.

Multitasking Programming

The VMCF functions may be used to multitask virtual machines. Each virtual machine can become a subtask (parallel or otherwise) of another virtual machine. A virtual machine task can be a simple program or a large processor. The VMCF functions provide the WAIT/POST, serialization and communication facilities to control such an environment. The existing VM/370 functions provide efficient scheduling, dispatching and basic resource controls. The advantage of such an environment is that a user is less restricted by operating system (software) limitations and gains the flexibility of machine languages and hardware.

Resource Sharing

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

Virtual Extensions to VM/370

It is conceivable that functions could be added to VM/370 without altering the control program (CP). A special privilege class virtual machine could be used to provide additional functions to non-privilege class users using the VMCF interface. Similarly, CMS capabilities could be expanded (or at least appear to be expanded) by linking CMS with other virtual machines.

Program Testing

The program testing capabilities offered by VMCF can range from device simulation to teleprocessing network simulation. In particular, VMCF can be used to provide external interactions from one virtual machine to another. A simulated teleprocessing network could be constructed with virtual machine. Each virtual machine would effectively become a node within the network. The network structure could range from a simple

tree type structure to a complicated multi-path mesh type structure. The program logic within each node virtual machine would be the same logic as required for a real teleprocessing node. In theory, a reasonably complicated structure could be simulated without requiring the physical hardware.

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

INTRA Virtual Machine Communication

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

Virtual Multiprocessing

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

SECURITY AND DATA INTEGRITY

The VMCF interface provides the following security aids:

- The user doubleword in the external interrupt message header can be used to contain a security code to prevent unwarranted users from accessing a shared data base or other confidential information.
- The AUTHORIZE SPECIFIC option allows a user to restrict messages sent to his virtual machine. This option is useful when slave machines are to communicate only with a host machine. The slave machines can AUTHORIZE SPECIFIC with the host and prevent unwarranted users from clogging their message queues.
- The design of VMCF prevents malicious users from intercepting transactions in process for other users. (for example, user D can not execute a RECEIVE, REPLY, REJECT or CANCEL to a message sent to user B from user A).

The VMCF support module is designed such that a user is always informed of conditions that could threaten the integrity of his own data. The user is notified either with a DIAGNOSE X'0068' return code or data transfer error code. There is no internal buffering of user data within the control program (CP), a message is always retained by either the SOURCE or SINK virtual machine. If a SEND type request fails, the SOURCE still has a copy of the original message. If a SINK REPLY fails, the SINK user still has a copy of the REPLY data. The Diagnose return code or data transfer error code can indicate to a user that a transaction failed. It is up to the user to preserve the associated transaction data. The following are considerations which should be noted by a VMCF user:

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

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

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

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

PERFORMANCE CONSIDERATIONS

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

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

- It is to a user's benefit to have the user parameter list, VMCPARM, in the same 4K page as the DIAGNOSE X'0068' instruction. This may eliminate a paging operation.

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

- The VMCF external interrupt masking is controlled by PSW bit 7 and CR0 bit 31. It is to a user's advantage to always have CR0 bit 31 set to 1 (while VMCF is in use) and control the interrupts with PSW bit 7 only. This reduces the number of LCTL instructions.

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

Note: Overall system VM/370 performance is not affected when VMCF is not being used by an installation.

GENERAL CONSIDERATIONS

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

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

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

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

Users should keep copies of VMCPARMs for all requests made via the SEND, SEND/RECV, or SENDX functions. When a final response interrupt is received and the interrupt message header indicates no data transfer errors, the corresponding VMCPARM copy can be released. If a data transfer error is indicated, the copy can be used to reinitiate the transaction.

VMCF Protocol

VMCF provides four types of protocol: SEND, SEND/RECV, SENDX, and IDENTIFY. The protocol used to communicate between two virtual machines depends on the application of VMCF and conventions established by virtual machine users authorized to use VMCF. A virtual machine must invoke the AUTHORIZE subfunction before it is allowed to use any of the other subfunctions.

The types of transactions that virtual machines can be involved in are described by a series of VMCF protocols. In these protocols the originating virtual machine is called the "source" virtual machine. The destination virtual machine is called the "sink" virtual machine.

The protocol for a transaction remains in effect for the duration of the transaction.

THE SEND PROTOCOL

The SEND protocol defines a one-way transfer of data from source virtual machine storage to sink virtual machine storage. The SEND protocol uses the SEND and RECEIVE subfunctions, as described in Figure 15. The source virtual machine first transfers data to the sink virtual machine. This is done by executing the SEND subfunction which specifies the userid of the sink virtual machine, a message ID, and the address and length of the data being sent. The sink virtual machine receives an external interrupt from CP notifying it of the data transfer request. The sink virtual machine can then respond via the RECEIVE subfunction. The RECEIVE request specifies the address and the length of the SINK buffer that is to receive the data and causes the data to be transferred

from source virtual machine storage to sink virtual machine storage. When the data transfer is complete, the source virtual machine receives an external interrupt from CP, indicating that the transaction is complete and that the sink virtual machine has received the data.

All virtual machines authorized to use VMCF can send data using this protocol.

The amount of data transferred is limited only by virtual machine storage size. Data is transferred in blocks of up to 2K (when necessary) and only one real page frame is locked during the data transfer operation.

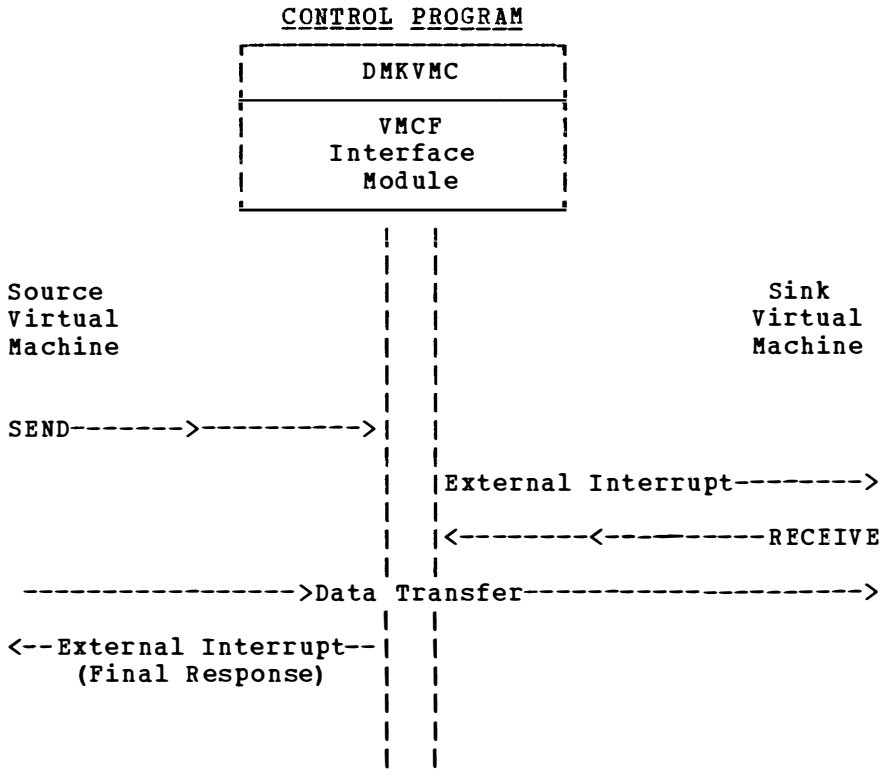


Figure 15. The SEND Protocol

THE SEND/RECV PROTOCOL

The SEND/RECV protocol defines a transaction calling for two-way transfer of data, as described in Figure 16. The SEND/RECV protocol uses the SEND/RECV, RECEIVE, and REPLY subfunctions.

The source virtual machine initiates the transaction using the SEND/RECV subfunction. Using an external interrupt, CP notifies the sink virtual machine that there is a message waiting. The sink virtual machine uses the RECEIVE subfunction to cause the data to be transfered from the source virtual machine's storage to the sink virtual machine storage. The sink virtual machine now uses the REPLY subfunction to cause data to be transferred from its storage to the source virtual machine's storage. When the REPLY subfunction completes processing, CP causes an external interrupt in the source virtual machine, notifying it that the transaction is complete.

The SEND/RECV request requires that the source virtual machine specify the address and length of the data to be transferred and the address where data is expected from the REPLY subfunction. (Both addresses are in source virtual machine storage.) These addresses, along with the length of the data to be transferred, are specified via the VMCPARM parameter list, described below.

When RECEIVE is issued by the sink virtual machine in response to the SEND/RECV request, VMCPARM contains the address in sink virtual machine storage where data is to be received. Finally, when the REPLY request is issued, VMCPARM contains the address in the sink virtual machine storage from which data is to be transferred.

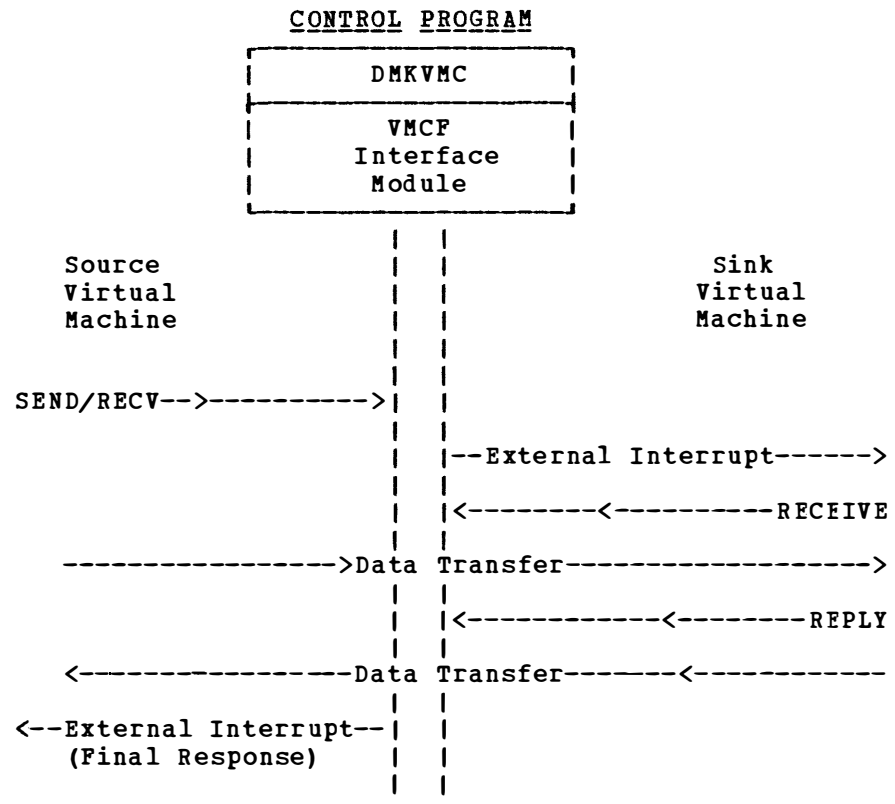


Figure 16. The SEND/RECV Protocol

THE SENDX PROTOCOL

The SENDX protocol defines a transaction calling for an expedited one-way transfer of data. Figure 17 shows the SENDX protocol graphically. SENDX differs from the SEND protocol in that the sink virtual machine need not issue the RECEIVE subfunction; data is transferred from source virtual machine storage to sink virtual machine storage at the same time the external interrupt from CP notifies the sink virtual machine of the transaction. Data sent by the source virtual machine is placed in the external interrupt buffer of the sink virtual machine.

Virtual machines using the SENDX protocol are responsible for specifying the userid for the sink virtual machine, a message ID, the address and length of the data being sent, and the external interrupt buffer address and data length for the sink virtual machine. A virtual machine to be used as a sink virtual machine with the SENDX protocol must specify this information via VMCPARM when that virtual machine issues the AUTHORIZE subfunction. The data length specified must be at least as long as the maximum amount of data to be transferred during a transaction; it need not be limited to the usual 40-byte external interrupt buffer. Effective use of the SENDX protocol requires that VMCF users agree on a maximum size for SENDX data and then issue the AUTHORIZE subfunction with the appropriate external interrupt buffer size.

If the sink virtual machine has not provided enough SENDX buffer area in the external interrupt buffer, CP notifies the source virtual machine that the transaction was not completed.

When a SENDX data transfer is complete, CP directs a response external interrupt to the source virtual machine, notifying it that the transaction is complete.

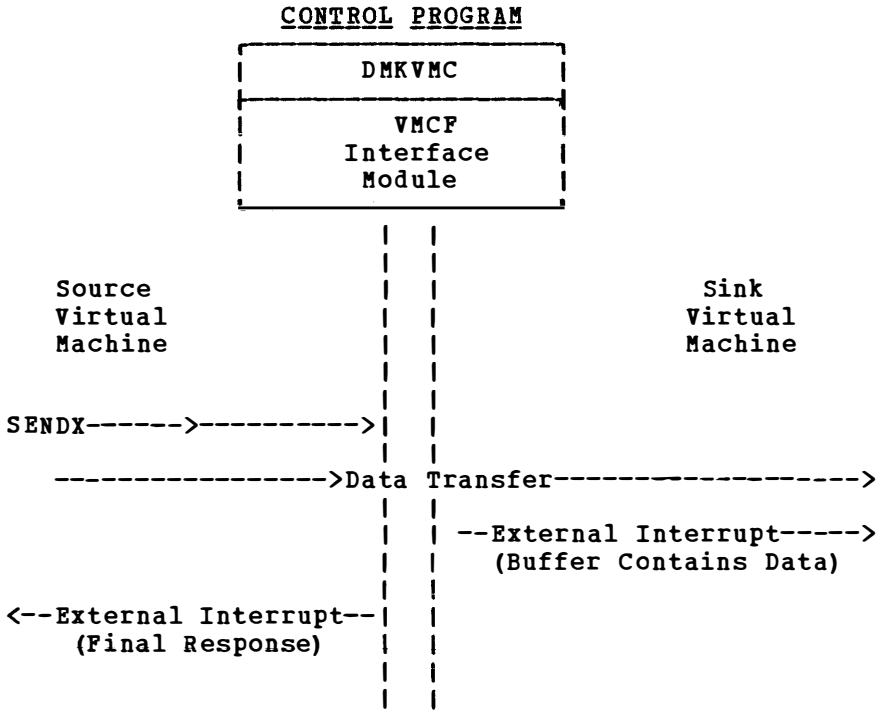


Figure 17. The SENDX Protocol

The IDENTIFY protocol defines a means for virtual machines to identify themselves to other virtual machines by passing user-defined control information via a standard VMCF message header. Figure 18 shows the IDENTIFY protocol graphically.

When the IDENTIFY subfunction is issued, CP directs an external interrupt to the sink virtual machine. Along with the external interrupt, the sink virtual machine receives a standard VMCF message header that contains user-defined information. The IDENTIFY protocol does not cause a response external interrupt to be directed the source virtual machine.

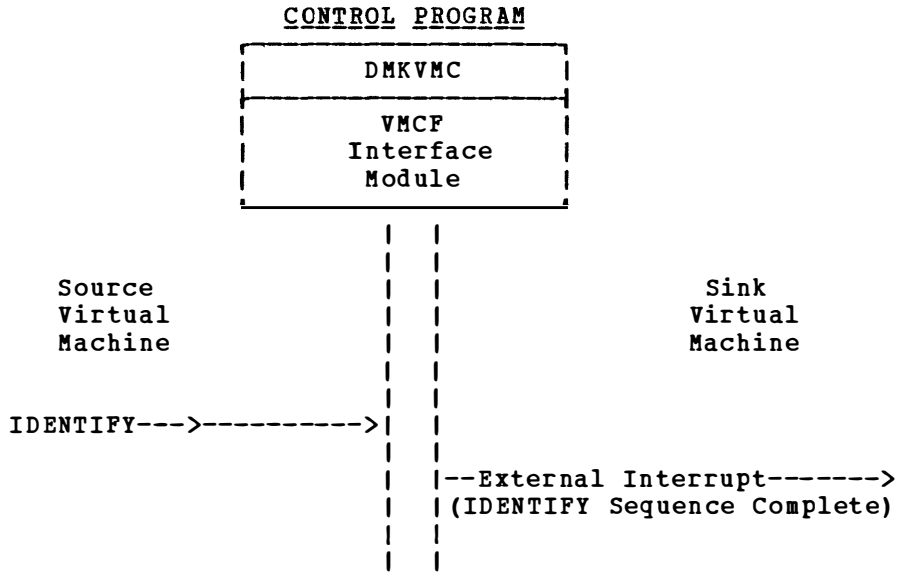


Figure 18. The IDENTIFY Protocol

Descriptions of VMCF Subfunctions

There are two types of VMCF subfunctions: data transfer and control.

THE CONTROL SUBFUNCTIONS

The VMCF control subfunctions allow efficient management of data transfer operations from your virtual machine console. The control subfunctions are: AUTHORIZE, UNAUTHORIZE, CANCEL, QUIESCE, RESUME, IDENTIFY, and REJECT.

AUTHORIZE: DIAGNOSE Code X'68' Subfunction Code X'0000'

AUTHORIZE enables VMCF for a virtual machine; once AUTHORIZE has been executed, the virtual machine can execute other VMCF subfunctions and receive messages and data from other authorized VMCF virtual machines. It is possible to specify three options with the AUTHORIZE subfunction: SPECIFIC, PRIORITY, and VMCPSMSG.

The SPECIFIC option authorizes communication with a specific virtual machine. Any messages sent to the virtual machine from other than the specified virtual machine will be rejected. The SPECIFIC option can be used in an application where virtual machines desire to communicate with a master controller but not among themselves. Under the special message facility, CP is authorized with every virtual machine that is to receive messages sent via the SMSG command. Virtual machines that are to receive messages must authorize themselves.

The PRIORITY option allows a virtual machine to authorize the receipt of priority messages. A virtual machine is allowed to send priority messages to another virtual machine only if the other virtual machine is authorized to receive priority messages. A priority message is one that will be queued ahead of nonpriority messages and therefore accepted first.

When you execute the AUTHORIZE subfunction, you must specify the address and length of the external interrupt buffer for your virtual machine. The buffer must be large enough to contain a fixed message header (40 bytes). The message header identifies messages sent by other virtual machines or responses to messages you might send to your own virtual machine.

If you are going to accept SENDX-type communications, you must specify the size of the external interrupt buffer as 40 plus the maximum size of SENDX data that you plan to accept. This has the effect of authorizing SENDX protocol. That is, a virtual machine may receive data along with the external interrupt in its external interrupt buffer. When a virtual machine sends data to another virtual machine via the SENDX subfunction the data must fit in that virtual machine's external interrupt buffer or the subfunction will be rejected. Messages sent via the special message facility require a buffer length of 169 bytes.

Any AUTHORIZE options in effect can be reset or changed by reexecuting the AUTHORIZE subfunction. If there are errors during execution of the AUTHORIZE subfunction, a virtual machine's authorization status is not changed.

UNAUTHORIZE: DIAGNOSE Code X'68' Subfunction Code X'0001'

UNAUTHORIZE terminates VMCF activity for a virtual machine. The UNAUTHORIZE subfunction causes any stacked or queued messages associated with the virtual machine to be purged. A virtual machine should execute the QUIESCE subfunction before executing UNAUTHORIZE if messages that are already queued are to be handled. When a virtual machine executing UNAUTHORIZE has pending final response external interrupts, the interrupts are purged. If a virtual machine has pending SEND external interrupts from another source virtual machine, a RESPONSE interrupt is reflected to the source indicating that the virtual machine is no longer available.

CANCEL: DIAGNOSE Code X'68' Subfunction Code X'0006'

CANCEL cancels a message or data transfer pending for but not accepted by another VMCF virtual machine. A virtual machine can CANCEL messages it originates with SEND, SENDX, or SEND/RECV subfunctions. A message cannot be canceled if any of the following conditions exist:

- The request was SENDX or IDENTIFY and the sink had already received the SEND external interrupt.
- The request was SEND and the sink had already executed the RECEIVE or REJECT subfunctions.
- The request was SEND/RECV and the sink had already executed the REPLY or REJECT subfunctions.

If the original request was SEND/RECV and the sink virtual machine had executed the RECEIVE subfunction but not the REPLY, the REPLY can be canceled. A virtual machine is notified of this condition with a DIAGNOSE return code. (For a description of the return codes, see Figure 19).

QUIESCE: DIAGNOSE Code X'68' Subfunction Code X'0008'

QUIESCE temporarily rejects SEND, SENDX, SEND/RECV, or IDENTIFY requests from other virtual machines. QUIESCE allows a virtual machine to receive any stacked or queued messages but reject further SEND, SENDX, IDENTIFY, or SEND/RECV requests from other virtual machines. QUIESCE can be used to indicate to other virtual machines that the virtual machine is in QUIESCE status, authorized for communication but not able to accept messages at this time (e.g., entering slowdown, my buffers are full, try again later). The IDENTIFY subfunction could be used to inform other virtual machines that a particular user is no longer in QUIESCE status. You should execute the QUIESCE subfunction before executing the UNAUTHORIZE subfunction to avoid losing messages (see "UNAUTHORIZE: DIAGNOSE Code X'68' Subfunction Code X'0001'"). A virtual machine can reset the QUIESCE status (exit slowdown) by executing the RESUME subfunction. (See "RESUME: DIAGNOSE Code X'68' Subfunction Code X'0009'"). A virtual machine in QUIESCE status may continue to send messages to other virtual machines. QUIESCE status for a virtual machine only affects messages sent from other virtual machines.

RESUME: DIAGNOSE Code X'68' Subfunction Code X'0009'

RESUME cancels the QUIESCE status, allowing your virtual machine to resume reception of VMCF requests from other virtual machines. You can use the IDENTIFY subfunction to inform other virtual machines that your virtual machine is no longer in QUIESCE status. (see "IDENTIFY: DIAGNOSE Code X'68' Subfunction Code X'000A'").

IDENTIFY: DIAGNOSE Code X'68' Subfunction Code X'000A'

IDENTIFY notifies another virtual machine that your virtual machine is available for VMCF communication. Use the IDENTIFY subfunction after issuing the AUTHORIZE subfunction or after your virtual machine has been in the VMCF QUIESCE state and you have issued the RESUME subfunction. IDENTIFY causes an external interrupt to be stacked for a specified virtual machine. The virtual machine executing the IDENTIFY subfunction specifies the userid of the user to receive the external interrupt. The external interrupt identifies the virtual machine executing the IDENTIFY subfunction. The IDENTIFY subfunction is provided to inform a host or controller virtual machine that a virtual machine is activated (logged on) and ready for VMCF communication. The IDENTIFY subfunction can also be used to inform other virtual machines that your virtual machine has exited QUIESCE state. There is no response external interrupt associated with the IDENTIFY subfunction.

The IDENTIFY subfunction can also be used to pass virtual machine defined control information. The fields in the VMCF parameter list (VMCPARM) not used by the IDENTIFY subfunction may be used to contain additional virtual machine data.

REJECT: DIAGNOSE Code X'68' Subfunction Code X'000B'

REJECT selectively rejects pending SEND or SEND/RECV requests from other VMCF virtual machines. REJECT causes a response external interrupt to be reflected to the originator of a message. The external interrupt indicates to the originator that the message was rejected. The user doubleword within the external interrupt header may tell a user why the message was rejected. When the user of a virtual machine executes the REJECT subfunction, he specifies within the VMCF parameter list (VMCPARM) the message ID of the message to be rejected. A virtual machine cannot reject a message sent with the SENDX subfunction since the message is received at the same time the external interrupt is received. The REJECT subfunction can be executed as response to either SEND or SEND/RECV requests.

THE DATA TRANSFER FUNCTIONS

The data transfer operations are SEND, SEND/RECV, SENDX, RECEIVE, and REPLY. These operations involve the movement of data from one virtual machine storage to another virtual machine storage.

SEND: DIAGNOSE Code X'68' Subfunction Code X'0002'

SEND directs a message or block of data to another virtual machine. Specify the virtual address and length of data to be sent within the user parameter list (VMCPARM). Also, specify in the parameter list a message ID to be associated with the message and the userid of the user to receive the message (data). You can also send a doubleword of data to be transmitted within the external interrupt message header (refer to the section "VMCF User Doubleword"). If the SEND subfunction is executed with a data length of zero, only the user doubleword is transmitted to the sink virtual machine. The sink virtual machine can then respond with a RECEIVE subfunction (zero length) and pass back a doubleword of data to the source virtual machine. The external interrupt message header identifies the SEND request. When the sink virtual machine executes a RECEIVE subfunction, the message is transmitted from the source virtual machine storage to the sink virtual storage. There is no internal buffering of data within the control program (CP). All data is transferred in 2K blocks from virtual storage to virtual storage. Data is transferred in 2K blocks to test for STORE/FETCH protection violations. When the data transfer subfunction is complete, the source virtual machine receives a response external interrupt indicating that the SEND request is complete. The sink virtual machine receives a DIAGNOSE X'68' return code indicating that the RECEIVE subfunction is complete. The return code can indicate error conditions associated with the RECEIVE function or normal completion.

The sink virtual machine has the option to reject a message rather than execute the RECEIVE subfunction (see "REJECT: DIAGNOSE Code X'68' Subfunction Code X'0011'"). The source virtual machine may cancel a SEND request before the sink virtual machine has executed a RECEIVE subfunction or REJECT function (see "CANCEL: DIAGNOSE Code X'68' Subfunction Code X'0006'").

If you are executing the SEND subfunction, you may specify the PRIORITY option. The PRIORITY option causes the external interrupt for the sink virtual machine to be queued ahead of all other nonpriority external interrupts. If there are other PRIORITY external interrupts pending for the sink virtual machine, the queuing is done in a first in first out manner. That is, PRIORITY interrupts are queued FIFO among themselves but ahead of all nonpriority interrupts.

SEND/RECV: DIAGNOSE Code X'68' Subfunction Code X'0003'

SEND/RECV provides the capability to both send and receive data in a single VMCF transaction. The SEND/RECV subfunction causes an external interrupt to be queued for the sink virtual machine. When the sink virtual machine receives the external interrupt, it can respond with the RECEIVE subfunction. The RECEIVE subfunction causes data to be transferred from the source virtual storage to sink virtual storage. The sink virtual machine can then respond with a REPLY subfunction. The REPLY subfunction causes data to be transferred from specified sink virtual storage to a REPLY buffer in the source virtual storage. The source virtual machine then receives a response external interrupt indicating that the SEND/RECV request is complete.

When the source virtual machine executes the SEND/RECV function it specifies the address and length of both the SEND buffer and REPLY buffer. The address and length specifications are contained within the user parameter list (VMCPARM). The user parameter list also contains a message ID and userid of the user to receive the data (see the "VMCPARM Parameter List").

The source virtual machine can cancel a previously executed SEND/RECV request provided the sink virtual machine has not yet executed the REPLY or REJECT subfunction. If the sink virtual machine has already executed the RECEIVE subfunction, only the REPLY can be canceled (see "CANCEL: DIAGNOSE Code X'68' Subfunction Code X'0006'").

The sink virtual machine can execute the REJECT subfunction in response to the SEND/RECV request and cause the entire operation to be terminated (see "REJECT: DIAGNOSE Code X'68' Subfunction Code X'0011'").

The sink virtual machine can respond to a SEND/RECV request with the REPLY subfunction without executing the RECEIVE subfunction. This has the effect of informing the source virtual machine that the sink virtual machine cannot accept data but that it can send data. The source virtual machine could have executed the SEND/RECV subfunction only as a means to solicit data from the sink virtual machine. The application of this protocol is up to VMCF users. The user doubleword can be used as a means to control such an application (see "VMCF User Doubleword").

You can execute a SEND/RECV request using the PRIORITY option. The PRIORITY option causes the sink external interrupt for the SEND/RECV request to be queued ahead of any other nonpriority external interrupts. Response external interrupts directed to the source of a PRIORITY message are also queued in priority order.

SENDX: DIAGNOSE Code X'68' Subfunction Code X'0004'

SENDX directs data to another virtual machine via a faster but more restrictive protocol than the SEND subfunction. SENDX subfunction data reaches the sink virtual machine at the same time the SEND external interrupt reaches the sink. In order to use the SENDX subfunction, the sink virtual machine must have an external interrupt buffer large enough to contain both the standard message header and the data. The size of the external interrupt buffer is specified when you execute the AUTHORIZE subfunction. Attempts to execute SENDX are rejected when the sink virtual machine's external interrupt buffer is not large enough to contain the data. After the sink virtual machine receives the SEND external interrupt and data, a response external interrupt is directed to the source virtual machine. The SENDX subfunction eliminates the need for a sink virtual machine to execute a RECEIVE subfunction.

A SENDX request can be canceled by the source virtual machine provided the SENDX external interrupt has not yet been reflected to the sink virtual machine (see "CANCEL: DIAGNOSE Code X'68' Subfunction Code X'0006'").

Specify the SENDX buffer address and length in the user parameter list (VMCPARM). The message ID and userid of the sink virtual machine are also specified in VMCPARM.

The SENDX subfunction can be executed with the PRIORITY option allowing the SEND external interrupt to be queued ahead of all nonpriority external interrupts for the sink virtual machine.

A SENDX request cannot be rejected by the sink virtual machine since the message is received at the same time the external interrupt is received.

You can execute the SENDX subfunction with a zero data length causing only the message header and user doubleword to be transmitted.

RECEIVE: DIAGNOSE Code X'68' Subfunction Code X'0005'

RECEIVE allows you to selectively accept messages or data sent via the SEND or SEND/RECV subfunctions. You must specify in the user parameter list (VMCPARM) the virtual address and length of the RECEIVE buffer. The parameter list also contains the message ID of the message to be received and userid of the virtual machine that originated the SEND or SEND/RECV request. When a virtual machine has more than one message pending, the RECEIVE function can be executed to select messages in any order by message ID.

You can execute the REJECT function in order to reject messages sent by other virtual machines. The REJECT subfunction terminates the SEND or SEND/RECV request (see "REJECT: DIAGNOSE Code X'68' Subfunction Code X'0011'").

You can execute the RECEIVE subfunction in response to a SEND/RECV request and then execute a REJECT subfunction rather than a REPLY. The user doubleword passed back with the REJECT subfunction could indicate "RESEND", for example, if the original data was not received correctly (depending on how you want to use the protocol).

REPLY: DIAGNOSE Code X'68' Subfunction Code X'0007'

REPLY allows you to direct data back to the sender of a SEND/RECV subfunction. (This simulates full duplex communication.) The REPLY subfunction is used with the SEND/RECV subfunction. A user who receives a SEND/RECV external interrupt normally responds by executing the RECEIVE subfunction. The RECEIVE subfunction causes data to be transferred from the source virtual storage to the sink virtual storage. The sink virtual machine can then respond with the REPLY subfunction causing data to be transferred from specified sink virtual storage to the source virtual storage. The REPLY subfunction causes a response external interrupt to be reflected to the source virtual machine.

The user parameter list (VMCPARM) identifies the virtual buffer address and length of reply data. When the REPLY subfunction is executed, the user parameter list (VMCPARM) also contains the message ID and the userid of the virtual machine to receive the reply.

The REPLY subfunction can be executed with a zero data length indicating no response. You can transmit a reply (zero length or otherwise) using the user doubleword.

A reply can be executed in response to a SEND/RECV request without executing the RECEIVE subfunction. This indicates that you do not want to receive the message but may want to send a reply. A reply of zero length could be executed simply to terminate the SEND/RECV request. The application of the REPLY subfunction is a user decision. It must be used to terminate a SEND/RECV request, however, unless the REJECT subfunction is executed (see "REJECT: DIAGNOSE Code X'68' Subfunction Code X'0011'"). The REPLY subfunction is complete when the source virtual machine receives the external interrupt response.

A REPLY subfunction cannot be executed in response to a SEND request (this is a protocol violation).

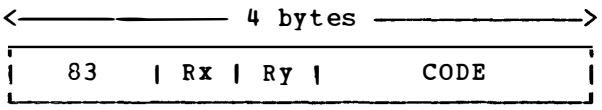
Invoking VMCF Subfunctions

VMCF subfunctions are invoked by means of:

- DIAGNOSE code X'68' subfunction codes
- The VMCPARM parameter list
- External interrupt code X'4001'
- The external interrupt message header

DIAGNOSE CODE X'68'

All VMCF subfunctions are invoked from within assembler language programs by means of DIAGNOSE code X'68':



where:

83 is X'83' and interpreted by the assembler as the DIAGNOSE instruction.

Note: There is no mnemonic for DIAGNOSE.

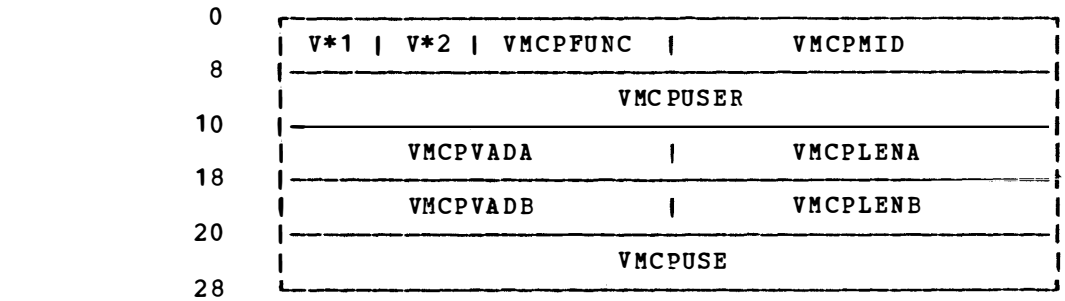
Rx specifies a register containing the address of the VMCPARM parameter list.

Ry is a register that contains a return code.

| CODE is X'0068' and specifies that you are requesting execution of a VMCF.

THE VMCPARM PARAMETER LIST

The Rx register of DIAGNOSE X'68' contains the address of a parameter list (VMCPARM). This parameter list is used to specify the VMCF subfunction to be executed, along with other information required by VMCF to execute that function. The address of VMCPARM must be doubleword-aligned. The following is the format of the VMCPARM parameter list and a description of each of the fields in that list.



where:

V*1
(VMCPFLG1)
is a flag byte used to specify options associated with a particular subfunction.

This flag byte can be set to the following values:

VMCPAUTS (X'80')
Indicates, for the AUTHORIZE subfunction, an AUTHORIZE SPECIFIC request. When this bit is set, the VMCPUSER field must contain the userid of the sink virtual machine. The status of the specified sink virtual machine is not checked by the control program (CP) at this time.

VMCPPRTY (X'40')
Indicates, for SEND, SEND/RECV, SENDX, and IDENTIFY requests, a PRIORITY message request. For an AUTHORIZE request, it indicates an AUTHORIZE PRIORITY request. You cannot send PRIORITY messages to another virtual machine unless that virtual machine has been authorized for PRIORITY messages. The SEND and RESPONSE external interrupts for a PRIORITY message are queued ahead of pending nonpriority external interrupts.

VMCPSMSG (X'20')
Indicates that the virtual machine will accept messages sent via the SMSG command.

Bits 3 through 7 are reserved for IBM use.

V*2
(VMCPFLG2)
Reserved for IBM use.

VMCPFUNC Contains the halfword DIAGNOSE X'68' subfunction code that defines the VMCF subfunction being requested as follows:

Command	Hexadecimal Code	Subfunction
VMCPAUTH	X'0000'	AUTHORIZE
VMCPUAUT	X'0001'	UNAUTHORIZE
VMCPSEND	X'0002'	SEND
VMCPSENR	X'0003'	SEND/RECV
VMCPSENX	X'0004'	SENDX
VMCPRECV	X'0005'	RECEIVE
VMCPCANC	X'0006'	CANCEL
VMCPREPL	X'0007'	REPLY
VMCPQUIE	X'0008'	QUIESCE
VMCPRESM	X'0009'	RESUME
VMCPIDEN	X'000A'	IDENTIFY
VMCPRJCT	X'000B'	REJECT

VMCPMID Contains a unique message identifier associated with a transaction. The source virtual machine must originate the message ID for SEND, SEND/RECV, and SENDX requests. The message ID is used by the sink virtual machine (along with VMCPUSER) to respond to the source request with a RECEIVE, REPLY, or REJECT request. The message ID allows the sink virtual machine to selectively RECEIVE, REPLY, or REJECT messages when more than one message is enqueued. The message ID is used by both the source and sink as a unique identification for all messages. You may send messages with the same message ID to multiple users; you cannot send multiple messages with the same message ID to one user. Once a transaction is completed, however, the message ID may be reused.

VMCPUSER Specifies the userid of the sink virtual machine for SEND, SEND/RECV, SENDX, IDENTIFY, and CANCEL requests and the userid of the source virtual machine for RECEIVE, REPLY, and REJECT requests. The sink virtual machine uses this field in combination with the message ID (VMCPMID) to respond to source requests. When the original source parameter list VMCPARM is passed to the sink as the message header VMCMHDR, the userid is changed from sink to source.

This field is also used to specify the SPECIFIC userid for an AUTHORIZE SPECIFIC request.

VMCPVADA Contains one of four addresses, depending upon which VMCF subfunction is requested:

For SEND, SEND/RECV, and SENDX requests, VMCPVADA contains the address of the source virtual machine data. For RECEIVE requests, VMCPVADA contains the address of a sink virtual machine RECEIVE buffer. For REPLY requests, VMCPVADA contains the address in sink virtual machine storage where REPLY data is located. For an AUTHORIZE request, VMCPVADA specifies the address of the virtual machine external interrupt buffer.

The length of the associated data or buffer is specified in the VMCPLENA field.

VMCPLENA Contains the length of the data sent by a user, the length of a RECEIVE buffer, or the length of an external interrupt buffer, whichever is specified in the field VMCPVADA. The size of the value specified in VMCPLENA is restricted only by virtual machine storage size.

The sink virtual machine can use the value in this field as the data length for RECEIVE operations.

VMCPVADB Contains the address of a source virtual machine's REPLY buffer for a SEND/RECV request. When the sink virtual machine issues a REPLY in response to a SEND/RECV from the source virtual machine, the REPLY data is moved in this buffer. The length of the REPLY buffer is contained in the field VMCPLENB.

VMCPLENB Specifies the length of the source virtual machine's REPLY buffer. The sink virtual machine uses this field to determine the maximum length of the REPLY. A corresponding field within the response message header contains a residual data count. The source virtual machine uses this residual count to determine the length of the sink reply. The original REPLY buffer length (less the residual count) is the length of the REPLY from the sink virtual machine.

VMCPUSE Contains the VMCF user doubleword. The user doubleword is transmitted to the sink virtual machine in the SEND message header for SEND, SEND/RECV, SENDX, and IDENTIFY requests. For RECEIVE, REPLY, and REJECT requests, the user doubleword is transmitted to the source virtual machine within the RESPONSE message header. The sink virtual machine can transmit the user doubleword to the source virtual machine with REJECT or REPLY requests only if the original request was a SEND/RECV. The user doubleword is transmitted only with requests that result in SEND or RESPONSE external interrupts.

The following chart summarizes the VMCPARM fields required for execution of each of the VMCF subfunctions. Possible return codes associated with each subfunction are also listed. A discussion of the return codes and their meanings can be found in the section "DIAGNOSE X'0068' RETURN CODES."

VMCF Subfunction	Applicable VMCPARM Parameters	Return Codes
AUTHORIZE	VMCPFLG1 - SPECIFIC/PRIORITY option VMCPFUNC - X'0000' - subfunction code VMCPUSER - SPECIFIC userid VMCPVADA - external interrupt buffer address VMCPLENA - external interrupt buffer length	0,1,2,6,15
UNAUTHORIZE	VMCPFUNC - X'0001' - subfunction code	0,2,4,15
SEND	VMCPFLG1 - PRIORITY option VMCPFUNC - X'0002' - subfunction code VMCPMID - message identifier VMCPUSER - sink userid VMCPVADA - SEND data address VMCPLENA - SEND data length VMCPUSE - user doubleword (See Note)	0,1,2,4,5,8 9,10,15,18
SEND/RECV	VMCPFLG1 - PRIORITY option VMCPFUNC - X'0003' - subfunction code VMCPMID - message identifier VMCPUSER - sink userid VMCPVADA - SEND data address VMCPLENA - SEND data length VMCPVADB - REPLY buffer address VMCPLENB - REPLY buffer length VMCPUSE - user doubleword	0,1,2,4,5,8,9, 10,15,18
SENDX	VMCPFLG1 - PRIORITY option VMCPFUNC - X'0004' - subfunction code VMCPMID - message identifier VMCPUSER - sink userid VMCPVADA - SEND data address VMCPLENA - SEND data length VMCPUSE - user doubleword (See Note)	0,1,2,4,5,7,8, 9,10,15,18
RECEIVE	VMCPFUNC - X'0005' - subfunction code VMCPMID - message identifier VMCPUSER - source userid VMCPVADA - RECEIVE buffer address VMCPLENA - RECEIVE buffer length VMCPUSE - user doubleword	0,1,3,2,4,5,6, 12,13,15,16,17

Note: Fields within the user parameter list that are not used by a particular subfunction may be used to contain additional user data. The data, however, can only be passed to the sink virtual machine by the source virtual machine. The REPLY buffer address and length fields (VMCPVADB+VMCPLENB) may be used to transmit additional user data for SEND and SENDX requests. All fields except VMCPFLG1, VMCPFLG2, VMCPFUNC, and VMCPUSER may be used to pass control information with an IDENTIFY request.

Figure 19. VMCF Subfunctions, Parameters, and Return Codes (Part 1 of 2)

VMCF Subfunction	Applicable VMCPARM Fields	Return Codes
CANCEL	VMCPFUNC - X'0006' - subfunction code VMCPMID - message identifier VMCPUSER - sink userid	0,2,3,4,5,11, 12,14,15,20
REPLY	VMCPFUNC - X'0007' - subfunction code VMCPMID - message identifier VMCPUSER - source userid VMCPVADA - REPLY data address VMCPLENA - REPLY data length VMCPUSE - user doubleword	0,1,2,3,4,5,6, 12,13,15,16,17,19
QUIESCE	VMCPFUNC - X'0008' - subfunction code	0,2,4,15
RESUME	VMCPFUNC - X'0009' - subfunction code	0,2,4,15
IDENTIFY	VMCPFLG1 - PRIORITY option VMCPFUNC - X'000A' - subfunction code VMCPUSER - sink userid VMCPUSE - user doubleword (See Note)	0,2,4,5,9,10 15,18
REJECT	VMCPFUNC - X'000B' - subfunction code VMCPMID - message identifier VMCPUSER - source userid VMCPUSE - user doubleword	0,2,3,4,12,13,15

Note: Fields within the user parameter list that are not used by a particular subfunction may be used to contain additional user data. The data, however, can only be passed to the sink virtual machine by the source virtual machine. The REPLY buffer address and length fields (VMCPVADB+VMCPLENB) may be used to transmit additional user data for SEND and SENDX requests. All fields except VMCPFLG1, VMCPFLG2, VMCPFUNC, and VMCPUSER may be used to pass control information with an IDENTIFY request.

Figure 19. VMCF Subfunctions, Parameters, and Return Codes (Part 2 of 2)

EXTERNAL INTERRUPT CODE X'4001'

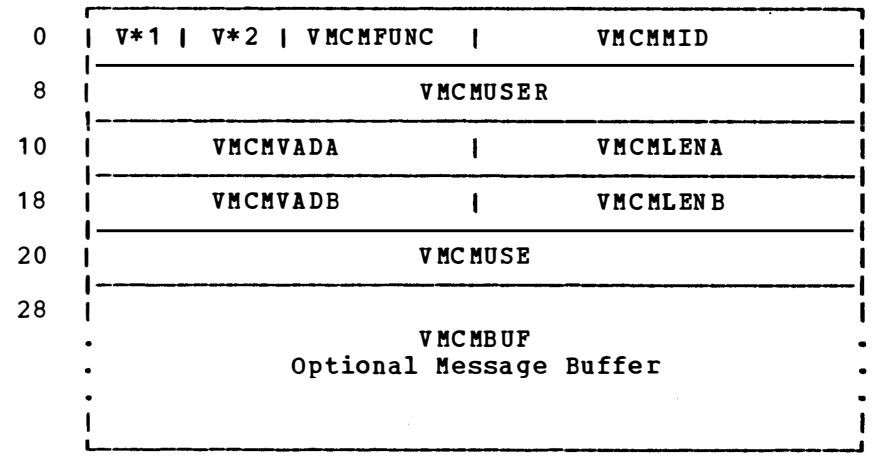
External interrupt code X'4001' is a special interrupt code recognized by CP as part of a VMCF transaction. Just as virtual machines use the DIAGNOSE instruction to communicate with CP, so too CP uses this interrupt code to communicate with virtual machines. External interrupt code X'4001' and DIAGNOSE code X'68' provide the mechanism VMCF uses to synchronize message processing.

The External Interrupt Message Header

Associated with external interrupt code X'4001' is a storage area referred to as the external interrupt message header. The external interrupt message header (VMCMHDR) contains the control information required to SEND and RECEIVE messages. The fields within the message header are, for the most part, a copy of VMCPARM parameter list fields.

CP passes the external interrupt buffer (containing the external interrupt message header) to the user's interrupt handler for processing. The user must specify the address and length of this buffer when he executes the AUTHORIZE subfunction. Then, when the user sends or receives messages, CP knows the address of the buffer and passes it to the appropriate interrupt handler routine.

Fields VMCMFUNC through VMCMUSE correspond to the fields VMCPFUNC through VMCPUSE in the VMCPARM parameter list transmitted by the source virtual machine. The format of the message header and optional SENDX data buffer is:



where:

V*1 (VMCMSTAT) is a status byte associated with the message header. The bits within the status byte are defined as follows:

VMCMRESP (X'80')
Indicates final external interrupt (transaction complete). This bit will be set for all RESPONSE external interrupts and the SEND external interrupt resulting from an IDENTIFY request.

VMCMRJCT (X'40')
This bit is set in a RESPONSE external interrupt to indicate that the sink virtual machine rejected the message via the REJECT subfunction.

VMCMPRTY (X'20')
This bit is set in both SEND and RESPONSE external interrupts to indicate a priority message. A virtual machine must be authorized for priority messages before it can receive them.

V*2 (VMCMEFLG) Contains a data transfer error code indicating success or errors associated with a data transfer operation. (Refer to the section "Data Transfer Error Codes.")

VMCMFUNC Contains the subfunction code of the original request. The sink virtual machine will use this field to determine the type of request. The possible subfunction codes are:

```
VMCPSEND  X'0002' - SEND
VMCPSENR  X'0003' - SEND/RECV
VMCPSENX  X'0004' - SENDX
VMCPIDEN  X'000A' - IDENTIFY
```

VMCMMID Contains the message ID associated with the original source request.

VMCMUSER Contains the userid of the source virtual machine for SEND external interrupts and the userid of the sink virtual machine for RESPONSE external interrupts. If a SMSG command was issued, 'SYSTEM' appears in this field.

VMCMVADA Contains the address of the original SEND data for SEND requests. This field would normally have no meaning to the sink virtual machine.

VMCMLENA Indicates the length of SEND data for SEND external interrupts. It indicates a data transfer residual count for RESPONSE external interrupts.

VMCMVADB Contains the virtual address of the REPLY buffer for SEND/RECV requests. This field has no meaning to the sink virtual machine.

VMCMLENB Contains the length of the source virtual machine REPLY buffer for SEND/RECV external interrupts; contains the residual REPLY count for RESPONSE external interrupts. The sink virtual machine uses this field to determine the maximum length of the REPLY; the source virtual machine uses this field to determine the length of the sink virtual machine REPLY data.

VMCMUSE Contains the user doubleword, which is transmitted to the sink virtual machine with SEND external interrupts and to the source virtual machine with RESPONSE external interrupts. If a SMSG command was issued, this field contains the user doubleword of the issuer of that command.

VMCMBUF This is the optional data buffer used by the SENDX subfunction. The data sent with the SENDX subfunction is moved into this buffer. The buffer size is specified when a virtual machine executes the VMCF AUTHORIZE subfunction.

VMCF USER DOUBLEWORD

VMCF provides a doubleword for user data that can be transmitted within the external interrupt message header. A user supplies the doubleword of data within the parameter list (VMCPARM) for certain VMCF request (that is, SEND, SENDX, SEND/RECV, RECEIVE, REPLY, IDENTIFY, and REJECT). You can use the user doubleword in any manner you desire. The doubleword is transmitted within the external interrupt message header for both SEND and RESPONSE type external interrupts.

The user doubleword can be used for control information in a user-defined higher level protocol. That is, you could have your own message headers defined within the data transmitted from one virtual machine to another. The user doubleword could be used to control such a protocol.

The user doubleword can also be used as a security code or provide additional information for subfunctions such as IDENTIFY and REJECT. You can specify a zero data length for any VMCF transaction. The effect of this is that only the external interrupt message header with user doubleword is transmitted or received.

DIAGNOSE X'68' RETURN CODES

The virtual machine initiating a VMCF request receives a return code in the general register specified as "Ry" in the DIAGNOSE instruction. The return code indicates successful completion of the request or error conditions associated with the request. Figure 20 is a description of all possible return codes returned to a virtual machine executing the DIAGNOSE X'68' subfunction.

Return Code	Meaning
0	The normal response. Indicates successful completion of a request or successful initiation of a request. For example, for an AUTHORIZE request, 0 indicates that the AUTHORIZE function is complete; for a SEND request, 0 indicates that the SEND was successfully initiated. The SEND request, of course, would not be complete until the final RESPONSE external interrupt was received by the source virtual machine.
1	Invalid virtual buffer address or length. A virtual machine attempted to execute a VMCF subfunction but specified an invalid address or length: • External interrupt buffer not within virtual storage. • External interrupt buffer address not doubleword aligned. • Message data or buffer not within virtual storage. • External interrupt buffer less than the standard message header length.
2	Invalid subfunction code. A virtual machine attempted to execute a VMCF subfunction but specified an unsupported subfunction code.
3	Protocol violation. A virtual machine attempted to execute a subfunction which would violate the defined protocol: • Cancel a message it did not originate. • Reply to a message not sent via SEND/RECV. • REJECT a message that it originated. • Executed more than one RECEIVE to a SEND or SEND/RECV request.
4	Source virtual machine not authorized. A virtual machine attempted to execute a subfunction (other than AUTHORIZE) but was not authorized to use VMCF (had not successfully executed the AUTHORIZE subfunction).

Figure 20. DIAGNOSE Code X'68' Return Codes (Part 1 of 3)

Return Code	Meaning
5	User not available. A virtual machine attempted to execute a function and specified a virtual machine currently not available for VMCF communication: • Not logged on. • Not authorized for VMCF communication. • Virtual machine authorized SPECIFIC for some other virtual machine.
6	Protection violation. A virtual machine attempted to execute a VMCF function that would result in a STORE or FETCH protection violation. The virtual machine specified a data or buffer address that contained a storage key other than its current PSW key (assume key was nonzero). This return code is also set if a virtual machine attempts to receive data in a CP-owned shared segment.
7	SENDX data too large. A virtual machine attempted to execute a SENDX request but specified a SENDX data length larger than the sink virtual machine external interrupt buffer.
8	Duplicate message. A virtual machine attempted to execute a SEND-type function and specified a message ID and virtual machine userid for which there was already an active message.
9	Target virtual machine in QUIESCE status. A virtual machine attempted to execute a SEND-type function and specified a sink virtual machine userid of a virtual machine in QUIESCE status.
10	Message limit exceeded. A virtual machine attempted to execute a SEND subfunction but already had 50 messages active. The virtual machine should clear any pending RESPONSE external interrupts or CANCEL previously sent messages in order to continue processing.
11	REPLY canceled. The source virtual machine executed a CANCEL to a previous SEND/RECV request. The sink virtual machine had already RECEIVED the message but had not yet executed a REPLY. The sink virtual machine REPLY in this case is canceled. The sink virtual machine will get return code 12 (message not found) when it executes the REPLY subfunction.
12	Message not found. A virtual machine attempted to execute a subfunction and specified a message ID and virtual machine userid for a message that does not exist. The message may have existed at one time but could have been cancelled by the originator.

Figure 20. DIAGNOSE Code X'68' Return Codes (Part 2 of 3)

Return Code	Meaning
13	Synchronization error. The sink virtual machine attempted to respond to a message for which it had not yet received the SEND external interrupt. This condition can occur if the sink virtual machine is anticipating certain messages but does not wait for the SEND external interrupt.
14	CANCEL too late. A virtual machine attempted to CANCEL a message that had already been processed. The sink virtual machine had already responded with RECEIVE or REJECT (SEND request) or REPLY or REJECT (SEND/RECV request). This return code is also set if a virtual machine attempts to CANCEL a SENDX request for which the sink virtual machine had already received the SEND external interrupt.
15	Paging I/O error. A virtual machine attempted to execute a subfunction which resulted in an uncorrectable paging I/O error. This is a hardware failure.
16	Incorrect length. A virtual machine executed a RECEIVE or REPLY function and specified a RECEIVE buffer length less than the source virtual machine SEND data length or a REPLY data length larger than the source virtual machine REPLY buffer length. The source virtual machine will receive a data transfer return code identifying the condition.
17	Destructive overlap. A virtual machine executed a RECEIVE or REPLY function and specified a RECEIVE buffer address which overlapped the source virtual machine SEND data address or a REPLY data address that overlapped the source virtual machine REPLY buffer address. This condition can occur only when a virtual machine is sending messages to itself (a "wrap connection").
18	User not authorized for PRIORITY messages. A virtual machine attempted to send a PRIORITY message to a virtual machine that was not authorized to accept PRIORITY messages (that is, had not executed the AUTHORIZE function with the PRIORITY option).
19	Data transfer error. A virtual machine executed a request that resulted in a data transfer error condition associated with the other virtual machine. The return code is returned to the sink virtual machine to indicate that the transaction did not complete successfully.
20	CANCEL - busy. A virtual machine attempted to cancel a message being processed. If this is a SEND/RECV request and the RECEIVE subfunction is in process, repeated retries may cancel the REPLY subfunction.

Figure 20. DIAGNOSE Code X'68' Return Codes (Part 3 of 3)

DATA TRANSFER ERROR CODES

When a virtual machine executes a SEND, SENDX, or SEND/RECV subfunction, the normal DIAGNOSE return code is zero, indicating that the request was successfully initiated. However, when the actual data transfer takes place, errors can occur. All errors occurring at data transfer time are communicated to the source virtual machine in the RESPONSE external interrupt message header, VMCMHDR. Figure 21 shows error codes indicating conditions that are possible after the SENDX, SEND, or SEND/RECV request is initiated. The error codes correspond to DIAGNOSE return code numbers.

Error Code	Meaning
0	The normal response (no errors).
1	Invalid buffer address or length. The SEND and/or RECEIVE buffers used for a data transfer operation are not within the virtual machine's virtual storage. The beginning and ending addresses were valid when a request was initiated but all addresses are not valid.
5	User not available. The sink virtual machine executed the UNAUTHORIZE function, reexecuted the AUTHORIZE SPECIFIC subfunction, or implicitly reset his virtual machine after the source virtual machine request was initiated.
6	Protection violation. The storage key for a virtual machine's SEND or RECEIVE buffer did not match its PSW key at the time the transfer was initiated. (Assume the key was nonzero.) This error code is also set if a virtual machine attempts to RECEIVE data into a CP-owned shared segment.
7	SENDX data is too large. The sink virtual machine reexecuted AUTHORIZE and specified an external interrupt buffer size less than the buffer size at the time a SENDX subfunction was executed. The SENDX data will no longer fit in the sink virtual machine buffer.
15	Paging I/O error. An uncorrectable paging I/O error occurred during the data transfer operation attempting to fetch a virtual machine SEND or RECEIVE buffer. This is a hardware failure.
16	Incorrect length. The sink virtual machine executed a RECEIVE subfunction with a data length (VMCPLENA) smaller than the original SEND data length or a REPLY subfunction with a REPLY data length larger than the source virtual machine REPLY buffer length.
17	Destructive overlap. A virtual machine was communicating with itself in a "wrap connection" and his SEND or RECEIVE buffers overlapped one another (intra-virtual machine communication).
19	Data transfer error. A data transfer error occurred which was associated with the other virtual machine. The transaction did not complete successfully.

Figure 21. DIAGNOSE Code X'68' Data Transfer Error Codes

Special Message Facility

The special message facility enables a virtual machine to send messages to another virtual machine by issuing the SMSG command. The special message facility uses the SENDX subfunction of the Virtual Machine Communication Facility (VMCF) to send the messages. However, the sending virtual machine does not need to perform the initialization required by VMCF. Initialization is handled by CP and is described later in this topic.

To send a message, a virtual machine need only prepare the message text -- up to 129 bytes -- and issue the class G command, SMSG. Parameters on the SMSG command identify the USERID of the receiving virtual machine and specify the message text. The format of the message text must be acceptable to the receiving virtual machine. The SMSG command is described in the VM/370 CP Command Reference for General Users.

Before the receiving virtual machine can receive special messages, it must enable itself to receive external interrupts, must set bit 31 in control register 0 to one, and must authorize itself. It authorizes itself by issuing DIAGNOSE Code X'68', Authorize. The parameter list, VMCPARM, specified with DIAGNOSE Code X'68' must contain a pointer to an external-interrupt buffer, must specify a buffer length of 169 bytes, and must have the special message flag (VMCPSMSG) turned on. The receiving virtual machine may turn on this flag by setting VMCPSMSG to a value of B'1'. Optionally, the receiving virtual machine may turn on the special message flag by issuing the class G command, SET SMSG ON. For information on using DIAGNOSE Code X'68', see "Description of VMCF Subfunctions" and "Invoking VMCF Subfunctions."

To understand how a special message is presented to the receiving virtual machine, see "The SENDX Protocol" in the section "VMCF Protocol."

When a virtual machine no longer wishes to accept special messages, it may turn off the special message flag by issuing the command, SET SMSG OFF. To resume receiving messages, the virtual machine may issue the command, SET SMSG ON. CP sends an error message to any virtual machine that attempts to send a special message to another virtual machine that is not accepting special messages.

CP handles VMCF initialization and special message processing as follows. When the SMSG command is issued, CP verifies that no invalid options were specified and that a valid USERID was specified. CP also verifies that the receiving virtual machine is authorized and accepting special messages. CP then obtains storage for the external-interrupt buffer and builds the parameter list required by VMCF. Part of the external interrupt buffer contains the external interrupt message header. Descriptions of the fields contained in this message header are described in the section "The External Interrupt Message Header." Note that the VMCMUSER field contains the word "SYSTEM" while the user doubleword of the user issuing the SMSG command is contained in the VMCMUSE field.

Finally, CP uses the SENDX subfunction of VMCF to send the message to the receiving virtual machine.

VM/370 Use of the IBM 3850 MSS

Virtual machines operating CMS, OS/VS1, or OS/VS2 (MVS) may access mass storage volumes containing VM/370 minidisks or entire mass storage volumes dedicated to the virtual machine. These volumes appear to the virtual machine as 3330 volumes and are accessed using 3330 device support in the virtual machine. VM/370 controls allocation, volume mounting, and volume demounting. Virtual machines that run OS/VS1 or OS/VS2 (MVS) with MSS support can also access mass storage volumes using dedicated device support.

VM/370 Access to the Mass Storage Control

Whenever an MSS 3330V volume must be mounted or demounted, the VM/370 control program first selects an appropriate device address. If a volume mount is required, the device is selected from the pool of available 3330V devices created at system generation time. If a volume must be demounted, CP selects the address of the device on which the volume is currently mounted.

To pass mount, demount, and relinquish orders, the virtual machine must have an MSC port dedicated to it via the ATTACH command or the DEDICATE directory statement. An application program named DMKMSS is distributed as part of VM/370; it acts as an interface between CP and the MSC. After DMKMSS is started in an OS/VS1 or OS/VS2 (MVS) virtual machine, it uses a special virtual I/O device and the VM/370 DIAGNOSE interface to communicate with the VM/370 control program.

DMKMSS first issues a DIAGNOSE Code X'78' subcode X'00' to inform CP that the MSS is available. Until CP receives this interrupt, the control program cannot mount or demount MSS volumes.

When CP receives the interrupt, DMKMSS then obtains from tables found in the MSC (the configuration table), a list of all VUAs (virtual unit addresses) associated with the processor (cpuid). After DMKMSS builds the list, it issues DIAGNOSE Code X'78' subcode X'14' to pass this list to CP. CP builds two tables from the list:

a. one table contains only shared VUAs
b. the other table contains only non-shared VUAs

Each table is also sorted according to SDG (staging drive group) number. The two tables are written to DASD and their slot address (CCPDs) are anchored in DMKMSS.

After DIAGNOSE Code X'78' subcode X'14' is issued with a mount or demount request, CP generates an attention interrupt on the virtual I/O device. When it receives this interrupt, DMKMSS issues another DIAGNOSE Code X'78' to indicate that it is ready to process an MSS request. CP places the required information into a buffer in the virtual machine. The address of this buffer is contained in DIAGNOSE X'78' instruction. The format of the information is described in the MSSCOM control block.

After the request has been processed by the MSC, DMKMSS again issues a DIAGNOSE Code X'78' that indicates that the MSS order is complete and reflects the MSC ending status.

The MSC handles volume mounts, demounts and relinquishes. If the MSC order was for a volume demount, then normal ending status from the MSC indicates that the volume is demounted. The CP RDEVBLOK corresponding to the device is reset to free status, and can now be used for allocation to another task.

If the request is for a volume mount, DMKMSS examines the MSSRETRY bit. This bit is turned on only when CP determines which VUA is used to mount a volume. For example, CP makes the determination for normal MDISKs, LINKs, and DEDICATEs of MSS volumes with no real address specified. If the MSSRETRY bit is off, the mount order is sent to the MSC and the resulting MSC return code is passed back to CP.

However, if CP determines which VUA will mount the volume, DMKMSS scans the MSC mounted volume table (MVT) to see if the volume is already staged in any SDG. If the volume is not staged, then DMKMSS sends the mount order to the MSC and the resulting MSC return code is passed back to CP. If the volume is staged in any SDG, DMKMSS compares that SDG number with the SDG number of the VUA on which CP wants to mount the volume. If a match is found, the mount order is sent to the MSC and the resulting MSC return code is passed back to CP. In the case of a SDG number match, there may be no cartridge picks required to fulfill the mount request. This situation reduces the amount of time spent waiting for a volume mount to complete. It also reduces the amount of mechanical work the MSS has to do.

If the SDG numbers do not match, DMKMSS examines the mount status of the volume. The volume may or may not be mounted. This is because the MVT shows that there are cylinders still allocated to that volume even though it may not be mounted.

If the volume is mounted by another processor, the correct SDG number is placed in the MSSCOM, the VOLMNTD bit in MSSCOM is turned on, and reason code X'10' is returned to CP. If the volume is not mounted (but still has cylinders in the staging space allocated to it), the correct SDG number is placed in MSSCOM, the VOLMNTD bit is left off, and reason code X'10' is returned to CP. For a mount that can be retried and has a reason code of X'10', CP attempts to select a VUA from the correct SDG. If CP finds one, MSSCOM is updated to reflect the new VUA and CP reissues the request. If no VUAs are available in the correct SDG and the VOLMNTD bit in MSSCOM is off, the MSSRETRY bit in MSSCOM is turned off and the original mount request is reissued. If no VUAs in the correct SDG are available, and the VOLMNTD bit is on, the request is terminated with reason code X'10'.

If the MSC request was for a volume mount, the MSC ending status indicated that the MSC was processed. If the MSC accepts the mount order, the MSC will order the staging adapter to generate a pack change interrupt (an unsolicited device end) on the device when that device has been mounted. CP receives the pack change interrupt, the RDEVBLOK is set to indicate that the volume is mounted, and any VM/370 task waiting for the volume is marked dispatchable. If the mount order was rejected, no further processing of the mount request occurs. The previously allocated RDEVBLOK is marked free and processing continues with the next MSS request.

The VM/370 control program destages any changed cylinders on a volume at detach time when the user count for the entire volume goes to zero. Destaging is accomplished when CP issues a relinquish order to the MSS via the central server application program (DMKMSS). A relinquish order is issued at detach time for volids mounted on SYSVIRT virtual unit addresses and VIRTUAL virtual unit addresses. This is the case only when the volume was mounted by the control program on behalf of the guest operating system. Refer to the VM/370 Planning and System Generation Guide that identifies VS1/VS2 APARs to be applied to the central server virtual machine operating system to support the VM/370 relinquish function (APAR 11344).

Asynchronous MSS Mount Processing

When an MSS volume mount is required to satisfy a LINK or ATTACH command or an MDISK or DED directory statement, CP returns control to the virtual machine as soon as MSC accepts the mount request. The virtual machine may continue to execute before the virtual device specified on the MDISK, DED, LINK, or ATTACH is available.

The reasons for asynchronous MSS mount processing are the relatively long time required to complete the mount, and the chance that an error may occur in the MSS after the mount order is accepted. The virtual device to be mounted may not be vital to the specific task to be accomplished. Also, if an error occurs in the MSS (such as a permanent read error on a cartridge) after the mount is accepted, the error indication is passed from the MSC to the virtual machine. VM/370 cannot determine that an error has occurred and that the mount will not complete. If the virtual machine were not dispatchable until the mount completed, it would be locked out until the MSS error was corrected.

With asynchronous mount processing, the virtual machine has the flexibility to either continue processing without the affected virtual device, or wait until the MSS mount completes. If the virtual machine issues an SIO instruction to a virtual device that is defined on the volume being mounted, VM/370 will queue the I/O request until the mount completes. The virtual machine will be marked I/O wait nondispatchable until the mount completes and the SIO is started.

VM/370 Processing of MSS Cylinder Faults

VM/370 supports 3330V cylinder fault processing in two ways: real channel programs directed to 3330Vs are constructed so that cylinder faults can be recognized and the channel program restarted; and the attention interruption (indicating that the cylinder fault has been satisfied) is recognized and any I/O for that device restarted.

When the VM/370 processor issues a seek CCW to a 3330V device, the staging adapter must translate the seek argument to the correct cylinder of staging space. If the cylinder referenced in the seek is staged, then the SIO is passed to the associated staging DASD drive. If the referenced cylinder is not staged, the staging adapter initiates cylinder fault processing. The staging adapter first passes a cylinder fault indication to the MSC, requesting the cylinder of data to be staged. It then returns a status modifier to the channel in response to the seek, which causes the channel to skip one CCW in its CCW fetch processing. That is, the channel does not fetch the next CCW after the seek.

As a result of the cylinder fault, the MSC allocates staging space for the requested data and causes it to be staged. The staging adapter then generates a channel end/device end interruption to indicate that the cylinder has been staged.

It is possible in error situations that the attention interruption may not be received. Each time an I/O request is queued by VM/370 as a result of a cylinder fault, a timer is set. If the timer expires before the interruption is received, a message (DMKSSS074I) is written to the VM/370 system operator and the request is retried.

Backup and Recovery of MSS Volumes

The process of creating backup copies of MSS volumes, and restoring from those backup copies, can be controlled through the OS/VS access methods services COPYV command. This command can operate without system operator intervention.

For each active volume in the MSS, there may be one or more copy volumes. At any time, the active volume may be copied to a copy volume with the access method services COPYV command. All volume mounts and data transfer are controlled by this command. If at any time it is necessary to restore the level of a volume to that of a copy, the OS/VS access methods services RECOVERV command is used.

All the OS/VS access methods services commands can be run from either a real processor or a VS virtual machine. If the MSS communicator virtual machine is in operation, these commands can be run from that virtual machine while it is acting as the communicator.

Timers in a Virtual Machine

This section describes the results obtained in using timers in a virtual machine created by CP.

Interval Timer

Virtual location 80 (X'50'), the interval timer, contains different values than would be expected when operating in a real machine. On a real machine, the interval timer is updated 300 times per second when enabled and when the real machine is not in manual state. The interval timer on a real machine thus reflects system time and wait state time. In a virtual machine, the interval timer reflects only virtual processor time, and not wait time. It is updated by CP whenever a virtual machine passes control to CP, and this one updating reflects the entire time the virtual machine had control. Note that during the time a virtual machine has control, the virtual interval timer does not change; the virtual processor time used is added to the virtual interval timer when CP regains control. For some privileged instructions, CP may be able to simulate the instruction and still return control to the virtual machine before the end of that virtual machine's time slice. In such cases, the virtual interval timer is updated but only for those privileged instructions that require normal or fast reflect entry into the dispatcher. For those privileged instructions that do not require entry into the dispatcher, the virtual interval timer is not updated until CP gets control at the end of the time slice.

If the virtual machine assist feature or Extended Control - Program Support is ON, more time is charged to the virtual interval timer than if the feature is OFF. When the virtual machine assist feature is OFF, the time spent by CP to simulate privileged instructions is not charged to the virtual interval timer; whereas, with the feature ON, the time spent is charged to the virtual interval timer.

Virtual Interval Timer Assist

The virtual interval timer assist feature is the updating of the virtual interval timer and presentation of timer interrupts to the virtual machine by the hardware. When the software simulates the interval timer, updating occurs only when CP takes over control. This usually results in an update frequency of once per time slice and repeatability of timed intervals suffers greatly under these conditions. When the virtual interval timer assist feature is active, the update frequency is the same for both virtual and real interval timers, 300 times a second.

In order for the virtual interval timer assist feature to be active, the following conditions must be met:

- VM/370 must be running on a Model 135-3, 138, 145-3, or 148.
- The virtual machine must have enabled the virtual machine assist and the virtual interval timer (SET TIMER {ON|REAL}).
- The virtual machine must have enabled both the virtual machine assist and the virtual interval timer assist (SET ASSIST ON TMR).

VM/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 processor time and virtual wait time, but not CP time used for services for that virtual machine, such as privileged instruction execution. The more services a virtual machine requires from CP, the greater the difference between the time represented by the interval timer and the actual time used by and for the virtual machine. The larger the number of active virtual machines contending for system resources, the greater the difference between virtual machine time and actual elapsed (wall clock) time.

Processor Timer

A virtual machine must have the ECMODE directory option to use the System/370 processor timer.

The processor timer is supported in a virtual machine in much the same way as is the interval timer. That is, the processor timer in a virtual machine records only virtual processor time, and it is updated when the virtual machine passes control back to CP.

If the real timer option is specified, the processor timer reflects all actual elapsed time except CP time used for services, such as privileged instruction execution, for that virtual machine.

The method of sampling the value in the processor timer causes it to appear to a virtual machine to be updated more often than an interval timer. The privileged instructions Set processor Timer (SPT) and Store processor Timer (STPT) are used to set a doubleword value in the processor timer and to store it in a doubleword location of virtual storage. When a virtual machine samples the value in the processor timer by issuing a STPT instruction, CP regains control to execute the privileged instruction, and updates the time. The act of sampling the processor timer from a virtual machine causes it to be brought up to date.

TOD Clock

The System/370 time-of-day (TOD) clock does not require simulation in a virtual machine. The System/370 in which CP is operating may have one real TOD clock for each processor, and all virtual machines can interrogate the real TOD clock. The Store Clock (STCK) instruction is nonprivileged; any virtual machine can execute it to store the current value of the TOD clock in its virtual storage. The Set Clock (SCK) instruction, which is used to set the TOD Clock value, can be issued from a virtual machine, but CP always returns a condition code of zero and does not actually set the clock. Note that the TOD clock is the only true source of actual elapsed time information for a virtual machine. The base value for the TOD clock in VM/370 is 00:00:00 GMT, January 1, 1900.

In an attached processor environment, the TOD clocks are synchronized using the procedure described in the IBM System/370: Principles of Operation, GA22-7000.

Clock Comparator

The clock comparator associated with the TOD clock is used in virtual machines for generating interrupts based on actual elapsed time. The ECMODE option must be specified for a virtual machine to use the clock comparator feature. The Set Clock Comparator (SCKC) instruction specifies a doubleword value that is placed in the clock comparator. When the TOD clock passes that value, an interrupt is generated.

Pseudo Timer

The pseudo timer is a special VM/370 timing facility. It provides 24 or 32 bytes of time and date information in the format shown in Figure 22.

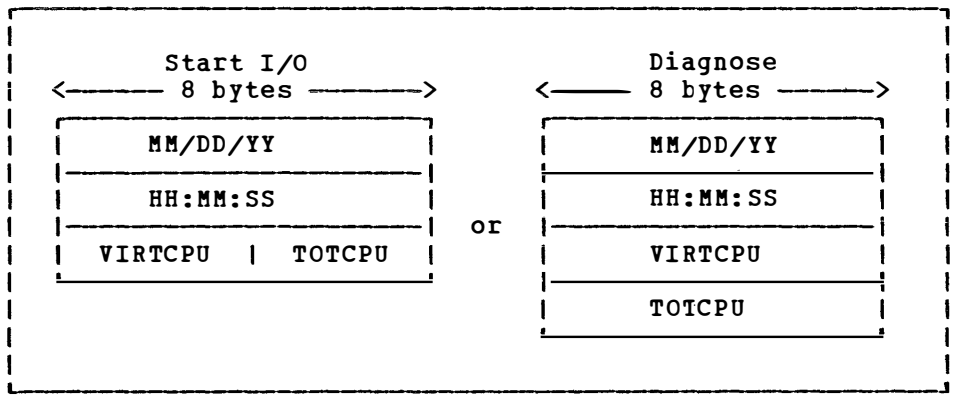


Figure 22. Formats of Pseudo Timer Information

The first eight-byte field is the date, in EBCDIC, in the form Month/Day-of-Month/Year. The next eight-byte field is the Time of Day in Hours:Minutes:Seconds. The VIRTCPU and TOTCPU fields contain virtual processor and total processor time used. The units in which the processor times are expressed and the length of the fields depend upon which of two methods is used for interrogating the pseudo timer.

PSEUDO TIMER START I/O

The pseudo timer can be interrogated by issuing a START I/O to the pseudo timer device, which is device type TIMER, and is usually at device address 0FF. No I/O interrupt is returned from the SIO. The address in virtual storage where the timer information is to be placed is specified in the data address portion of the CCW associated with the SIO. This address must not cross a page boundary in the user's address space. If this method is used, the virtual processor and the total processor times are expressed as fullwords in high resolution interval timer units. One unit is 13 microseconds.

PSEUDO TIMER DIAGNOSE

The pseudo timer can also be interrogated by issuing DIAGNOSE with an operation code of C, as described under "DIAGNOSE Instruction in a Virtual Machine." If this method is used, the virtual and total processor times are expressed as doublewords in microseconds.

CP in Attached Processor Mode

In an attached processor environment, two processors share main storage. There is special code in CP to ensure that the two processors do not interfere in each other's operation. Most of this special code is executed only in an attached processor environment. For information about system generation of the special code, see the VM/370 Planning and System Generation Guide.

PSA

Each processor needs its own area for processor-related information. During CP initialization, CP obtains an area from the high-end of real storage for prefix storage areas (PSA) for each processor. Each processor accesses its own PSA by a process called prefixing. Prefixing is described in detail in the System/370: Principles of Operation (GA22-7000).

When code executing on either processor references an address from 0 to 4096, the referenced address is added to the contents of the prefix register for that processor to produce the "absolute" address that will be accessed. A reference to the first 4K of storage, therefore, results in the PSA residing in high core being accessed. In this way, each processor is given its own work area and save areas. Figure 23, a storage map of the V=R machine, shows where PSAs are located in real storage after CP initialization completes. However, if a processor is varied offline and then online after CP initialization completes, the processor's PSA may be located in any contigeous 4K byte area of free storage.

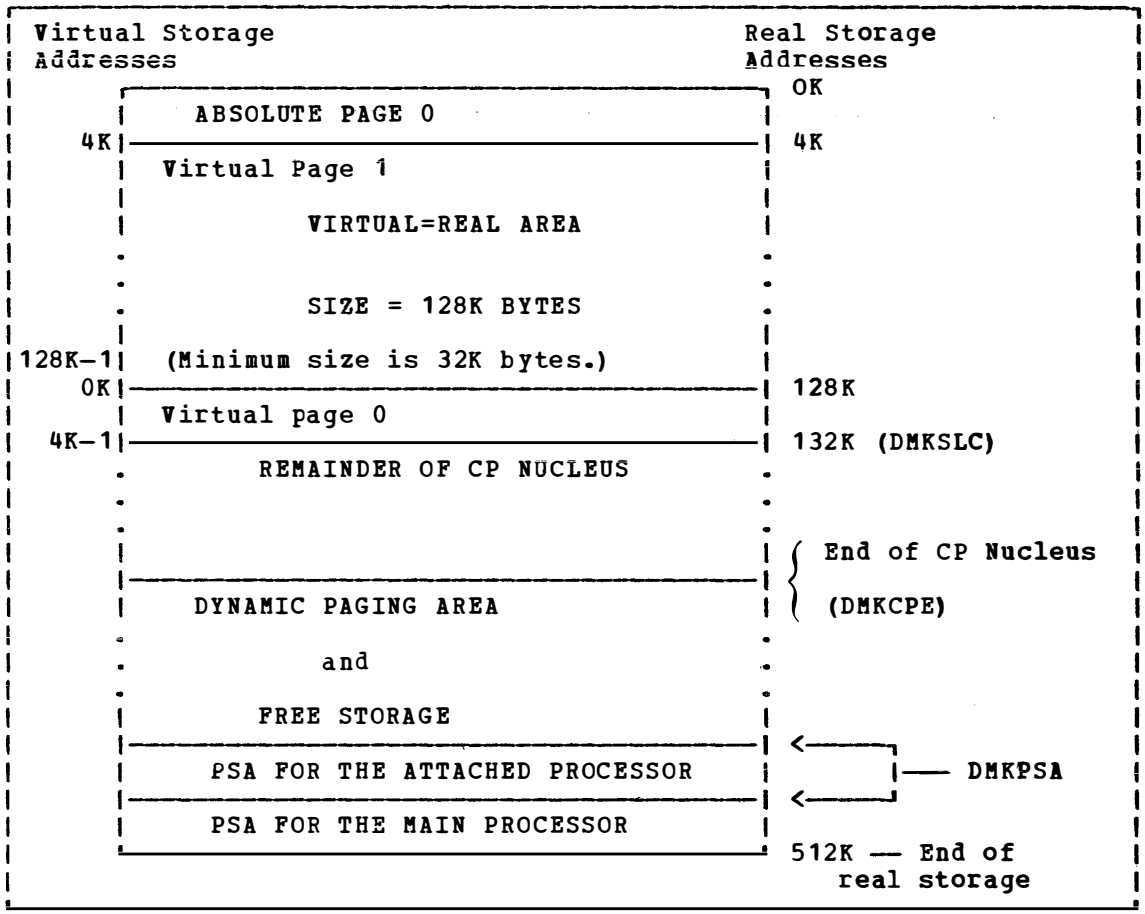


Figure 23. Storage in a Virtual=Real Machine

I/O HANDLING

In an attached processor environment, only the main processor is capable of handling I/O. If a command being executed on the attached processor requires I/O processing, it is dispatched to the main processor via the SWITCH macro. This is not apparent to the user as it is handled internally by CP.

When the control program is executing, the I/O configuration tables are protected only by the global system lock. Because the first-level interrupt handler (FLIH) executes without the lock in some cases, there is the possibility of both processors modifying an I/O table. To eliminate this situation, critical references to the I/O table are preceded by forcing the system onto the main processor via a SWITCH macro. Since the unlocked I/O FLIH never executes on the attached processor, the critical I/O code is serialized on the main processor.

SIGNALING

During certain critical periods, the locking mechanism alone is not enough. In these situations one processor must signal the other to request a specific action. As part of initialization, the main processor issues a SIGP to activate the attached processor. Some critical functions, such as machine check handling or extending free storage, require that one processor be quiesced for a time and then reactivated. This is done with the SIGNAL macro.

LOCKING

To provide system integrity, VM/370 attached processor support is designed around one global system lock, a VMBLOK local lock, and several system local locks for specifically identified queues or modules.

Global System Lock

All of CP runs under the global system lock except for specifically identified paths. If the lock cannot be obtained, the function will be deferred by storing the necessary information in the VMBLOK appendage and stacking that VMBLOK for later processing. That processor will then take a special unlocked path through the dispatcher to dispatch a new virtual machine.

In some situations, the processor cannot defer the requested function and will spin on the lock until it becomes available.

To ensure system integrity along the special unlocked paths, various local locks have been defined. These locks are basically spin locks and are held for short periods of time.

VMBLOK Lock

The VMBLOK lock is obtained by the dispatcher before dispatching a virtual machine in problem program state or before performing any system service for that virtual machine. This lock will prevent a virtual machine from being serviced by CP while it is running in problem program state.

Note: This lock is not a spin lock.

Free Storage Lock

The free storage lock is a spin lock obtained by DMKFRE for all FREE and FRET requests for free storage.

There are several other locks used by CP in situations where the global system lock is not held. All of the locks used by CP are described in detail in VM/370 System Logic and Problem Determination Guide.

User-Defined Locks

If you have user-defined areas that are used by more than one virtual machine, you will need to define your own locking conventions. You can use the LOCK macro to obtain and release a PRIVATE lock. The format of the LOCK macro is:

```
[label]  LOCK  {OBTAIN },TYPE=PRIVATE,[SPIN={YES}][,SAVE]
               {RELEASE}               {NO }
```

where:

label — is any desired use label.

OBTAIN
RELEASE — is a required positional operand indicating whether the lock is to be obtained or released.

TYPE=PRIVATE
is a required operand that indicates that the lock is a user lock.

SPIN=YES|NO
specifies whether control is to be returned without the lock being held. The default is SPIN=YES.

SAVE — is an optional keyword that indicates registers 0, 1, 14 and 15 are to be saved before the rest of the macro expansion. These are saved in the PSA of the processor that is executing this macro. The registers are restored before exit from the macro expansion.

The condition code (cc) is set as a result of the invocation of the LOCK macro.

cc=0 OBTAIN - lock obtained
RELEASE - lock released

cc=1 OBTAIN,SPIN=NO - lock owned by another processor.

A failure to release a lock results in a LOK003 abend.

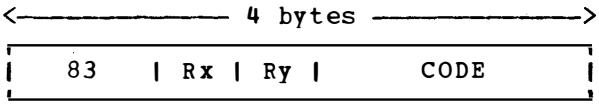
The address of the lockword must be specified in register 1 and the lockword must be a fullword aligned on a fullword boundary. Spin time for private locks is kept in the DMKLOKSI timer value for all non-DMKLOK locks.

AFFINITY

If one processor has a special feature that the other processor does not have, you can tell the system that you always want to execute on that processor. This is done by requesting affinity, either in the directory or with a SET AFFINITY command. See the VM/370 CP Command Reference for General Users for details on the SET AFFINITY command.

DIAGNOSE Instruction in a Virtual Machine

The DIAGNOSE instruction cannot be used in a virtual machine for its normal function. If a virtual machine attempts to execute a DIAGNOSE instruction, a program interrupt returns control to CP. Since a DIAGNOSE instruction issued in a virtual machine results only in returning control to CP and not in performing normal DIAGNOSE functions, the instruction is used for communication between a virtual machine and CP. The machine language format of DIAGNOSE is:



where:

83 is X'83' and interpreted by the assembler as the DIAGNOSE instruction.

Note: There is no mnemonic for DIAGNOSE.

Rx, Ry are general purpose registers that contain operand storage addresses or return codes passed to the DIAGNOSE interface. If the registers contain addresses, those addresses must be real to the virtual machine issuing the DIAGNOSE.

CODE is a two-byte hexadecimal value that CP uses to determine what DIAGNOSE function to perform. The codes defined for the general VM/SP user are described in this section. The code must be multiple of four. Codes X'00' through X'FC' are reserved for IBM use, and X'100' through X'1FC' are reserved for users.

Because DIAGNOSE operates differently in a virtual machine than it does in a real machine, a program should determine that it is operating in a virtual machine before issuing a diagnose instruction, and prevent execution of a DIAGNOSE when in a real machine. The Store Processor ID (STIDP) instruction provides a program with information about the processor in which it is executing, including the processor version number. If STIDP is issued from a virtual machine, the version number will be X'FF' in the first byte of the CPUID field.

A virtual machine issuing a diagnose instruction should run with interrupts disabled. This prevents loss of status information pertaining to the diagnose operation such as condition codes and sense data.

DIAGNOSE Code X'00' -- Store Extended-Identification Code

Execution of DIAGNOSE code X'00' allows a virtual machine to examine the VM/370 extended-identification code. For example, an OS/VS1 virtual machine issues a DIAGNOSE code X'00' instruction to determine if the version of VM/370 under which it is executing 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 executing does not support the DIAGNOSE code X'00' instruction, no data is returned to the virtual machine. If it does support the DIAGNOSE code X'00' instruction, the following data is returned to the virtual machine (at the location specified by Rx):

Field	Description	Characteristics
System Name	"VM/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 Processor 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 Processor Address) instruction to determine the processor address.	2 bytes, hexadecimal
Userid	The userid of the virtual machine issuing the DIAGNOSE.	8 bytes, EBCDIC
Program Product Bit Map	Bits that indicate the Program Products that are installed.	8 bytes, hexadecimal (Reserved for IBM use)

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 160 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 X'04' -- Examine Real Storage

Execution of a DIAGNOSE Code X'04' 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.

For each address in the list of CP addresses, VM/370 provides a fullword of data obtained from the specified address in real storage. VM/370 stores this data into the result field identified by the Ry+1 register.

There is a one-to-one correspondence between entries in the list of addresses and entries in the result field. For example, data obtained from the address in the first entry of the address list is stored in the first entry of the result field, data obtained from the second entry of the address list is stored in the second entry of the result field, and so forth.

Note: The request and result tables must be in the same page of virtual storage, and that page must be resident in real storage, at the time the DIAGNOSE is executed. This is guaranteed if the instruction itself is also in the same page.

In the attached processor environment, each processor has a prefix register to relocate addresses between 0 and 4095 to another page frame in main storage. The prefix register enables each processor to use a different page frame in order to avoid conflict with the other processor for such activity as interrupt code recording. Thus, the range 0 through 4095 refers to different areas of storage, depending upon which processor generates the address.

All references to main storage from either processor are handled as if they were made on the main processor. Existing user programs remain valid for performance data; in the attached processor environment, they receive the statistics for the main processor.

References to the PSA of the attached processor may be made as follows: first, retrieve the value of PREFIXB, the value of the prefix register for the other processor (the attached processor in this case). Next, specify addresses that are the sum of the value of PREFIXB and the PSA displacement. References to 0 through 4095 are made by summing the value of PREFIXA and the PSA displacement to form the request address. Several system values that are processor independent are maintained in 0 through 4095, such as the restart PSW and the trace table vectors.

DIAGNOSE Code X'08' -- Virtual Console Function

DIAGNOSE Code X'08' enables a virtual machine running in supervisor state to issue CP commands. The virtual machine must specify the command, the command parameters, and whether CP is to return the command response to the user's terminal or to a buffer. In addition to returning the command response, CP sets a completion code in the Ry register and may set a condition code.

When DIAGNOSE Code X'08' is issued, the Rx and Ry registers must be set up as follows:

Rx -- Rx must point to the character string in virtual storage that contains the CP commands and parameters. If the character string contains multiple commands, each command and its associated parameters must be separated from adjacent commands by the value X'15'.

Ry -- The high-order byte contains flag bits; the other three bytes specify, in bytes, the length of the CP commands and parameters. The maximum allowable length is 132 characters.

Set the flag bits as follows. If CP is to reject a password entered on the same line as a LINK command, set the high-order bit to a value of one (X'80'). CP rejects passwords only if the installation specified password suppression during system generation. If CP is to return the command response in a buffer, set the second flag bit to a value of one (X'40').

If the Ry register contains the value X'00000000', the DIAGNOSE Code acts as a no-operation (NOP) instruction.

If the command response is to be returned in a buffer, Rx and Ry cannot be consecutive registers nor can either be register 15. In addition, the Rx+1 and Ry+1 registers must be setup as follows:

Rx+1 -- Rx+1 must point to the buffer in virtual storage where CP is to return the command response.

Ry+1 -- Ry+1 must specify, in bytes, the length of the buffer. This value must not exceed 8192.

If the command response is to be returned in a buffer, CP sets a condition code and returns information as follows:

condition code 0 -- The request was successful. The Rx+1 register points to the buffer that contains the command response. The Ry+1 register specifies the length of the response.

condition code 1 -- The request was unsuccessful. The response does not fit into the buffer. The Ry+1 register contains a value that specifies how many bytes of the response would not fit into the buffer.

If an error is encountered while processing DIAGNOSE Code X'08', CP prints an error message and sets a completion code in the Ry register. The completion code is the hexadecimal representation of the numeric portion of the error message. For example, if error message DMKCFM045E is issued, CP sets a completion code of X'002D' which is the hexadecimal representation of 045.

If CP is executing multiple commands and encounters an invalid command, processing stops and CP ignores the remaining commands.

Following are two examples showing how to specify DIAGNOSE Code X'08'. The first example shows how a program issues the QUERY FILES command. In this example the response is returned to the user's terminal. Note that in a virtual storage (VS) environment, a load real address (LRA) instruction must be used to load the Rx register.

```
|           LA*     6,CMMD
            LA      10,CMMDL
            DC      X'83',X'6A',XL2'0008'
             .
             .
             .
CMMD        DC      C'QUERY FILES'
CMMDL       EQU     *-CMMD
              .
              .
```

The second example shows how to specify a string of commands when multiple commands are to be issued.

```
|           LA*     6,CMMD
            LA      10,CMMDL
            DC      X'83',X'6A',XL2'0008'
             .
             .
             .
CMMD        DC      C'QUERY FILES'
            DC      X'15'
            DC      C'PURGE PRINTER'
CMMDL       EQU     *-CMMD
```

| *Note that if you are in CMS mode you must code a LRA instruction
| instead of a LA instruction if you are running a virtual storage system
| (for example, MVS) in a virtual machine and want to specify the address
| of the CMMD parameter.

DIAGNOSE Code X'0C' -- Pseudo Timer

Execution of DIAGNOSE Code X'0C' 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 on a doubleword boundary. The information returned is in the format shown in Figure 22.

The first eight bytes contain the Month/Day-of-Month/Year. The next eight bytes contain the time of day in Hours:Minutes:Seconds. The last 16 bytes contain the virtual and total processor time used by the virtual machine that issued the DIAGNOSE. The last 16 bytes are expressed as a doubleword, unsigned integer. The time is expressed in microseconds. No completion code is returned, and the condition code remains unchanged.

DIAGNOSE Code X'10' -- Release Pages

Pages of virtual storage can be released by issuing a DIAGNOSE Code X'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 on 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.

Do not use DIAGNOSE Code X'10' to release noncontiguous storage: use DIAGNOSE Code X'64' for this purpose.

DIAGNOSE Code X'14' -- Input Spool File Manipulation

Execution of DIAGNOSE Code X'14' causes DMKDRDER to perform input spool file manipulation. Depending upon 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, which must be an even register, contains either the virtual address of a spool input card reader or, if Ry+1 contains X'0FFF', a spool file ID number. Ry+1 contains a hexadecimal code indicating the file manipulation to be performed. The codes are:

Code	Function
0000	Read next spool buffer (data record)
0004	Read next print spool file block (SFBLOK)
0008	Read next punch spool file block (SFBLOK)
000C	Select a file for processing
0010	Repeat active file nn times
0014	Restart active file at beginning
0018	Backspace one record
001C	Read next monitor spool file block
0020	Read next monitor spool record
0FFF	Retrieve subsequent file descriptor

On return Ry+1 may contain error codes that further define a returned condition code of 3.

Condition Code	Ry+1	Error
0		Data transfer successful
1		End of file
2		File not found
3	4	Device address invalid
3	8	Device type invalid
3	12	Device busy, reader not ready, or device is a real device
3	16	Fatal paging I/O error
3	20	Page already locked for I/O

SUBCODE X'0000'

Rx = start address of fullpage virtual buffer
Ry = virtual spool reader address

The specified device is checked for a file activated via DIAGNOSE. If one is found, the next fullpage buffer is made available to the virtual machine via a call to DMKRPAGT. If a file is not found, the chain of reader files is searched for a file for the calling user and connected to the virtual device for further reading. If no file is found, virtual condition code 2 is set. When the end of an active file is reached, the device status settings are tested for "spool continuous."' If not set, virtual condition code 1 is set, indicating end of file. If the device is set for continuous input, the active file is examined to determine whether or not it is a multiple-copy file. If it is, reading is restarted at the beginning of the file. If it is not, the file is closed via DMKVSPCR and the reader chain is searched for another input file. If no other file is found, virtual condition code 1 is set. A specific DIAGNOSE X'14' Subcode X'0000' must be issued to get the first spooled page again.

SUBCODE X'0004'

Rx = virtual address of a 13-doubleword buffer
Ry = virtual spool reader address

If the specified device is in use via diagnose, the VSPLCTL block is checked to see whether or not this is a repeated call for printer SFBLOKs. If it is, then the chain search continues from the point where the last SFBLOK was given to the virtual machine. In this case, cc = 1 is set when there are no more print files. If this is the first call for an SFBLOK, or if there have been intervening calls for file reading, the spool input chain is searched from the beginning, and cc=2 is set if no files are found.

Note: The virtual buffer specified via Rx must not cross a page boundary or a specification exception will result.

SUBCODE X'0008'

Rx = virtual address of a 13-doubleword buffer
Ry = virtual spool reader address

Processing for subcode X'0008' is the same as for subcode X'0004', except that only card-image input files are processed.

Note: For both subcode X'0004' and subcode X'0008', the format definition for a VM/370 SFBLOK can be found in the system macro library.

SUBCODE X'000C'

Rx = file identifier of requested file
Ry = virtual spool reader address

The spool input chain is searched for the file specified. If it is not found, cc=2 is set. If it is found, the file is moved to the head of the chain so that it will be the next file processed by any of the other functions.

SUBCODE X'0010'

Rx = new copy count for the active file
Ry = virtual spool reader address

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'0014'

Rx = start address of virtual fullpage 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.

SUBCODE X'0018'

Rx = start address of virtual fullpage 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 subcode X'0000'. If the file is already positioned at the first record, the first record is given to the user.

SUBCODE X'001C'

Rx = virtual address of a 13-doubleword buffer
Ry = virtual spool reader address

Processing is the same as Subcode X'0008', except that only monitor spool files, as identified by the SFBMON flag is SFBFLAG2, can be handled.

SUBCODE X'0020'

Rx = start address of fullpage virtual buffer
Ry = virtual spool reader address

Processing is the same as Subcode X'0000', except that only monitor spool files, as identified by the SFBMON flag in SFBFLAG2, can be handled.

SUBCODE X'0FFF'

Rx = virtual address of a 252-byte buffer
Ry = spool file ID number

If Ry is nonzero, the spool input chain is searched for a file with a matching ID number: If none is found or if one is found that is owned by a different virtual machine, cc=2 is set. The chain search is continued from the file that was found, or from the anchor if Ry is zero, for the next file owned by the caller, independent of file type, class, 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 X'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 Code X'18'. No I/O interrupts are returned by CP to the virtual machine; the DIAGNOSE instruction is completed only when the READ or WRITE commands associated with the DIAGNOSE are completed. 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 X'18' is used, as shown below. DIAGNOSE must not be used to read or write record-overflow-formatted data. 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 X'1C' -- Clear Error Recording Cylinders

Execution of DIAGNOSE Code X'1C' allows a user with privilege class F to clear the error recording data on disk. The DMKIOEFM routine performs the clear operation. The register specified as Rx contains a one-byte code value in the low-order byte as follows:

Code	Function
X'01'	Clear and reformat all error recording, leaving any frame records intact
X'02'	Clear and reformat all error recording cylinders, erasing both frame records and error records

DIAGNOSE Code X'20' -- General I/O

With DIAGNOSE Code X'20', a virtual machine user can specify any valid CCW chain to be performed on a tape or disk device. (An exception: DIAGNOSE must not be used to read or write record-overflow-formatted data on DASD devices.) No I/O interrupts are reflected to the virtual machine; the DIAGNOSE instruction is completed only when all I/O commands in the specified CCW chain are finished. The register specified as Rx contains the virtual device address. The Ry register contains the address of the CCW chain.

The CCWs are processed via DMKCCWTR through DMKGIOEX, providing full virtual I/O in a synchronous fashion (self-modifying CCWs are not permitted, however) to any virtual machine specified. Control returns to the virtual machine only after completion of the operation or detection of a fatal error condition. EREP support is provided for tape and DASD devices only; all other devices 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 error codes returned are as follows:

cc=0 I/O completed with no errors

cc=1 Error condition. Register 15 contains the following:

R15=1 Device is either not attached or the virtual channel is dedicated.
R15=5 Virtual device is busy or has an interrupt pending.

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 rightmost positions of the user's Ry register contain the first two sense bytes

DIAGNOSE Code X'24' -- Device Type and Features

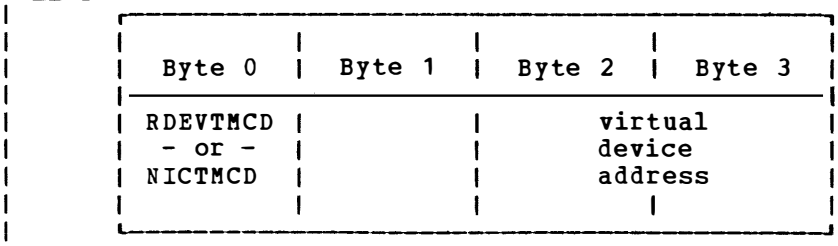
DIAGNOSE Code X'24' requests CP to provide a virtual machine with identifying information and status information about a specified virtual device. The virtual machine must specify the virtual device for which information is requested. CP returns information about the virtual device and associated real device in the Rx, Ry, and Ry+1 registers. CP also provides a condition code identifying the specific device information returned to the virtual machine.

When a virtual machine issues DIAGNOSE Code X'24', the Rx register must contain the virtual device address for which information is requested or the value negative 1 (-1). Specify -1 when the device is a virtual console whose address is unknown to the virtual machine.

When CP returns control to the virtual machine, the Ry, Ry+1, and Rx registers contain device information. The Ry register contains information about the virtual device and the Ry+1 register information about the real device. If -1 was specified and CP located the virtual console, the Rx register contains information about the virtual console.

CP obtains device information from three control blocks: virtual device information from the virtual device block (VDEVBLOK), and real device information from the real device block (RDEVBLOK) and from NICBLOK. The following diagrams identify specific information returned by CP and show how to locate this information in the Rx, Ry, and Ry+1 registers. The symbolic names used in these diagrams are the symbolic names used with VDEVBLOK, RDEVBLOK, and NICBLOK in VM/370 Data Areas and Control Block Logic.

Rx Register



Symbolic Name	Meaning
RDEVTMCD - or - NICTMCD	Terminal code bits defining the type of console and the translate table the console is using. RDEVTMCD is for a local virtual console; NICTMCD for a remote 3270 virtual console

By Register

Byte 0	Byte 1	Byte 2	Byte 3
VDEVTYPC	VDEVTYPE	VDEVSTAT	VDEVFLAG

Symbolic Name Meaning

VDEVTYPC Virtual device type class

VDEVTYPE Virtual device type

VDEVSTAT Virtual device status

VDEVFLAG Virtual device flags

By +1 Register

Byte 0	Byte 1	Byte 2	Byte 3
RDEVTYPC	RDEVTYPE - or - NICDTYPE	RDEVMDL - or - NICMDL	RDEVFTR - or - RDEVLLEN - or - NICLLEN

Symbolic Name Meaning

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 local virtual console

NICDTYPE Real device type for a remote 3270 virtual console

NICMDL Real device model number for a remote 3270 virtual console

NICLLEN Current device line length for a remote virtual console

The following chart lists the condition codes CP can return for DIAGNOSE Code X'24', the meaning of each condition code, and the registers where data is returned.

If the condition code equals	This register contains information: Rx[1]	Ry	Ry+1[2]	Comments
0	X	X	X	Normal completion
1				Undefined
2	X	X		The virtual device exists but is not associated with a real device
3				Invalid device address or the virtual device does not exist

[1] The Rx register contains information only when DIAGNOSE Code X'24' specifies a virtual console whose address in unknown.

[2] If Ry is register 15, CP returns only virtual device information: no information is returned in register Ry+1.

DIAGNOSE Code X'28' -- Channel Program Modification

DIAGNOSE Code X'28' allows a virtual machine to correctly execute some channel programs modified after the Start I/O (SIO) instruction is issued and before the input/output operation is completed. The channel command word (CCW) modifications allowed are:

- A Transfer in Channel (TIC) CCW modified to a No Operation (NOP) CCW
- A TIC CCW modified to point to a new list of CCWs
- A NOP modified to a TIC CCW

When a virtual machine modifies a TIC CCW, it is modifying a virtual channel program. CP has already translated that channel program and is waiting to execute the real CCWs. The DIAGNOSE instruction, with Code X'28', must be issued to inform CP of the change in the virtual channel program, so that CP can make the corresponding change to the real CCW before it is executed. In addition, when a NOP CCW is modified to point to a new list of CCWs, CP translates the new CCWs.

To be sure that the DIAGNOSE instruction is recognized in time to update the real CCW chain, the virtual machine issuing the DIAGNOSE instruction should have a high favored execution value and a low dispatching priority value. The CP SET command should be issued:

```
SET FAVORED xx

SET PRIORITY nn
```

where xx has a high numeric value and nn has a low numeric value. The virtual machine issuing the DIAGNOSE Code X'28' must be in the supervisor mode at the time it issues the DIAGNOSE instruction.

When DIAGNOSE Code X'28' is issued, the Rx register contains the address of the TIC or NOP CCW that was modified by the virtual machine. The Ry register contains the device address in bits 16 through 31. Rx and Ry cannot be the same register. The addresses specified in the Rx register, the new address in the modified TIC CCW, and the new CCW list to which the modified TIC CCW points must all be addresses that appear real to the virtual machine: CP knows these addresses are virtual, but the virtual machine thinks they are real.

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

cc=0 The real channel program was successfully modified; register 15 contains a zero.

cc=1 There was probably an error in issuing the DIAGNOSE instruction. Register 15 (R15) contains one of the following completion codes:

R15=1 The same register was specified for Rx and Ry.
R15=2 The device specified by the Ry register was not found.
R15=3 The address specified by the Rx register was not within the user's storage space.
R15=4 The address specified by the Rx register was not doubleword aligned.
R15=5 A CCW string corresponding to the device (Ry) and address (Rx) specified was not found.
R15=6 The CCW at the address specified by the Rx register is not a TIC nor a NOP, or the CCW in the channel program is not a TIC nor a NOP.
R15=7 The new address in the modified TIC CCW is not within the user's storage space.
R15=8 The new address in the modified TIC CCW is not doubleword aligned.

cc=2 The real channel program cannot be modified because a channel end or device end already occurred. Register 15 contains a 9. The virtual machine should restart the modified channel program.

DIAGNOSE Code X'2C' -- Return DASD Start of LOGREC

Execution of DIAGNOSE Code X'2C' allows a user with privilege class C, E, or F to find the location on the disk of the error recording area, the number of error recording cylinders, and the location of the first error record.

The register specified as Rx contains a one-byte code in the low-order byte, indicating the function to be performed:

X'01' - Return the DASD location of the start of the error recording area, and the number of error recording cylinders.

X'02' - Return the HDRSTART value (DASD location of first error record).

X'04' - Return indication of whether there are frame records on the error recording cylinders.

On return to the issuer of DIAGNOSE '2C':

If code '01' is specified: Register Rx will contain the DASD location (in VM/370 control program internal format) of the start of the error recording area. Ry contains, in the low-order halfword, the number of error recording cylinders.

If code '02' is specified: Register Rx will contain the DASD location of the first error record (in CCPD format). The value actually points to the last frame record written, or record 2 if no frame records present.

If code '04' is specified: Register Ry will contain a X'02' in the low-order byte if frame records are present on the error recording cylinders; X'00' if no frame records present.

Note: Codes '02' and '04' may both be specified (code '06') on invoking DIAGNOSE. Both an Rx and Ry value must be specified.

DIAGNOSE Code X'30' -- Read One Page of LOGREC Data

Execution of DIAGNOSE Code X'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	I/O error
3	Invalid cylinder, outside recording area

DIAGNOSE Code X'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 X'34' instruction. However, this Diagnose Code cannot read spool files that contain VMDUMP records -- use DIAGNOSE Code X'14' for this purpose. If a program attempts to use DIAGNOSE Code X'34' to read VMDUMP records, CP returns a condition code of 2. The register specified as Rx contains the virtual address of a page-size buffer to receive the data. The Ry register, which must not be register 15, contains the virtual address of the spool input card reader. Ry+1, on return, may contain error codes as follows:

Condition Code	Ry+1 Error Code	Meaning
0		Data transfer successful
1		End of file
2		File not found
3	4	Device address invalid
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 X'38' -- Read System Symbol Table

Execution of DIAGNOSE Code X'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 X'3C' -- VM/370 Directory

Execution of DIAGNOSE Code X'3C' allows a user to dynamically update the VM/370 directory. The register specified as Rx contains the first 4 bytes of the volume identification. The first two bytes of Ry contain the last 2 bytes of the volume identification. The routine DMKUDRDS dynamically updates the directory.

DIAGNOSE Code X'48' -- Issue SVC 76 From a Second Level VM/370 Virtual Machine

A second level VM/370 operating system issues SVC 76 using this DIAGNOSE. SVC 76 handles I/C error recording for virtual operating systems. For instance, a virtual machine issues SVC 76 to record data about hardware errors that occur on devices dedicated to it.

R1 is the Rx register. The Ry register is not used in this DIAGNOSE. R1 must contain either of two values:

X'04' indicates an SVC 76 request from a VM/370 virtual machine

X'08' indicates that a VM/370 virtual machine issued DIAGNOSE X'48'

CP checks first for the X'04' value. If it is present, CP sets VMSPMFLG in the virtual machine's VMBLOK to X'04' and processes the SVC 76 request on behalf of the virtual machine.

If R1 contains a X'08' value, CP sets VMSPMFLG in the virtual machine's VMBLOK to X'08'. It then reflects the SVC 76 back to the virtual machine. The virtual machine then handles its own error recording.

For more information on SVC 76 and I/O error recording procedures, refer to VM/370 OLTSEP and Error Recording Guide, GC20-1809.

DIAGNOSE Code X'4C' -- Generate Accounting Records 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 zeros, 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 ACNTBIOK 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 ACCTBIOK, 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.

This page left blank

DIAGNOSE Code X'50' -- Save the 370X Control Program Image

DIAGNOSE Code X'50' (Privilege class A, B, or C only) invokes the CP module DMKSNC to (1) validate the parameter list and (2) write the page-format image of the 370X control program to the appropriate system volume.

When a 370X control program load module is created, the CMS service program SAVENCP builds a communications controller list (CCPARM) of control information. It passes this information to CP via a DIAGNOSE Code X'50'.

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 X'54' -- Control the Function of the PA2 Function Key

DIAGNOSE Code X'54' controls the function of the PA2 function key. The PA2 function key can be used either to simulate an external interrupt to a virtual machine or to clear the output area of a display screen.

The function performed depends upon how Rx is specified when DIAGNOSE Code X'54' is issued. If Rx contains a nonzero value, the PA2 key simulates an external interrupt to the virtual machine. If Rx contains a value of zero, the PA2 key clears the output area of the display screen.

The external interrupt is simulated only when the display screen is in the VM READ, HOLD, or MORE status and the TERMINAL APL ON command has been issued.

DIAGNOSE Code X'58' -- 3270 Virtual Console Interface

Execution of DIAGNOSE Code X'58' allows a virtual machine to display large amounts of data on a 3270 in a very rapid fashion. It is possible to 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 through 31) the device address of the virtual console.

To specify the display CCW, use the following assembler language instructions:

```
DS 0D
DC X'19', AL3 (dataddr), AL1(flags), AL1(ctl), AL2(count)
```

where:

dataddr is the beginning address of the data to be displayed.

flags is the standard CCW flag field with the suppress incorrect length indication (SLI) 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 a two byte field indicating the number of bytes to be displayed. The 3278-2A display console can display a maximum of 1440 bytes; all other display consoles can display a maximum of 1760 bytes.

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.

Note: An I/O error occurs when the display screen is placed in MORE status and the PA2 key is pressed to allow screen display.

DIAGNOSE Code X'5C' -- Error Message Editing

Execution of DIAGNOSE Code X'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 performed depends upon the function subcode in the register specified as Ry.

Subcode		Function
X'00'	LOADSYS	-- Loads a named segment in shared mode
X'04'	LOADSYS	-- Loads a named segment in nonshared mode
X'08'	PURGESYS	-- Releases the named segment from virtual storage
X'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, on a doubleword boundary, left justified, and padded with blanks.

The LOADSYS Function

When the LOADSYS diagnose function is executed, CP finds the system name table entry for the segment and builds the necessary page and swap tables (two sets one for each processor, when running in attached processor mode). CP releases all the virtual pages of storage that are to contain the named segment and then loads the segment in those virtual pages. When the LOADSYS function is executed, CP expands the virtual machine size dynamically, if necessary. CP also expands the segment tables to match any expansion of virtual storage.

When LOADSYS executes successfully, the address of where the named segment was loaded is returned in the register specified as Rx. When the LOADSYS function loads a segment in shared mode, it resets instruction and branch tracing if either was active.

After a LOADSYS function executes, the storage occupied by the named segment is addressable by the virtual machine, even if that storage is beyond the storage defined for the virtual machine. However, any storage beyond that defined for the virtual machine and below that defined for the named segment is not addressable. Figure 24 shows the virtual storage that is addressable before and after the LOADSYS function executes.

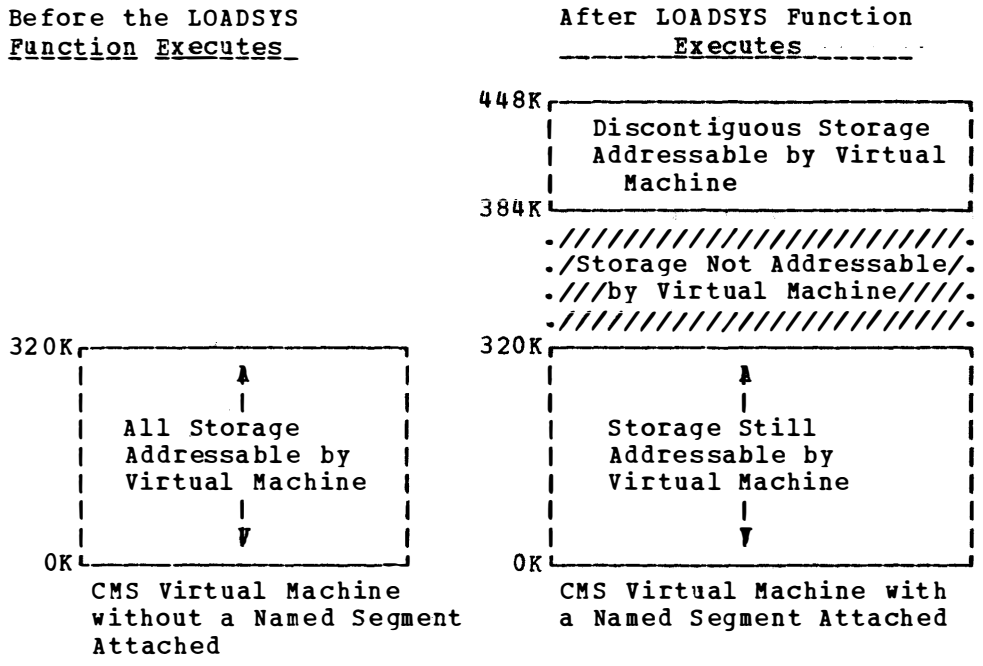


Figure 24. Addressable Storage Before and After a LOADSYS Function

When you save a named segment that is later loaded by the LOADSYS function, you must be sure that the addresses at which segments are saved are correct and that they do not overlay required areas of storage in the virtual machine. This is crucial because the LOADSYS function invokes the PURGESYS function before it builds the new page and swap tables. CP purges all saved systems that are overlayed in any way by the saved system it is loading.

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

DIAGNOSE Code X'68' -- Virtual Machine Communication Facility (VMCF)

The DIAGNOSE Code X'68' is used by a virtual machine to initiate a subfunction of the Virtual Machine Communication Facility (VMCF). The general register specified as Rx contains the virtual address, doubleword aligned, of a parameter list (VMCPARM). One of the entries in this parameter list is a subfunction code, specifying the particular request being initiated. The subfunctions and their codes are:

Subfunction	Code
AUTHORIZE	X'0000'
UNAUTHORIZE	X'0001'
SEND	X'0002'
SEND/RECV	X'0003'
SENDX	X'0004'
RECEIVE	X'0005'
CANCEL	X'0006'
REPLY	X'0007'
QUIESCE	X'0008'
RESUME	X'0009'
IDENTIFY	X'000A'
REJECT	X'000B'

A description of all the fields of the VMCPARM is contained in the section "Virtual Machine Communication Facility."

The general register specified as Ry will contain the return code upon completion of DIAGNOSE X'68' or the detection of an error condition. The return codes are contained in the section "Virtual Machine Communication Facility."

Rx and Ry can be any general register, R0 through R15. They may also be the same register.

DIAGNOSE Code X'74' -- Saving or Loading a 3800 Named System

DIAGNOSE Code X'74' is invoked to save an image library as a 3800 named system or to load a named system into virtual storage when that named system is required by the 3800 printer.

When the DIAGNOSE Code X'74' is invoked, the Rx, Rx+1, Ry, and Ry+1 registers must contain the following:

- Registers Rx and Rx+1 - must contain the eight-character name of the system to be saved or loaded, left-justified and padded with blanks.
- Register Ry - must contain the virtual address at which to start saving or loading the named system.
- Register Ry+1 - must contain a X'00' in the high order byte if a LOAD operation is required, and a X'04' for a SAVE operation. The remainder of the register must contain the number of bytes to be saved or loaded into virtual storage.

A specification exception occurs if Register 15 is specified in either Rx or Ry, or if the virtual address specified in Ry is not on a page boundary. If the area to be saved or loaded extends beyond the user's virtual storage, an addressing exception occurs. Finally, a privileged operation exception results if the user does not have privileged class A, B, or C. These exceptions cause abnormal termination (abend) and the user is notified.

When DIAGNOSE Code X'74' processing completes, one of the following condition codes is placed into register Ry and returned to CP:

Return Code	Meaning
X'00'	load/save successfully performed
X'04'	named system not found
X'08'	named system currently active

X'0C'	volid for system not CP owned
X'10'	volid for system not mounted
X'14'	too many bytes to load/save; residual byte count is in Ry+1
X'18'	paging error during load/save

DIAGNOSE Code X'78' -- MSS Communication

DIAGNOSE Code X'78' is used to communicate with the VM/370 control program about MSS volume mounts and demounts. The Ry register contains a subfunction code. The valid subfunction codes and their meanings are:

- X'00' - The virtual machine issuing the DIAGNOSE instruction is running OS/VS with MSS support and the DMKMSS program for MSS communication. The Rx register contains the device address of the virtual machine's MSS communicator virtual device.
- X'04' - The virtual machine is ready to process an MSS request. The MSSCOM block representing the request should be placed at the virtual machine address indicated by the Rx register.
- X'08' - An MSS request represented by the MSSCOM block located at the virtual machine address indicated by the Rx register has been accepted by the MSC.
- X'0C' - An MSS request represented by the MSSCOM block located at the virtual machine address indicated by the Rx register has been rejected by the MSC.
- X'10' - The DMKMSS program will no longer be available to process MSS requests.
- X'14' - The DMKMSS program has created a list of all VUAs associated with this processor (cpuid) and requests CP to build its shared and non-shared SDG tables from that list.

If the DIAGNOSE Code X'78' is specified incorrectly, CP terminates the user program with one of the following exceptions:

Error Return (DMKHVC)

Protection Exception - No DMKSSS module exists

Specification Exception - MSSCOM crosses a page

DIAGNOSE Code X'78' condition codes and return codes are:

Condition code = 0	Successful completion.
Condition code = 1	Error condition. Register 15 contains one of the following:
RC = 4	Subfunction code was either less than zero or greater than 16.
RC = 8	Subfunction code was within the valid range but not a multiple of 4.

RC = 12	Addressing Exception trying to bring in the buffer page
RC = 16	Issuer is not the issuer of subfunction zero

DIAGNOSE Code X'84' -- Directory Update In-Place

DIAGNOSE Code X'84' enables a class B user to replace certain data in any entry of the VM/370 directory. The user must specify the directory entry and may replace the following data:

- logon password
- virtual machine storage size
- maximum virtual machine storage size
- privilege classes
- dispatching priority
- logical editing symbols
- initial program load (IPL) system
- account number
- distribution word
- user options
- minidisk access mode
- minidisk read, write, or multiple password

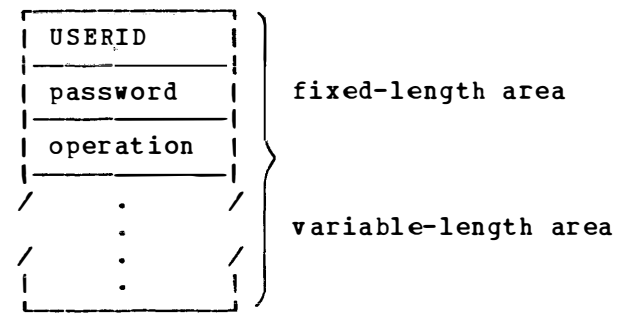
With the exception of the account number, all changes to the entry take effect the next time the USERID associated with the entry logs onto VM/370. The account number may be updated such that the change (1) takes effect immediately, (2) takes effect immediately but is temporary lasting only until the USERID is logged off, or (3) takes effect the next time the USERID associated with the entry is logged on.

DIAGNOSE Code X'84' cannot add new entries to the directory, cannot delete existing entries, nor can it alter directory user-description statements. It can only replace existing directory data. Data is replaced in the form of the directory created by the directory service program, that is, in VM/370 control blocks.

For a detailed description of the directory data, see the VM/370 Planning and System Generation Guide.

When DIAGNOSE Code X'84' is issued, the Rx register must point to a variable length parameter list and the Ry register must specify, in bytes, the length of the list. The parameter list contains an area of fixed length followed by an area of variable length. Data in the fixed-length area identifies the directory entry to be updated, the password of the USERID associated with the entry, and the data field to be replaced in the directory entry. The variable-length area contains replacement data for the directory entry. All entries in the parameter list must contain unpacked, EBCDIC data.

The parameter list is organized as follows:



Fixed-length area

USERID
The USERID of the user whose directory entry will be updated. This is an eight-character, left-justified value and must be padded with blanks.

password
The current password of the USERID whose directory entry will be updated. This is an eight-character, left-justified value and must be padded with blanks.

operation
An eight-byte, left-justified character string that identifies the data that is to be replaced in the directory entry. Valid values and the data that each identifies for replacement are defined in the description of the variable-length area which follows.

Variable-length area

The following diagram shows for each value of the operation field, (1) the data that must be in the variable-length area of the parameter list, and (2) the format and characteristics of the data.

operation field value	data	characteristics/format
LOGPASS	logon password	An eight-byte, left-justified value padded with blanks.
STORAGE	virtual machine storage size	An eight-byte, left-justified decimal value followed by the letter K. Pad with blanks following the letter K.
MAXSTOR	maximum virtual machine storage size	An eight-byte, left-justified decimal value followed by the letter K. Pad with blanks following the letter K.
PRIVLEGE	privilege classes	An eight-byte value where each byte represents a privilege class. Valid values for each byte are A through H. All existing classes in the directory entry are replaced. Therefore, specify existing classes that are to be retained as well as classes that are to be changed. The data must be left-justified and padded with blanks.
PRIORITY	dispatching priority	An eight-byte, left-justified value where the first two bytes, counting from the left, specify the dispatching priority. Valid values for these bytes are 1 - 99. Values 1 through 9 must be padded with a blank. The other six bytes are reserved for IBM use.
EDITCHAR	logical editing symbols	An eight-byte value where the first four bytes, counting from the left, are line edit symbols. The first or high-order byte is the "line-end" symbol, the second byte is the "line-delete" symbol, the third byte is the "character-delete" symbol, and the fourth byte is the "escape-character" symbol. All existing symbols in the directory are replaced. Therefore, specify existing symbols that are to be retained as well as symbols that are to be changed. Unspecified symbols must contain blanks. The last four bytes of the eight-byte value are reserved for IBM use.
IPL	system name or virtual device address	A one-to-eight character value, left-justified and padded with blanks.

Operation field value	data	characteristics/format
ACCOUNT	account number	A one-to-eight character value, left justified and padded with blanks. (This change takes effect the next time the USERID is logged on.)
IACCOUNT	account number	A one-to-eight character value left-justified and padded with blanks. (This change takes effect immediately.)
TACCOUNT	account number	A one-to-eight character value, left-justified and padded with blanks. (This change takes effect immediately but is temporary, lasting only until the USERID is logged off.)
DISTRIB	distribution identification word	A one-to-eight character value, left-justified and padded with blanks.
OPTIONS	user options	An eighty-byte, left-justified value padded with blanks. Specify each option as a character string with a blank character between options. For a description of each option and a list of valid values, see VM/370 Planning and System Generation Guide. All existing options are replaced in the directory entry. Therefore, specify existing options that are to be retained as well as options that are to be changed. The options field must be followed by the value X'FFFFFFFF'.
MDISK	minidisk address, access mode, read password, write password, and multiple password	A thirty-byte field defined as follows. All values must be left justified and padded with blanks. Valid values for the access mode and for passwords are defined in the VM/370 Planning and System Generation Guide. Bytes 1-3, counting from the left, specify a minidisk address. This is the minidisk whose mode and passwords will be changed. The address must already exist in the directory entry. Bytes 4-6 specify the access mode. Bytes 7-14 specify the read password.

Operation field value	data	characteristics/format
		Bytes 15-22 specify the write password. Bytes 23-30 specify the multiple password. The access mode, the read password, the write password, and the multiple password are replaced in the directory entry. Therefore, specify existing values that are to be retained as well as values that are to be changed.

Before control is returned to the virtual machine, DIAGNOSE Code X'84' sets a condition code and, if errors were detected, a return code in the Ry register. The condition codes and return codes are defined as follows:

condition code	meaning
0	The directory was successfully updated.
1	DIAGNOSE Code X'84' detected an error. The directory is unchanged. The return code defines the error.

return code	meaning
10,11	An error occurred writing the directory to a direct access device. To update the directory, use the directory service program described in the VM/370 Planning and System Generation Guide.
20 thru 25, 27, 90	DIAGNOSE Code X'84' encountered a processing error. To update the directory, use the directory service program described in the VM/370 Planning and JSystem Generation Guide.
26	Specified minidisk address does not exist in directory entry.
28	The value in the OPERATION field of the parameter list is invalid.
30	The specified USERID could not be found.
31	The password specified in the fixed-length area of the parameter list does not match the current password of the USERID being updated.
40,41	The value specified for the virtual machine storage size or for the maximum virtual machine storage size is too large. The maximum allowable size is 16 megabytes.
42, 43	The value specified for the virtual machine storage size or for the maximum virtual machine storage size contains a syntax error or an invalid character.

return code	meaning
50, 51	The specified privilege classes are invalid.
52, 53	The specified privilege classes contain a syntax error or an invalid character.
60, 61, 62	The specified priority contains a syntax error or an invalid character.
63	The priority value is too large. The maximum allowable value is 99.
65,66	Parameter list size error; if return code=65, the list exceeds 112 bytes; if return code=66, the list size is less than zero bytes long.
70	A specified option is invalid.
71	The value X'FFFFFFFF' was not coded after the list of options.
72	The option value contains a syntax error or an invalid character.
80	The parameter list contains an invalid minidisk address.
81	The parameter list specifies an invalid access mode for a minidisk.
82, 83	The minidisk read, write, or multiple password specified in the parameter list requires a change in the size of the directory entry.

CP Conventions

CP Coding Conventions

The following are coding conventions used by CP modules. This information should prove helpful if you debug, modify, or update CP.

1. FORMAT:

Column	Contents
1	Labels
10	Op Code
16	Operands
31, 36, 41, etc.	Comments (see Item 2)

2. COMMENT:

   Approximately 75 percent of the source code contains comments. Sections of code performing distinct functions are separated from each other by a comment section.

3. CONSTANTS:

   Constants follow the executable code and precede the copy files and/or macros that contain DSECTs or system equates. Constants are defined in a section followed by a section containing initialized working storage, followed by working storage. Each of these sections is identified by a comment. Wherever possible for a module that is greater than a page, constants and working storage are within the same page in which they are referenced.

4. No program modifies its own instructions during execution.

5. No program uses its own unlabeled instructions as data.

6. REGISTER USAGE:

   For CP, in general

Register	Use
6	RCHBLOK, VCHBLOK
7	RCUBLOK, VCUBLOK
8	RDEVBLOK, VDEVBLOK
10	IOBLOK
11	VMBLOK
12	Base register for modules called via SVC
13	SAVEAREA for modules called via SVC
14	Return linkage for modules called via BALR
15	Base address for modules called via BALR

   For Virtual-to-Real address translation

Register	Use
1	Virtual address
2	Real address

7. When describing an area of storage in mainline code, a copy file, or a macro, DSECT is issued containing DS instructions.

8. Meaningful names are used instead of self-defining terms: for example, 5,X'02',C'I' to represent a quantity (absolute address, displacement, length, register, etc.). All labels, displacements, and values are symbolic. All bits should be symbolic and defined by an equate (EQU) listing. For example:

```
VMSTATUS  EQU  X'02'
```

To set a bit, use:

```
OI  BYTE,BIT
```

where BYTE = name of field, BIT is an EQU symbol.

To reset a bit, use:

```
NI  BYTE,255-BIT
```

To set multiple bits, use:

```
OI  BYTE,BIT1+BIT2
```

etc. ...

All registers are referred to as:

R0, R1, ..., R15.

All lengths of fields or control blocks are symbolic, that is, length of VMBLOK is:

```
VMBLOKSZ  EQU  *-VMBLOK
```

9. Avoid absolute relative addressing in branches and data references, (that is, location counter value (*) or symbolic label plus or minus a self-defining term used to form a displacement).

10. When using a single operation to reference multiple values, specify each value referenced, for example:

```
LM  R2,R4,CONT  SET R2=CON1
                SET R3=CON2
                SET R4=CON3
.
.
.
CON1  DC  F'1'
CON2  DC  F'2'
CON3  DC  F'3'
```

11. Do not use PRINT NOGEN or PRINT OFF in source code.

12. MODULE NAMES:

    Control Section Names and External References are as follows:

    Control Section or Module Name

    The first three letters of the module name are the assigned component code.

        Example: DMK

    The next three letters of the module name identify the module and must be unique.

        Example: DSP

    This three-letter, unique module identifier is the label of the TITLE card.

    Each entry point or external reference must be prefixed by the six letter unique identifier of the module.

        Example: DMKDSPCH

13. TITLE CARD:

    DSP TITLE 'DMKDSP VM/370 DISPATCHER VERSION v LEVEL 1'

14. PTF Card

    Example:

    CP/CMS: PUNCH 'xxxxxxxx APPLIED'

    where:

    xxxxxxxx is the 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 several DC statements, and the insert positions must 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 Processor ID (STIDP) instruction. If STIDP is issued from a virtual machine, the version number (the first byte of the CPUID field) returned will be X'FF'.

CP Loadlist Requirements

The CP loadlist EXEC contains a list of CP modules used by the VMFLOAD procedures when punching the text decks that will make up the CP system. All modules following DMKCPE in the list are pageable CP modules. Each 4K page in this area may contain one or more modules. The module grouping is governed by the order in which they appear in the loadlist. An SPB[1] (Set Page Boundary) card, a loader control card placed in the text file, forces the loader to start this module at the next higher 4K boundary. The loader automatically moves a module to the next higher 4K boundary if it cannot fit in with its predecessors on the load list. In this case a message is placed on the load map:

"SPB INSERTED"

as part of the line

"**EXTERNAL SYMBOL DICTIONARY FOR DMKXXX"

An SPB card is required only for the first module following DMKCPE. If more than one module is to be contained in a 4K page, only the first can be assembled with an SPB card. The second and subsequent modules for a multiple module 4K page must not contain SPB cards.

The position of two modules in the loadlist is critical. All modules following DMKCPE must be reenterable and must not contain any address constants referring to anything in the pageable CP area. DMKCKP must be the last module in the loadlist.

The following modules are distributed with SPB cards:

DMKCDB	DMKSAV
DMKCKP	DMKSEV
DMKCPI	DMKSYM
DMKCPS	DMKTAP
DMKCPV	DMKVSI
DMKPGS	

---

[1] A 12-2-9 multipunch must be in column 1 of an SPB card and the characters SPB in columns 2, 3, and 4 respectively.

How to Add a Console Function to CP

Installations may add their own commands to their VM/370 system. First, code the module to handle the command processing. Follow the CP coding conventions outlined in an earlier section of this book.

Second, add an entry for the command in the CP DMKCFC module. DMKCFC has two entry points: one for logged-on users and another for nonlogged-on users. If the command is for logged-on users, be sure its entry is beyond the label COMNBEG1.

To place an entry for the command in the DMKCFC module, insert a line with the following format:

```
| [label] | COMND | commandname,class,min,entrypt[,NCL=1]
```

where:

commandname is a 1- to 8-character name.

class is the command privilege class (up to four classes are allowed). 0 is coded for nonlogged-on user commands or when NCL=1.

min is the number of characters allowed as the minimum truncation.

entrypt is the entry point of the module you write to process the new command.

NCL=1 is specified if the command is to be allowed before the user logs on. When NCL=1, the class is not checked.

After the above entry has been inserted in the DMKCFC module, reload DMKCFC as a pageable module ensuring that it does not cross a page boundary. You must also load your own module which may or may not be a resident module.

Print Buffers and Forms Control

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 and 3203 Printers

Name	Meaning
AN	Normal AN arrangement
HN	Normal HN arrangement
PCAN	Preferred character set, AN
PCHN	Preferred character set, HN
QN	PL/I - 60 graphics
QNC	PL/I - 60 graphics
RN	FORTRAN, COBOL commercial
YN	High speed alphanumeric
TN	Text printing 120 graphics
PN	PL/I - 60 graphics
SN	Text printing 84 graphics

- UCSB - for the 3203 and 3211 Printers

Name	Meaning
A11	Standard Commercial
H11	Standard Scientific
G11	ASCII
P11	PLI
T11	Text Printing

- FCB - for the 3203 and 3211 Printers

  There is only one name provided for an FCB image.

Name	Meaning
FCB1	Space 6 lines/inch 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

For the exact contents of the buffer images, see IBM 2821 Control Unit Component Description, and 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 that 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 that is used by the CP module DMKCSO. The format of the UCS macro is:

```
┌──────────────────────────────────────────────────────────────────────┐
│     │ UCS     │ ucsname                                              │
└──────────────────────────────────────────────────────────────────────┘
```

where:

ucsname is a 1- to 4-character name that is assigned to the buffer load.

Next, supply the exact print image. The print image is supplied by coding DCs in hexadecimal or character format. The print image may consist of several DCs, the total length of the print image cannot exceed 240 characters.

The UCSCCW macro must immediately follow the print image. This macro creates a CCW string to print the buffer load image when VER is specified by the operator on the LOADBUF command. The format of the UCSCCW macro is:

```
|      | UCSCCW | ucsname[,(print1,print2,...,print12)]                |
```

where:

ucsname is a 1- to 4-character name that is assigned to the buffer load by the UCS macro.

[(print1,...,print12)]
is the line length (or number of characters to be printed by the corresponding CCW) for the verify operation. Each count specified must be between 1 and 132 (the length of the print line on a 1403 printer) and the default line length is 48 characters. Up to 12 print fields may be specified. However, the total number of characters to be printed may not exceed 240.

Finally, insert the macros just coded, UCS and UCSCCW, into the DMKUCS module. This module must be reloaded. DMKUCS is a pageable module (with no executable code) 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 twenty-four 10-character entries. If, after DMKUCS is reloaded, the command

```
LOADBUF  00E  UCS  NUM1  VER
```

is specified, 4 lines of 60 characters (the 10-character string repeated 6 times) are printed to verify the buffer load).

Example 3: The print image can be specified in character or hexadecimal notation, or a combination of the two. The code in DMKUCS to support the preferred character set, AN, is as follows:

```
UCS     PCAN
DC      C'1234567890,-PQR#$@/STUVWXYZ',X'9C'
DC      C'.*1234567890,-JKLMNOABCDEFGHI+.*'
DC      C'1234567890,-PQR&&$%/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.

UCSB IMAGES

The Universal Character Set Buffer (UCSB) contains up to 512 characters and supports the 3211 printer. To add a new UCSB image, first code the UCB macro. This macro creates a 12-byte header record for the buffer load that is used by the CP module, DMKCSO. The format of the UCB macro is:

```
|     | UCB    | ucbname                                                |
```

where:

ucbname is a 1- to 4-character name that is assigned to the buffer load.

Next, supply the exact print image. The print image is supplied by coding DCs in hexadecimal or character notation. The total length of the print image cannot exceed 512 characters.

The format of the UCSB is:

Position	Contents
1-432	Print train image.
433-447	Reserved for IBM use. Must be all zeros.
448-511	Associative field. See Figure 25 for an explanation of the contents of this field. The associative field is used to check (during print line buffer (PLB) loading) that each character loaded into the PLB for printing also appears in the train image field of the UCSB and, therefore, is on the print train. Any character loaded into the PLB without its associated code in the train image field of the UCSB is nonprintable, and causes a "print data check" to be set immediately. The associative field also contains 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	¯	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 25. 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 1- to 4-character name that is assigned to the buffer load by the UCB macro.

[(print1,...,print12)]
specifies the line length of each line (up to 12) printed to verify the buffer load. The line length must be between 1 and 150 (the length of a print line on a 3211 printer). The default specification for verification is 48 characters per line for nine lines. The total number of characters to be printed must not exceed the size of the print train image, 432 characters.

Finally, insert the two macros just coded, UCB and UCBCCW, into the DMKUCB module. This module must be reloaded before the new buffer image can be used. DMKUCB is a pageable module (with no executable code) that is called by DMKCSO. DMKUCB must be on a page boundary and cannot exceed a full page in size.

Examples of UCSB Images

The code for the A11 UCSB is as follows:

```
          A11 STANDARD COMMERCIAL 48 GRAPHICS 3211
UCB       A11
DC        9C'1<.+IHGFEDCBA*$-RPQONMLKJ%,&&ZYXWVUTS/@#098765432'
DC        X'000000' 433-435
DC        X'00000000000000000000000001010110' 436-450
DC        X'10101010101010000404042400040110' 451-465
DC        X'10101010101010100004040410000040' 466-480
DC        X'40101010101010101000040400000000' 481-495
DC        X'10101010101010101010000404044480' 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 3203 or 3211 printer. A virtual 3203 or 3211 file can be printed on a real 1403; in fact, one way to provide forms control for a 1403 is to define it as a virtual 3211.

There is an FCB macro to support 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 that is to be the first printed position. (The "index" specification can be overridden with the LOADBUF command).

VM/370 provides one real FCB image, FCB1. This FCB is in pageable module DMKFCB. Installations may add additional FCB images to DMKFCB as long as the size of DMKFCB does not exceed the size of a page.

There is a default virtual FCB image located in resident storage (module DMKVSP). This FCB image is used by CP for a spooled virtual 3203 or 3211 printer if the user has not previously loaded an FCB for that virtual device.

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 3203 or 3211 printer. The LOADVFCB command is for the virtual printer and the LOADBUF command is for the real printer. 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 using the sum of the two specifications minus 1. For example, the command

```
LOADVFCB 00F FCB SPEC INDEX
```

indexes the virtual print file 10 positions because 10 was specified in the FCB macro for the SPEC forms control buffer. When this file is sent to the real printer, the 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).

Because the 3203 Model 4 and printers do not have indexing capabilities, the LOADVFCB and LOADBUF commands with the INDEX option causes a command reject error for the 3203 printer.

3203 Model 4 and 5 Printer Forms Control and Print Buffer

The Form Control Buffer for the 3203 is exactly like the 3211 Form Control Buffer. The 3203 uses the Universal Character Set (UCS) used by the 1403 Printer. The 3203 attaches a 64-byte associative field to the end of the UCS to check, during print line buffer (PLB) loading, that each character loaded into the PLB for printing also is on the print train. The 3203 associative field is exactly like the 3211 associative field described in Figure 25.

UCC BUFFER IMAGES

The UCC buffer contains up to 240 characters and supports the 1403 printer. To add a new UCC buffer image, first code the UCC macro. This creates a 12-byte header for the buffer load that is used by the CP module DMKCS0. The format of the UCC macro is:

```
┌──────────────────────────────────────────────────────────┐
│     │   UCC     │  uccname                               │
└──────────────────────────────────────────────────────────┘
```

where:

uccname is a 1- to 4-character name that is assigned to the buffer load.

Next, supply the exact print image. The print image is supplied by coding DCs in hexadecimal or character format. The print image may consist of several DCs, the total length of the print image cannot exceed 240 characters.

The UCCCCW macro must immediately follow the 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 UCCCCW macro is:

```
┌──────────────────────────────────────────────────────────┐
│     │  UCCCCW  │  uccname[,(print1,print2,...,print12)]  │
└──────────────────────────────────────────────────────────┘
```

where:

uccname is a 1- to 4-character name that is assigned to the buffer load by the UCC macro.

[(print1,...,print12)]
is the line length (or number of characters to be printed by the corresponding CCW) for the verify operation. Each count specified must be between 1 and 132 (the length of the print line on a 3203 printer) and the default line length is 48 characters. Up to 12 print fields may be specified. However, the total number of characters to be printed may not exceed 240.

Finally, insert the macros just coded, UCC and UCCCCW, into the DMKUCC module. This module must be reloaded. DMKUCC is a pageable module (with no executable code) that is called by DMKCS0. DMKUCC must be on a page boundary and cannot exceed a full page in size.

Examples of New UCC Buffer Images

Example 1: You do not have to specify the line length for verification of the buffer load. Insert the following code in DMKUCC:

```
      UCC     EX01
      DC      5CL'1234567890A...Z1234567890*/'
      UCCCCW  EX01
```

The buffer image is 5 representations of a 48-character string containing:

- The alphabetic characters
- The numeric digits, twice
- The special characters: * and /

Since the line length for the print verification is not specified on the UCCCCW macro, it defaults to 48 characters per line for 5 lines.

Example 2: Insert the following code in DMKUCC:

```
UCC      NUM1
DC      24CL'1234567890'
UCCCCW NUM1,(60,60,60,60)
```

The NUM1 print buffer consists of twenty-four 10-character entries. If, after DMKUCC is reloaded, the command

```
LOADBUF  00E  UCS  NUM1  VER
```

is specified, 4 lines of 60 characters (the 10-character string repeated 6 times) are printed to verify the buffer load).

IBM 3800 Printing Subsystem

The IBM 3800 Printing Subsystem is a high-speed, nonimpact printer that combines electrophotographic and laser technology. The 3800 printer can achieve speeds of up to 20,040 lines/minute, while several unique features give the user the ability to control the characteristics of printed output.

The features of the 3800 printer include:

- Forms control buffer - controls the amount of vertical space between printed lines. The user can specify vertical spacing of 6, 8, or 12 lines/inch.
- Multiple copy printing - allows the user to request multiple copies without the use of carbon paper. The 3800 uses its high speed to repeat-print the specified number of originals.
- Copy modification - allows the user to print or suppress predefined information on specified copies of a page. For example, a different name and address can be printed on each copy of a page.
- Forms overlay - allows the user to specify a form or grid to be printed (flashed) from a negative while output is being printed inside the form.
- Character arrangement tables - allow the user to specify which predefined character set will be used to print a data set. Each character set contains up to 64 printable characters.
- Character modification - allows character sets to be modified or extended to meet the user's needs.

For detailed information on the 3800 Printing Subsystem see:

- Introducting the IBM 3800 Printing Subsystem and Its Programming
- Concepts of the IBM 3800 Printing Subsystem
- IBM 3800 Printing Subsystem Programmer's Guide, OS/VS1, OS/VS2
- Reference Manual for the IBM 3800 Printing Subsystem

VM/370 supports the 3800 printer both as a dedicated device and as a real spooling device.

Using the 3800 Printer as a Dedicated Device

VM/370 allows a virtual machine that is configured to support a real 3800 to attach the 3800 for that machine's exclusive use. When used as a dedicated device, VM/370 supports all of the facilities of the 3800.

Using the 3800 Printer as a Real Spooling Device

VM/370 allows users of spool files to print their files on an IBM 3800 Printing Subsystem. The copy modification, forms overlay, character modification, and multiple copy features are fully supported. However, when the 3800 is used as a real spooling device, only one character arrangement table may be specified for each spool file. In addition, the entire spool file must be printed with the same line spacing on each page.

Parameters on five commands enable the user to take advantage of the 3800 printer's capabilities. The CHANGE, SPOOL, and START commands allow the user to specify the character arrangement table, copy modifications, forms overlay, and FCB to be used for printing. The BACKSPAC and QUERY commands also support the 3800 printer.

Two utilities, GENIMAGE and IMAGELIB, construct and modify the character arrangement tables, graphic modifications, copy modifications, and FCBs used by the 3800. DIAGNOSE Code X'74' is invoked by IMAGELIB to load and save this control information as a named system.

Finally, the NAME3800 macro instruction allows the user to create the named system that contains the control information needed to print a spool file.

SPECIFYING PRINTER OPTIONS

Five parameters on the SPOOL and CHANGE commands support the 3800 printer as a real spooling device. (See CP Command Reference for General Users for detailed coding information.)

- FLASH - identifies the form overlay, if any, to be used when printing the file
- CHARS - names the character arrangement table to be used to print the file
- MODIFY - indicates the copy modification module, if any, to be used when printing the file
- FCB - specifies the name of the forms control buffer to be used for the file
- COPY - indicates the number of copies to be printed

The START command includes parameters that enable the VM/370 operator to name the character arrangement table and FCB to be used for the separator page. The operator can also identify the forms overlay currently loaded in the 3800 via the FLASH operand of the START command. In addition, the operator uses the IMAGE parameter to specify the named system to be used to print the spool file. Finally, by specifying the PURGE parameter, the operator can purge all spool files that cause errors when loaded into the 3800. See VM/370 Operator's Guide for further information on the START command.

CREATING CONTROL TABLES

VM/370 uses the OS/VS utility IEBIMAGE to create and dynamically modify character arrangement tables, copy modifications, graphic modifications, and FCBs. Two service programs, GENIMAGE and IMAGELIB, interface with IEBIMAGE.

GENIMAGE creates or modifies text files on a CMS disk. These text files contain the images to be used by the 3800 Printing Subsystem.

IMAGELIB loads the new or changed text files created by GENIMAGE into virtual storage. When all the files are loaded, DIAGNOSE Code X'74' is invoked to save these files as a named system.

See VM/370 Planning and System Generation Guide or more information on IMAGELIB and GENIMAGE.

STORING AND LOADING CONTROL TABLES

As part of VM/370 support of the 3800 printer, character arrangement tables, copy modifications, graphic modifications, and FCBs are stored in a named system.

Prior to printing a spool file, the VM/370 operator specifies a named system on the IMAGE parameter of the START command. The control tables specified for the file (via the SPOOL and CHANGE commands) are loaded into the 3800 from that named system and the file is printed.

The NAME3800 macro instruction establishes the named system at system generation. See the VM/370 Planning and System Generation Guide for further information.

RECOVERING FROM I/O ERRORS

Because the actual printing of lines on the page is slower than the output of lines from the processor, spool files are placed into a delayed purge queue to await printing by the 3800. Only when the maximum number of files are in the queue will the first one actually be purged. The size of the queue can be specified at sysgen via the DPMSIZE parameter on the RDEVICE macro instruction. DPMSIZE can have a maximum value of nine.

Because spool files are queued, the BACKSPAC command may be used for the 3800 printer to restore pages that are lost when an I/O error occurs. In addition, the operator may specify the EOF parameter, which indicates that backspacing should begin at the end of the file and continue for the number of pages specified. See the VM/370 Operator's Guide for more information on the BACKSPAC command.

DISPLAYING PRINTER CONTROL INFORMATION

The QUERY command enables G-, B-, and D-privilege users to display the names of the character arrangement table, copy modification, and FCB currently in effect for a spool file or a virtual printer. In addition, the VM/370 operator can use the QUERY command to determine the image library used and the forms loaded on a real 3800.

See the VM/370 Operator's Guide for details on the QUERY command.

Journaling LOGON, AUTOLOG, and LINK Commands

LOGON, AUTOLOG, and LINK Journaling attempts to detect and record certain occurrences of the LOGON, AUTOLOG, or LINK commands. Using the recorded information, an installation may be able to identify attempts to logon to VM/370 by users that issue invalid passwords. Also, the installation may be able to identify users that successfully issue the LINK command to protected minidisks not owned by that user.

Briefly, LOGON, AUTOLOG, and LINK journaling works like this. While journaling is turned on, CP monitors all occurrences of the LOGON, AUTOLOG, and LINK commands. CP keeps count of the number of times a user issues one of these commands with an invalid password. When this count exceeds an installation defined threshold value, CP optionally:

- Writes a record to the accounting data set to record the incident
- Rejects subsequent LOGON, AUTOLOG, or LINK commands issued by the user
- Sends a message to an installation-defined user identification to alert the installation to the incident

Also, each time CP detects that a user has successfully issued a LINK command to a protected minidisk not owned by that user, CP optionally records the incident by writing a record to the accounting data set. A protected minidisk is a minidisk whose password is anything but ALL for the type of LINK attempted.

For a description of the accounting records that CP writes for LOGON, AUTOLOG, and LINK journaling, see the section "Accounting Records."

The SYSJRL macro instruction, the SET command, and the QUERY command enable an installation to control LOGON, AUTOLOG, and LINK journaling. To make journaling available and to specify options, code the SYSJRL macro instruction in module DMKSYS. Instructions for coding this macro instruction are in the VM/370 Planning and System Generation Guide. To turn journaling on or off, use the class A SET command. To determine whether journaling is on or off, use the class A QUERY command.

Suppressing Passwords Entered on the Command-Line

CP optionally rejects LOGON or LINK commands that have the password entered on the same line as the command. Rejecting these commands prevents passwords from being displayed or from being printed without masking -- masking a password means overprinting the password so it cannot be read.

This capability is also available to virtual machines that issue LINK commands via DIAGNOSE Code X'08'. For a description of DIAGNOSE Code X'08', see the section "DIAGNOSE Instruction in a Virtual Machine."

To request password suppression, specify it as an option on the SYSJRL macro instruction in module DMKSYS during system generation of VM/370. Once requested, password suppression is always on: an operator cannot turn it off.

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
- DOS/VS Support Under CMS
- CMS Support for OS and DOS VSAM Functions
- Saving the CMS system
- Batch Monitor
- Auxiliary Directories
- Assembler Virtual Storage Requirements

Introduction to CMS

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 that 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 Command and Macro Reference.

Figure 28 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 that may imply specific file characteristics to the CMS file management routines. The third is a filemode designator that describes the location and access mode of the file.

User files can be created directly from the terminal with the 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.

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.

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

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

Virtual disks may be shared by CMS users; the facility is provided by VM/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 that execute in CMS can create files on unlabeled tape in any record and block size; the record format can be fixed, variable, or undefined.

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. In order 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 that 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, see "OS Macro Simulation under CMS" and "DOS/VS Support under CMS."

Interrupt Handling in CMS

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

SVC Interruptions

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

INTERNAL LINKAGE SVCS

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

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

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

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

OTHER SVCs

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

In handling non-CMS SVC calls, DMSITS refers first to a user-defined SVC table (if one has been set up by the DMSHDS program). If the user-defined SVC table is present, any SVC number (other than 202 or 203) is looked for in that table. If it is found, control is transferred to the routine at the specified address.

If the SVC number is not found in the user-defined SVC table (or if the table is nonexistent), DMSITS either transfers control to the CMSDCS shared segment (if SETDOS ON has been issued), or the standard system table (contained in DMSSVT) of OS calls is searched for that SVC number. If the SVC number is found, control is transferred to the corresponding address in the usual manner. If the SVC is not in either table, then the supervisor call is treated as an abend call.

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

Input/Output Interruptions

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

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

If the wait bit is on, the interruption was probably caused by a nonterminal (synchronous) I/O operation. The program that initiated the operation most likely called the DMSIOW function routine to wait for a particular type of interruption (usually a device end). In this case, DMSITI checks the pseudo-wait bit in the device table entry for the interrupting device. If this bit is off, the system is waiting for some event other than the interruption from the interrupting device; DMSITI returns to the wait state by loading the saved I/O old PSW. (This PSW has the wait bit on.)

If the pseudo-wait bit is on, the system is waiting for an interruption from that particular device. If this interruption is not the one being waited for, DMSITI loads the saved I/O old PSW. This 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, HC, 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 interruption control area (PICA) into PIE.) DMSITP then determines whether or not the event that caused the interruption was one of those selected by a SPIE macro instruction. If it was not, DMSITP passes control to the DMSABN abend recovery routine.

If the cause of the interruption was one of those selected in a SPIE macro instruction, DMSITP picks up the exit routine address from the PICA and passes control to the exit routine. Upon return from the exit routine, DMSITP returns to the interrupted program by loading the original program check old PSW. The address field of the PSW was modified by a SPIE exit routine in the PIE.

External Interruptions

An external interruption causes control to be passed to the external interrupt handler DMSITE. If the user has issued the HNDEXT macro to trap external interrupts, DMSITE passes control to the user's exit routine. If the interrupt was caused by the timer, DMSITE resets the timer and types the BLIP character at the terminal. The standard BLIP timer setting is two seconds, and the standard BLIP character is uppercase, followed by the lowercase (it moves the typeball without printing). Otherwise, control is passed to the DEBUG routine.

Machine Check Interruptions

Hard machine check interruptions on the real processor are not reflected to a CMS virtual user by CP. A message prints on the console indicating the failure. The user is then disabled and must IPL CMS again in order to continue.

Functional Information

The most important thing to remember about CMS, from a debugging standpoint, is that it is a one-user system. The supervisor manages only one user and keeps track of only one user's file and storage chains. Thus, everything in a dump of a particular machine relates only to that virtual machine's activity.

You should be familiar with register usage, save area structuring, and control block relationships before attempting to debug or alter CMS.

Register Usage

When a CMS routine is called, R1 must point to a valid parameter list (PLIST) for that program. On return, R0 may or may not contain meaningful information (for example, on return from a call to FILEDEF with no change, R0 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 the following are in effect:

Register	Contents
1	The address of the PLIST supplied by the caller.
12	The address entry point of the called routine.
13	The address of a work area (12 doublewords) supplied by SVCINT.
14	The return address to the SVCINT routine.
15	The entry point (same as register 12).

On return from a routine, Register 15 contains:

Return Code	Meaning
0	No error occurred
<0	Called routine not found
>0	Error occurred

If a CMS routine is called by an SVC 202, registers 0 through 14 are saved and restored by CMS.

Most CMS routines use register 12 as a base register.

Structure of DMSNUC

DMSNUC is the portion of storage in a CMS virtual machine that contains system control blocks, flags, constants, and pointers.

The CSECTs in DMSNUC contain only symbolic references. This means that an update or modification to CMS, which changes a CSECT in DMSNUC, does not automatically force all CMS modules to be recompiled. Only those modules that refer to the area that was redefined must be recompiled.

USERSECT (USER AREA)

The USERSECT CSECT defines space that is not used by CMS. A modification or update to CMS can use the 18 fullwords defined for USERSECT. There is a pointer (AUSER) in the NUCON area to the user space.

DEVTAB (DEVICE TABLE)

The DEVTAB CSECT is a table describing the devices available for the CMS system. The table contains the following entries:

- 1 console
- 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 26 describes the devices supported by CMS.

Structure of CMS Storage

Figure 27 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, 3203, 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 26. 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 reentrant code for the CMS Nucleus routines. In shared CMS systems, this is the "protected segment," which must consist only of reentrant code, and may not be modified under any circumstances. Thus, such functions as DEBUG breakpoints or CP address stops cannot be placed in Segment 1 when it is a protected 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 that has been allocated from it by DMSFREE.

- Loader Tables (Top pages of storage). The top of storage is occupied by the loader tables, which are required by the CMS loader. These tables indicate which modules are currently loaded in the user program area (and the transient program area after a LOAD command). The size of the loader tables can be varied by the SET LDRTBLS command. However, to successfully change the size of the loader tables, the SET LDRTBLS command must be issued immediately after IPL.

Free Storage Management

Free storage can be allocated by issuing the GETMAIN or DMSFREE macros. Storage allocated by the GETMAIN macro is taken from the user program area, beginning after the high address of the user program.

Storage allocated by the DMSFREE macro can be taken from several areas.

If possible, DMSFREE requests are allocated from the low address free storage area. Otherwise, DMSFREE requests are satisfied from the storage above the user program area.

There are two types of DMSFREE requests for free storage: requests for USER storage and NUCLEUS storage. Because these two types of storage are kept in separate 4K pages, it is possible for storage of one type to be available in low storage, while no storage of the other type is available.

GETMAIN FREE STORAGE MANAGEMENT

All GETMAIN storage is allocated in the user program area, starting after the end of the user's actual program. Allocation begins at the location pointed to by the NUCON pointer MAINSTRT. The location MAINHIGH in NUCON is the "high extend" pointer for GETMAIN storage.

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 ││                                │
│          │          │ └          └    ┘┘                                │
└──────────────────────────────────────────────────────────────────────────┘
```

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 that are not allocated and that are, therefore, available for allocation by a GETMAIN instruction. These blocks are chained together, with the first one pointed to by the NUCON location MAINSTRT. Refer to Figure 27 for a description of CMS virtual storage usage.

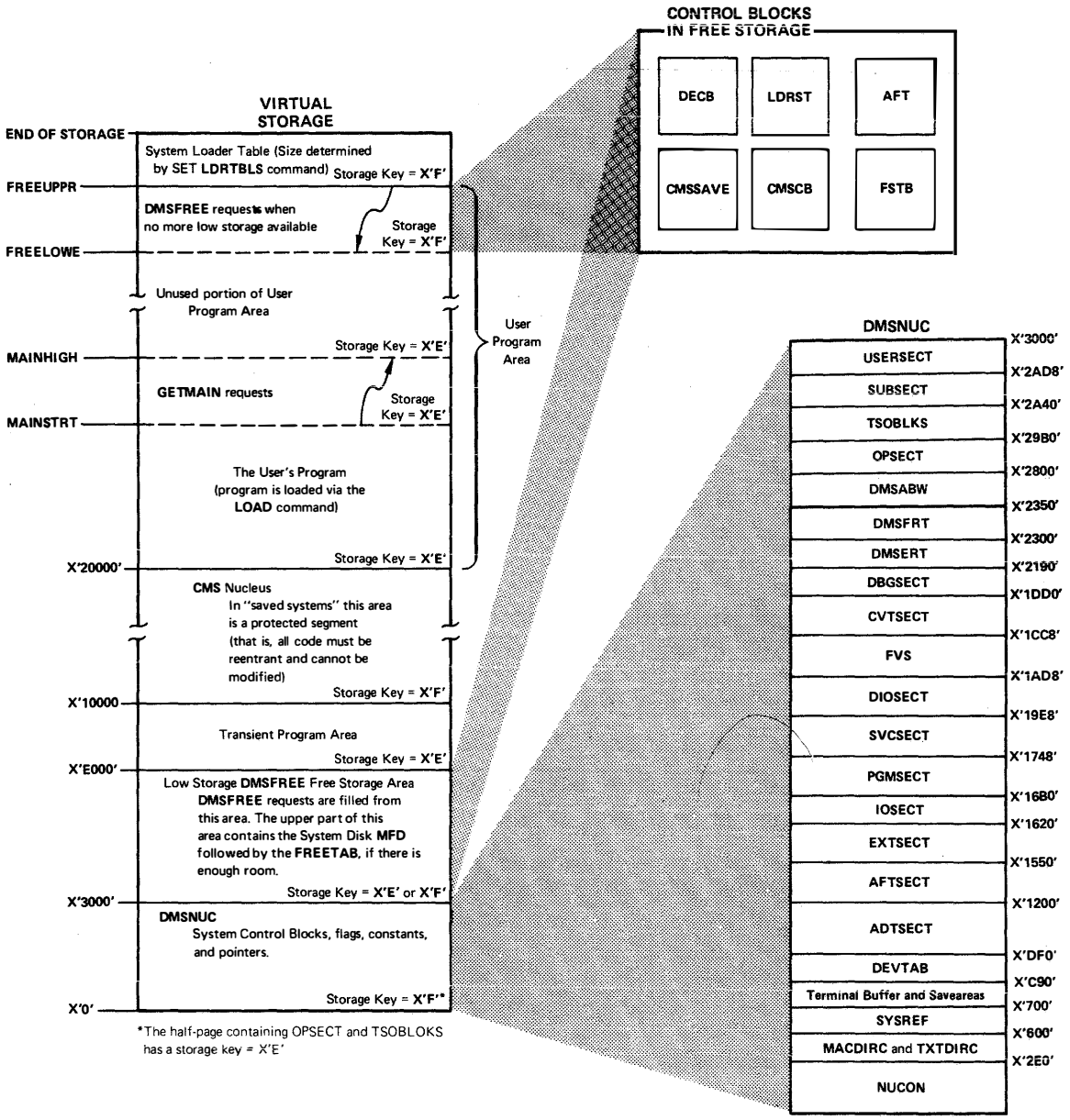
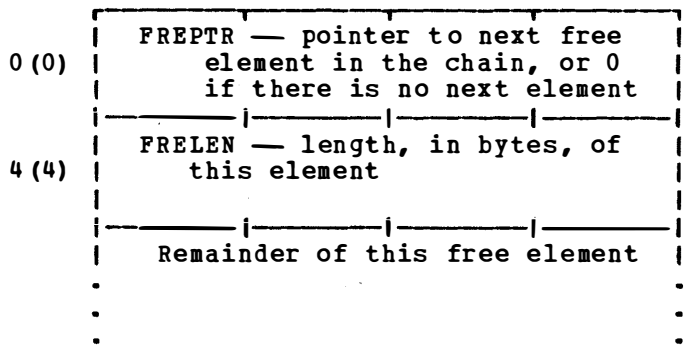


Figure 27. CMS Storage Map

The format of an element on the GETMAIN free element chain is as follows:

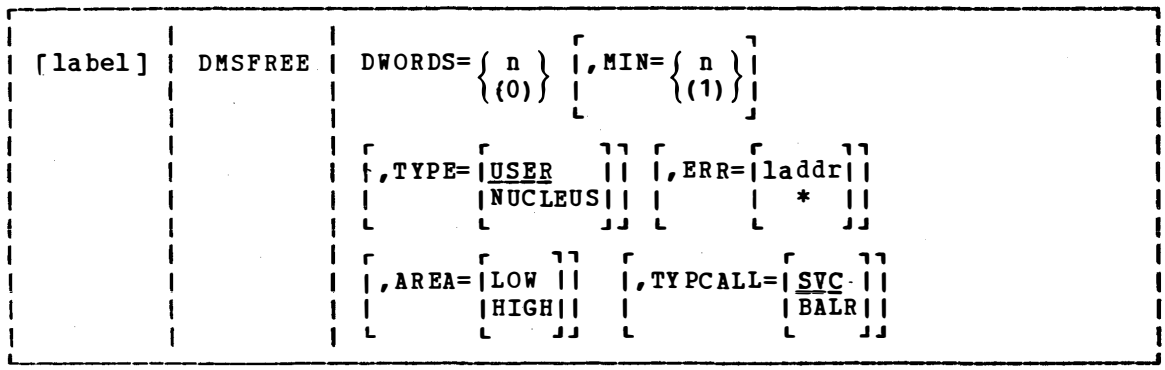


The amount of storage reserved for CMS use is determined at CMS IPL time based on the size of the virtual machine. The formula for calculating this reserved storage is:

Virtual Machine Size	Amount of Storage
Less than 512K	10 pages
512K	12 pages + 2 pages for each 256K in excess of 512K
Greater than 512K	12 pages + 2 pages for each 256K in excess of 512K

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. Do not specify any register other than register 0.

CMS returns, in register 0, the number of doublewords allocated and, in register 1, the address of the first byte of allocated storage.

MIN={n|(1)}
indicates a variable request for free storage. If the exact number of doublewords indicated by 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. Do not specify any register other than register 1.

TYPE=USER|NUCLEUS
indicates the type of CMS storage with which this request for free storage is filled: USER or NUCLEUS.

ERR=laddr|*
is the return address if any error occurs. "laddr" is any address that can be referred to in an LA (load address) instruction. The error return is taken if there is a macro coding error or if there is not enough free storage available to fill the request. If the asterisk (*) is specified for the return address, the error return is the same as a normal

return. There is no default for this operand. If it is omitted and an error occurs, the system will abend.

```
     ┌    ┐
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. Because DMSFREE is a nucleus-resident routine, other nucleus-resident routines can branch directly to it (TYPCALL=BALR) while routines that are not nucleus-resident must use linkage SVC (TYPCALL=SVC).

The pointers FREEUPPR and FREELOWE in NUCON indicate the amount of storage that DMSFREE has allocated from the high portion of the user program area. These pointers are initialized to the beginning of the loader tables.

The pointer FREELOWE is the "low extend" pointer of DMSFREE storage in the user program area. As storage is allocated from the user program area to satisfy DMSFREE requests, this pointer 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 storage is insufficient to satisfy the request. Figure 27 shows the relationship of these storage areas.

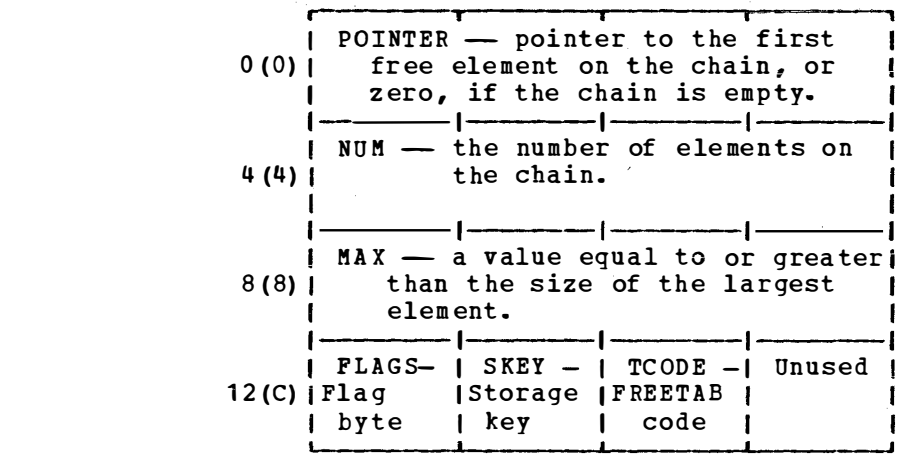
The FREETAB free storage table is kept in free storage, usually in low storage, just below the Master File Directory for the System Disk (S-disk). However, the FREETAB may be located at the top of the user program area. This table contains one byte for each page of virtual storage. Each such byte contains 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 that will fail. It contains a number equal to or greater than 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 to which FREELOWE points, then that page can be removed from the chain, and FREELOWE can be increased.
- All completely unallocated 4K pages are kept on the user chain, by convention. Thus, if one of the nucleus chains (low storage or high storage) contains a full page, then this page must be transferred to the corresponding user chain.

FLCLB (X'40') -- Destroyed flag. Set if the chain has been destroyed.

FLHC (X'20') -- High storage chain. Set for both the nucleus and user high-storage chains.

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:

```
[label] | DMSFRET | DWORDS={ n  },LOC={laddr}
        |         |        {(0) }     { (1) }
        |         | [,ERR=[laddr]] [,TYPCALL=[SVC ]]
        |         |       [  *  ]        [BALR]
```

where:

label — is any valid Assembler language label.

DWORDS={ n } / {(0)} — is the number of doublewords of storage to be released. DWORDS=n specifies the number of doublewords directly and DWORDS=(0) indicates that register 0 contains the number of doublewords being released. Do not specify any register other than register 0.

LOC={laddr} / { (1) } — is the address of the block of storage being released. "laddr" is any address that can be referred to in an LA (load address) instruction. LOC=laddr specifies the address directly while LOC=(1) indicates the address is in register 1. Do not specify any register other than register 1.

ERR=[laddr] / [*] — is the return address if 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 an asterisk (*) is specified, the error return address is the same as the normal return address. There is no default for this operand. If it is omitted and an error occurs, the system will abend.

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 that was never allocated.
- All requests that are satisfied in high storage must be of a temporary nature, since all storage allocated in high storage is released when the second free storage initialization routine is invoked.

When CP's saved system facility is used, the CMS system is saved at the point just after the A-Disk has been made accessible. It is necessary for DMSFRE to be used before the size of virtual storage is known, since the saved system can be used on any size virtual machine. Thus, the first initialization routine initializes DMSFRE so that limited functions can be requested, while the second initialization routine performs the initialization necessary to allow the full functions of DMSFRE to be exercised.

INIT2 — invokes the second initialization routine. This routine is invoked after the size of virtual storage is known, and it performs initialization necessary to allow all the functions of DMSFRE to be used. The second initialization routine performs the following steps:

- Releases all storage that has been allocated in the high storage area.
- Allocates the FREETAB free storage table. This table contains one byte for each 4K page of virtual storage, and so cannot be allocated until the size of virtual storage is known.
- The FREETAB table is initialized, and all storage protection keys are initialized.
- 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 that checks all free storage chains for consistency and correctness. Thus, it checks to see whether or not any free storage pointers have been destroyed. This option can be used at any time for system debugging.

CKON turns on a flag that causes the CHECK routine to be invoked each time a call is made to DMSFREE or DMSFRET. This can be useful for debugging purposes (for example, when you wish to identify the routine that destroyed free storage management pointers). Care should be taken when using this option, since the CHECK routine is coded to be thorough rather than efficient. Thus, after the CKCN option has been invoked, each call to DMSFREE or DMSFRET will take much longer to be completed than before.

CKOFF turns off the flag that was turned on by the CKON option.

UREC is used by DMSABN during the abend recovery process to release all user storage.

CALOC is used by DMSABN after the abend recovery process has been completed. It invokes a routine which returns, in register 0, the number of doublewords of free storage that have been allocated. This number is used by DMSAEN to determine whether or not the abend recovery has been successful.

TYPCALL=[SVC|BALR] indicates how control is passed to DMSFES. Since DMSFRES is a nucleus-resident routine, other nucleus-resident routines can branch directly to it, (TYPCALL=BALR) while routines that are not nucleus-resident must use SVC linkage (TYPCALL=SVC).

ERROR CODES FROM 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.

Code	Error
6	(DMSFRET) The block of storage that is being released was never allocated by DMSFREE. Such an error is detected if one of the following errors is found: • The block does not lie entirely inside either the low storage free storage area or the user program area between FREELOWE and FREEUPPR. • The block crosses a page boundary that separates a page allocated for USER storage from a page allocated for NUCLEUS type storage. • 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 are executed with a PSW key of X'E', while nucleus code is executed with a PSW key of X'0'.

There are, however, some exceptions to this rule. Certain disk-resident CMS commands run with a PSW key of X'0', since they have a constant need to modify nucleus pointers and storage. The nucleus routines called by the GET, PUT, READ, and WRITE macros run with a user PSW key of X'E', to increase efficiency.

Two macros 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 that 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 being executed with a user protect key, but wishes to execute a single instruction that, 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. Programs that modify or manipulate bits in DMSNUC or other CMS control blocks may, however, hinder the operation of CMS causing it to function ineffectively.

Whenever possible, CMS commands are executed with a user protect key. This protects the CMS Nucleus in cases where there is an error in the system command that would otherwise destroy the nucleus. If the command must execute a single instruction or small group of instructions that modify nucleus storage, then the DMSKEY or DMSEXS macros are used, so that the system PSW key will be used for as short a period of time as is possible.

CMS SVC Handling

DMSITS (INTSVC) is the CMS system SVC handling routine. The general operation of DMSITS is as follows:

1. The SVC new PSW (low storage location X'60') contains, in the address field, the address of DMSITS1. The DMSITS module 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 that originally made the SVC call).

SVC TYPES AND LINKAGE CONVENTIONS

SVC conventions are important to any discussion of CMS because the system is driven by SVCs (supervisor calls). SVCs 202 and 203 are the most common CMS SVCs.

SVC 202

SVC 202 is used 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 that is to be called, since DMSITS takes the absolute value of the code to determine the routine called.

Since only the second byte of the absolute value of the code is examined by DMSITS, seven bits (bits 1-7) are available as flags or for other uses. Thus, for example, DMSFREE uses these seven bits to indicate such things as conditional requests and variable requests.

When an SVC 203 is invoked, DMSITS stores the halfword code into the NUCON location CODE203, so that the called routine can examine the seven bits made available to it.

All calls made by means of SVC 203 should be made by macros, with the macro expansion computing and specifying the correct halfword code.

User-Handled SVCs

The programmer may use the HNDSVC macro to specify the address of a routine that will handle any SVC call other than for SVC 202 and SVC 203.

In this case, the linkage conventions are as required by the user-specified SVC-handling routine.

OS and 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 as it handles an error return from a legitimate SVC routine. The error code is -3.

3. Invalid SVC 203 code. If an invalid code follows SVC 203 inline, then an error message is displayed, and the abend routine is called to terminate execution.

SEARCH HIERARCHY FOR SVC 202

When a program issues SVC 202, passing a routine or command name in the parameter list, then DMSITS 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. 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. The system function name table is searched, to see if a command by this name is a nucleus-resident command. If the search is successful, control goes to the specified nucleus routine.

3. A search is then made for a disk file with the specified name as the filename, and MODULE as the filetype. The search is made in the standard disk search order. If this search is successful, then the specified module is loaded (via the LOADMOD command), and control passes to the storage location now occupied by the command.

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 of these searches fail, then DMSINT displays the error message UNKNOWN CP/CMS COMMAND.

See Figure 28 for a description of this search for a command name.

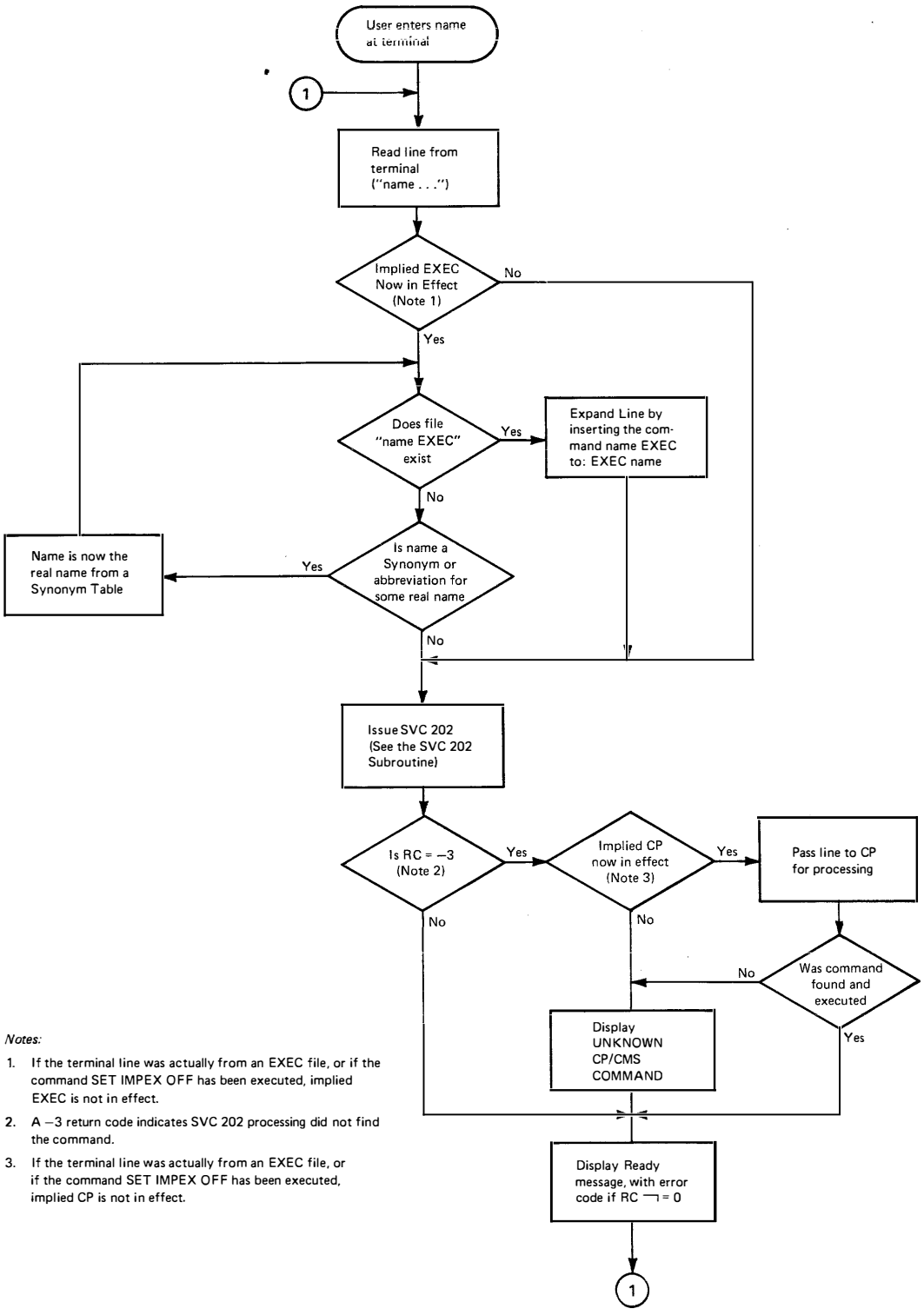


Figure 28. CMS Command (and Request) Processing (Part 1 of 2)

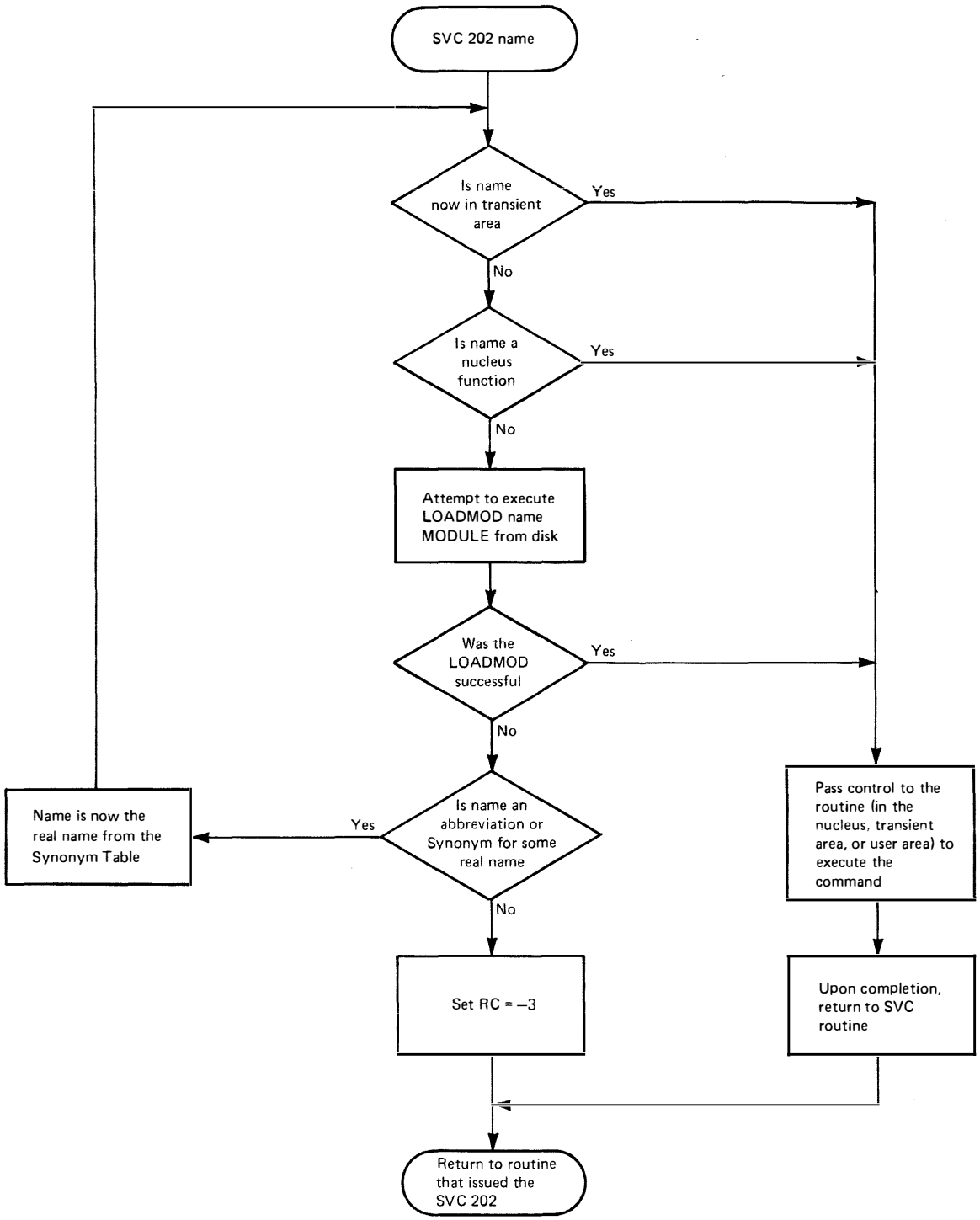


Figure 28. CMS Command (and Request) Processing (Part 2 of 2)

USER AND TRANSIENT PROGRAM AREAS

Two areas hold programs that are loaded from disk. These areas are called the user program area and the transient program area. (See Figure 27 for a description of CMS storage usage.)

The user program area starts at location X'20000' and extends upward to the loader tables. Generally, all user programs and certain system commands (such as EDIT, and COPYFILE) are executed in the user program area. Since only one program can be executing in the user program area at any one time, it is impossible (without unpredictable results) for one program being executed in the user program area to invoke, by means of SVC 202, a module that is also intended to be executed in the user program area.

The transient program area is two pages long, extending from location X'E000' to location X'FFFF'. It provides an area for system commands that may also be invoked from the user program area by means of an SVC 202 call. When a transient module is called by an SVC, it is normally executed with the PSW system mask disabled for I/O and external interrupts.

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 being executed in the transient program area may not invoke another program intended for execution in the transient program area, including OS macro simulation SVC calls that are handled by DMSSVT. For example, a program being executed 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 the programs to be executed in the user program area enabled for all interrupts but starts the programs to be executed in the transient program area disabled for all interrupts. The individual program may have to use the SSM (Set System Mask) instruction to change the current status of its system mask.

CALLED ROUTINE START-UP TABLE

Figures 29 and 30 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 29. 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 30. Register Contents When Called Routine Starts

RETURNING TO THE CALLING ROUTINE

When the called routine finishes processing, control is returned to DMSITS, which in turn returns control to the calling routine.

Return Location

The return is accomplished by loading the original SVC old PSW (which was saved at the time DMSITS was first entered), after possibly

modifying the address field. The address field modification depends upon the type of SVC call, and upon whether or not the called routine indicated an error return.

For SVC 202 and 203, the called routine indicates a normal return by placing a zero in register 15 and an error return by placing a nonzero code in register 15. If the called routine indicates a normal return, then DMSITS makes a normal return to the calling routine. If the called routine indicates an error return, DMSITS passes the error return to the calling routine, if one was specified, and abnormally terminates if none was specified.

For an SVC 202 not followed by "DC AL4(address)", 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 that was in register 15 when the called routine returned to DMSITS after it had completed processing.

Called Routine Modifications to System Area

If the called routine has system status, so that it runs with a PSW storage protect key of 0, then it may store new values into the System Save Area.

If the called routine wishes to modify the location to which control is to be returned, it must modify the following fields:

- For SVC 202 and 203, it must modify the NUMRET and ERRET (normal and error return address) fields.
- For other SVCs, it must modify the address field of OLDPSW.

To modify the registers that are to be returned to the calling routine, the fields EGPR1, EGPR2, ..., EGPR15 must be modified.

If this action is taken by the called routine, then the SVCTRACE facility may print misleading information, since SVCTRACE assumes that these fields are exactly as they were when DMSITS was first entered. Whenever an SVC call is made, DMSITS allocates two save areas for that particular SVC call. Save areas are allocated as needed. For each SVC call, a system and user save area are needed.

When the SVC-called routine returns, the save areas are not released, but are kept for the next SVC. At the completion of each command, all SVC save areas allocated by that command are released.

The System Save Area is used by DMSITS to save the value of the SVC old PSW at the time of the SVC call, the calling routine's registers at the time of the call, and any other necessary control information. Since SVC calls can be nested, there can be several of these save areas at one time. The system save area is allocated in protected free storage.

The user save area contains 12 doublewords (24 words), allocated in unprotected free storage. DMSITS does not use this area at all, but simply passes a pointer to this area (via register 13.) The called routine can use this area as a temporary work area, or as a register save area. There is one user save area for each system save area. The USAVEPTR field in the system save area points to the user save area.

The exact format of the system save area can be found in the VM/370 Data Areas and Control Block Logic. The most important fields, and their uses, are as follows:

Field	Usage
CALLER	(Fullword) The address of the SVC instruction that resulted in this call.
CALLEE	(Doubleword) Eight-byte symbolic name of the called routine. For OS and user-handled SVC calls, this field contains a character string of the form SVC nnn, where nnn is the SVC number in decimal.
CODE	(Halfword) For SVC 203, this field contains the halfword code following the SVC instruction line.
OLDPSW	(Doubleword) The SVC old PSW at the time that DMSITS was entered.
NRMRET	(Fullword) The address of the calling routine to which control is to be passed in the case of a normal return from the called routine.
ERRET	(Fullword) The address of the calling routine to which control is to be passed in the case of an error return from the called routine.
EGPRS	(16 Fullwords, separately labeled EGPR0, EGPR1, EGPR2, EGPR3, ..., EGPR15) The entry registers. The contents of the general registers at entry to DMSITS are stored in these fields.
EFPRS	(4 Doublewords, separately labeled EFPR0, EFPR2, EFPR4, EFPR6) The entry floating-point registers. The contents of the floating-point registers at entry to DMSITS are stored in these fields.

SSAVENXT (Fullword) The address of the next system save area in the chain. This points to the system save area that is being used, or will be used, for any SVC call nested in relation to the current one.

SSAVEPRV (Fullword) The address of the previous system save area in the chain. This points to the system save area for the SVC call in relation to which the current call is nested.

USAVEPTR (Fullword) Pointer to the user save area for this SVC call.

CMS Interface for Display Terminals

CMS has an interface that allows it to display large amounts of data in a very rapid fashion. This interface for 3270 display terminals (also 3138, 3148, and 3158) is much faster and has less overhead than the normal write because it displays up to 1760 characters in one operation, instead of issuing 22 individual writes of 80 characters each (that is one write per line on a display terminal). Data that is displayed in the screen output area with this interface is not placed in the console spool file.

The DISPW macro allows you to use this display terminal interface. It generates a calling sequence for the CMS display terminal interface module, DMSGIO. DMSGIO creates a channel program and issues a DIAGNOSE instruction (Code X'58') to display the data. DMSGIO is a TEXT file which must be loaded in order to use DISPW. The format of the CMS DISPW macro is:

```
┌──────────────────────────────────────────────────────────────────────┐
│           │          │        ┌        ┐  ┌           ┐              │
│ [label]   │  DISPW   │  bufad │,LINE=n│  │,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.

Note: It is advisable for the user to save registers before issuing the DISPW macro and to restore them after the macro, because neither the macro nor its called modules save the user's registers.

How to Add a Command or EXEC Procedure to CMS

You can create a module or EXEC procedure that executes in the user area and resides on disk. In order 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	

OS Macro Simulation under CMS

When a language processor or a user-written program is executing in the CMS environment and using OS-type functions, it is not executing OS code. Instead, CMS provides routines that simulate the OS functions required to support OS language processors and their generated object code.

CMS functionally simulates the OS macros in a way that presents equivalent results to programs executing under CMS. The OS macros are supported only to the extent stated in the publications for the supported language processors, and then only to the extent necessary to successfully satisfy the specific requirement of the supervisory function.

The restrictions for COBOL and PL/I program execution listed in "Executing a Program that Uses OS Macros" in the VM/370 Planning and System Generation Guide exist because of the limited CMS simulation of the OS macros.

Figure 31 shows the OS macro functions that are partially or completely simulated, as defined by SVC number.

OS Data Management Simulation

The disk format and data base organization of CMS are different from those of OS. A CMS file produced by an OS program running under CMS and written on a CMS disk, has a different format from that of an OS data set produced by the same OS program running under OS and written on an OS disk. The data is exactly the same, but its format is different. (An OS disk is one that has been formatted by an OS program, such as 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 in relation to preceding or following items in the data set.

Refer to Figure 31 and the "Simulation Notes," then read "Access Method Support" to see how CMS handles these access methods.

Since CMS does not simulate the indexed sequential access method (ISAM), no OS program that uses ISAM can execute under CMS. Therefore, no program can write an indexed sequential data set on a CMS disk.

HANDLING FILES THAT RESIDE ON OS OR DOS DISKS

By simulating OS macros, CMS can read, but not write or update, OS sequential and partitioned data sets that reside on OS disks. Using the same simulated OS macros, CMS can read 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 Name	SVC Number	Function
XDAP[1]	00	Read or write direct access volumes
WAIT	01	Wait for an I/O completion
POST	02	Post the I/O completion
EXIT/RETURN	03	Return from a called phase
GETMAIN	04	Conditionally acquire user storage
FREEMAIN	05	Release user-acquired storage
GETPOOL	–	Simulate as SVC 10
FREEPOOL	–	Simulate as SVC 10
LINK	06	Link control to another phase
XCTL	07	Delete, then link control to another load phase
LOAD	08	Read a phase into storage
DELETE	09	Delete a loaded phase
GETMAIN/ FREEMAIN	10	Manipulate user free storage
TIME[1]	11	Get the time of day
ABEND	13	Terminate processing
SPIE[1]	14	Allow processing program to handle program interrupts
RESTORE[1]	17	Effective NOP
BLDL/FIND[1]	18	Manipulate simulated partitioned data files
OPEN	19	Activate a data file
CLOSE	20	Deactivate a data file
STOW[1]	21	Manipulate partitioned directories
OPENJ	22	Activate a data file
TCLOSE	23	Temporarily deactivate a data file
DEVTYPE[1]	24	Obtain device-type physical characteristics
TRKBAL	25	NOP
FEOV	31	Set forced EOV error code
WTO/WTOR[1]	35	Communicate with the terminal
EXTRACT[1]	40	Effective NOP
IDENTIFY[1]	41	Add entry to loader table
ATTACH[1]	42	Effective LINK
CHAP[1]	44	Effective NOP
TTIMER[1]	46	Access or cancel timer
STIMER[1]	47	Set timer
DEQ[1]	48	Effective NOP
SNAP[1]	51	Dump specified areas of storage
ENQ[1]	56	Effective NOP
FREEDBUF	57	Release a free storage buffer
STAE	60	Allow processing program to decipher abend conditions
DETACH[1]	62	Effective NOP
CHKPT[1]	63	Effective NOP
RDJFCB[1]	64	Obtain information from FILEDEF command
SYNAD[1]	68	Handle data set error conditions
BSP[1]	69	Back up a record on a tape or disk
GET/PUT	–	Access system-blocked data
READ/WRITE	–	Access system-record data
NOTE/POINT	–	Manage data set positioning
CHECK	–	Verify READ/WRITE completion
TGET/TPUT	93	Read or write a terminal line
TCLEARQ	94	Clear terminal input queue
STAX	96	Create an attention exit block

[1]Simulated in the transient routine DMSSVT. Other simulation routines reside in the nucleus.

Figure 31. 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.

Due to the design of the CMS loader, an XCTL from the explicitly loaded phase, followed by a LINK by succeeding phases, may cause unpredictable results.

Listed below are descriptions of all the OS macro functions that are simulated by CMS as seen by the programmer. Implementation and program results that differ from those given in OS Data Management Macro Instructions and OS Supervisor Services and Macro Instructions are stated. HIARCHY options and those used only by OS multitasking systems are ignored by CMS. Validity checking is not performed within the simulation routines. The entry point name in LINK, XCTL, and LOAD (SVC 6, 7, 8) must be a member name or alias in a TXTLIB directory unless the COMPSWT is set to on. If the COMPSWT is on, SVC 6, 7, and 8 must specify a module name. This switch is turned on and off by using the COMPSWT macro. See the VM/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 upon whether this is an exit or return from a linked or an attached routine. Do not use EXIT/RETURN to exit from an explicitly loaded phase. If EXIT/RETURN is used for this purpose, CMS issues abend code A0A.
GETMAIN-SVC4	All options of GETMAIN are supported except SP and HIARCHY, which are ignored by CMS, and LC and LU, which will result in abnormal termination if used. GETMAIN gets blocks of free storage.
FREEMAIN-SVC5	All options of FREEMAIN are supported except SP, which is ignored by CMS, and L, which 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 register zero. However, if the specified entry point is not in core when SVC 8 is issued, and the subroutine contains VCONs that 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	CMS supports only the DEC parameter of the TIME macro instruction. However, the time value that CMS returns does not contain tenths of seconds or hundredths of seconds: the time is expressed as HHMMSS.
ABEND-SVC13	The completion code parameter is supported. The DUMP parameter is not. If a STAE request is outstanding, control is given to the proper STAE routine. If a STAE routine is not outstanding, a message indicating that an abend has occurred is printed on the terminal along with the completion code.
SPIE-SVC14	All the options of SPIE are supported. The SPIE routine specifies interruption exit routines and program interruption types that 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 TXTLIBs and MACLIBs, item numbers are filled in the TTR field of the BLDL list; the K, Z, and user data fields, as described in OS/VS Data Management Macro Instructions, are set to zeros. The "alias" bit of the C field is supported, and the remaining bits in the C field are set to zero.
FIND-SVC18	All the options of FIND are supported. FIND sets the read/write pointer to the item number of the specified member.

STOW-SVC21

All the options of STOW are supported. The "alias" bit is supported, but the user data field is not stored in the MACLIB directory since CMS MACLIBs do not contain user data fields.

OPEN/OPENJ-SVC19/22

All the options of OPEN and OPENJ are supported except for the DISP and RDBACK options, which are ignored. OPEN creates a CMSCB (if necessary), completes the DCB, and merges necessary fields of the DCB and CMSCB.

CLOSE/TCLOSE-SVC20/23

All the options of CLOSE and TCLOSE are supported except for the DISP option, which is ignored. The DCB is restored to its condition before OPEN. If the device type is disk, the file is closed. If the device type is tape, the REREAD option is treated as a REWIND.

DEVTYPE-SVC24

With the exception of the RPS option, which CMS ignores, CMS accepts all options of the DEVTYPE macro instruction. In supporting this macro instruction, CMS groups all devices of a particular type into the same class. For example, all printers are grouped into the printer class, all tape drives into the tape drive class, and so forth. In response to the DEVTYPE macro instruction, CMS provides the same device characteristics for all devices in a particular class. Thus, all devices in a particular class appear to be the same device type.

The device type characteristics CMS returns for each class are:

Class	Device Characteristics
Printer	1403
Card reader	2540
Console	1052
Tape drive	2400 (9 track)
DASD	2314
Card punch	2540
DUMMY	2314
unassigned	2314

FEOV-SVC31

Control is returned to CMS with an error code of 4 in register 15.

WTO/WTOR-SVC35

All options of WTO and WTOR are supported except those options concerned with multiple console support. WTO displays a message at the operator's console. WTOR displays a message at the operator's console, waits for a reply, moves the reply to the specified area, sets a completion bit in the specified ECB, and returns. There is no check made to determine if the operator provides a reply that is too long. The reply length parameter of the WTOR macro instruction specifies the maximum length of the reply. The WTCR macro instruction reads only this amount of data.

EXTRACT-SVC40

The EXTRACT routine in CMS is essentially a NOP. The user-provided answer area is set to zeros and control is returned to the user with a return code of 4 in register 15.

IDENTIFY-SVC41 The IDENTIFY routine in CMS adds a RPQUEST block to the load request chain for the requested name and address.

ATTACH-SVC42 All the options of ATTACH are supported in CMS as in OS PCP. The following options are ignored by CMS: DCB, LPMOD, DPMOD, HIARCHY, GSPV, GSPL, SHSPV, SHSPL, SZERO, PURGE, ASYNCH, and TASKLIB. ATTACH passes control to the routine specified, fills in an ECB completion bit if an ECB is specified, passes control to an exit routine if one is specified, and returns control to the instruction following the ATTACH.

Since CMS is not a multitasking system, a phase requested by the ATTACH macro must return to CMS.

CHAP-SVC44 The CHAP routine in CMS is a NOP. It returns control to the user.

TTIMER-SVC46 All the options of TTIMER are supported.

STIMER-SVC47 All options of STIMER are supported except for TASK and WAIT. The TASK option is treated as if the REAL option had been specified, and the WAIT option is treated as a NOP; it returns control to the user.

DEQ-SVC48 The DEQ routine in CMS is a NOP. It returns control to the user.

SNAP-SVC51 Except for SDATA, PDATA, and DCB, all options of the SNAP macro are processed normally. SDATA and PDATA are ignored. Processing for the DCB option is as follows. The DBC address specified with SNAP is used to verify that the file associated with the DCB is open. If it is not open, control is returned to the caller with a return code of 4. If the file is open, then storage is dumped (unless the FCB indicates a DUMMY device type). SNAP always dumps output to the printer. The dump contains the PSW, the registers, and the storage specified.

ENQ-SVC56 The ENQ routine in CMS is a NOP. It returns control to the user.

FREEDBUF-SVC57 All the options of FREEDBUF are supported. FREEDBUF returns a buffer to the buffer pool assigned to the specified DCB.

STAE-SVC60 All the options of STAE are supported except for the XCTL option, which is set to XCTL=YES; the PURGE option, which is set to HALT; and the ASYNCH option, which is set to NO. STAE creates, overlays, or cancels a STAE control block as requested. STAE retry is not supported.

DETACH-SVC62 The DETACH routine in CMS is a NOP. It returns control to the user.

CHKPT-SVC63 The CHKPT routine is a NOP. It returns control to the user.

RDJFCB-SVC64 All the options of RDJFCB are supported. RDJFCB causes a Job File Control Block (JFCB) to be read from a CMS Control Block (CMSCB) into real storage for each

data control block specified. CMSCBs are created by FILEDEF commands.

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 relative position of the last block read or written.

POINT — All the options of POINT are supported. POINT causes the control program to start processing the next read or write operation at the specified item number. The TTR field in the block address is used as an item number.

CHECK — All the options of CHECK are supported. CHECK tests the I/O operation for errors and exceptional conditions.

DCB — The following fields of a DCB may be specified, relative to the particular access method indicated:

Operand	BDAM	BPAM	BSAM	QSAM
BFALN	F,D	F,D	F,D	F,D
BLKSIZE	n(number)	n	n	n
BUFCB	a(address)	a	a	a
BUFL	n	n	n	n
BUFNO	n	n	n	n
DDNAME	s(symbol)	s	s	s
DSORG	DA	PO	PS	PS
EODAD	-	a	a	a
EXLST	a	a	a	a
KEYLEN[1]	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	-

---

[1]If an input data set is not a BDAM data set, zero is the only value that should be specified for KEYLEN. This applies to the user exit lists as well as to the DCB macro instruction.

ACCESS METHOD SUPPORT

The manipulation of data is governed by an access method. To facilitate the execution of OS Code under CMS, the processing program must see data as OS would present it. For instance, when the processors expect an access method to acquire input source cards sequentially, CMS invokes specially written routines that simulate the OS sequential access method and pass data to the processors in the format that the OS access methods would have produced. Therefore, data appears in storage as if it had been manipulated using an OS access method. For example, block descriptor words (BDW), buffer pool management, and variable records are updated in storage as if an OS access method had processed the data. The actual writing to and reading from the I/O device is handled by CMS file management. Note that the character string X'61FFFF61' is interpreted by CMS as an end of file indicator.

The essential work of the volume table of contents (VTOC) and the data set control block (DSCB) is done in CMS by a master file directory (MFD) which updates the disk contents, and a file status table (FST) (one for each data file). All disks are formatted in physical blocks of 800 bytes.

CMS continues to update the OS format, within its own format, on the auxiliary device, for files whose filemode number is 4. That is, the block and record descriptor words (BDW and RDW) are written along with the data. If a data set consists of blocked records, the data is written to, and read from, the I/O device in physical blocks, rather than logical records. CMS also simulates the specific methods of manipulating data sets.

To accomplish this simulation, CMS supports certain essential macros for the following access methods:

- BDAM (direct) -- identifying a record by a key or by its relative position within the data set.
- BPAM (partitioned) -- seeking a named member within data set.
- BSAM/QSAM (sequential) -- accessing a record in a sequence in relation to preceding or following records.
- VSAM (direct or sequential) -- accessing a record sequentially or directly by key or address.

  Note: CMS support of OS VSAM files is based on 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 (SVC 19) is issued for a data set. The CMS Control Block consists of a file control block (FCB) for the data file, and partial simulation of the job file control block (JFCB), input/output block (IOB), and data extent block (DEB).

The data control block (DCB) and the data event control block (DECB) are used by the access method simulation routines of CMS.

Note: The results may be unpredictable if two DCBs access the same data set at the same time.

The GET and PUT macros are not supported for use with spanned records except in GET locate mode. READ, WRITE, and GET (in locate mode) are supported for spanned records, provided the filemode number is 4, and the data set is physical sequential format.

GET (QSAM)
All the QSAM options of GET are supported. Substitute mode is handled the same as move mode. If the DCBRECFM is FB, the filemode number is 4, and the last block is a short block, an EOF indicator (X'61FFFF61') must be present in the last block after the last record.

GET (QISAM)
QISAM is not supported in CMS.

PUT (QSAM)
All the QSAM options of PUT are supported. Substitute mode is handled the same as move mode. If the DCBRECFM is FB, the filemode number is 4, and the last block is a short block, an EOF indicator is written in the last block after the last record.

When LOCATE mode is used with PUT, issue an explicit CLOSE prior to returning to CMS to obtain the last record.

PUT (QISAM)
QISAM is not supported in CMS.

PUTX
PUTX support is provided only for data sets opened for QSAM-UPDATE with simple buffering.

READ/WRITE (BISAM)
BISAM is not supported in CMS.

READ/WRITE (BSAM and BPAM)
All the BSAM and BPAM options of READ and WRITE are supported except for the SE option (read backwards).

READ (Offset Read of Keyed BDAM dataset)
This type of READ is not supported because it is used only for spanned records.

READ/WRITE (BDAM)
All the BDAM and BSAM (create) options of READ and WRITE are supported except for the R and RU options.

When an input or output error occurs, do not depend on OS sense bytes. An error code is supplied by CMS in the ECB in place of the sense bytes. These error codes differ for various types of devices and their meaning can be found in VM/370 System Messages, under DMS message 120S.

BDAM Restrictions

The four methods of accessing BDAM records are:

1. Relative Block RRR
2. Relative Track TTR
3. Relative Track and Key TTKey
4. Actual Address MBBCCHHR

The restrictions on these access methods are as follows:

- Only the BDAM identifiers underlined above can be used to refer to records, since CMS files have a two-byte record identifier.

- CMS BDAM files are always created with 255 records on the first logical track, and 256 records on all other logical tracks, regardless of the block size. If BDAM methods 2, 3, or 4 are used and the RECFM is U or V, the BDAM user must either write 255 records on the first track and 256 records on every track thereafter, or he must not update the track indicator until a NO SPACE FOUND message is returned on a write. For method 3 (WRITE ADD), this message occurs when no more dummy records can be found on a WRITE request. For methods 2 and 4, this will not occur, and the track indicator will be updated only when the record indicator reaches 256 and overflows into the track indicator.

- The user must create variable length BDAM files (in PL/1, they are Regional 3 files) entirely under CMS. He must also specify, on the XTENT option of the FILEDEF command, the exact number of records to be written. When reading variable length BDAM files, the XTENT and KEYLEN information specified for the file must duplicate the information specified when the file was created. CMS does not support WRITE ADD of variable length BDAM files; that is, the user cannot add additional records to the end of an already existing variable length BDAM file.

- Two files of the same filetype, both of which use keys, cannot be open at the same time. If a program that is updating keys does not close the file it is updating for some reason, such as a system failure or another IPL operation, the original keys for files that are not fixed format are saved in a temporary file with the same filetype and a filename of $KEYSAVE. To finish the update, run the program again.

- Once a file is created using keys, additions to the file must not be made without using keys and specifying the original length.

- Note that there is limited support from the CMS file system for BDAM-created files (sparse). Sparsed files will be manipulated with CMS commands but will not be treated as sparsed files by most CMS commands. The number of records in the FST will be treated as a valid record number.

- The number of records in the data set extent must be specified using the FILEDEF command. The default size is 50 records.

- The minimum LRECL for a CMS BDAM file with keys is eight bytes.

READING OS DATA SETS AND 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:

```
┌──────────────────────────────────────────────────────────────────────────┐
│         │                                                                │
│ FIledef │ ⎧ddname⎫ ⎛ ┌              ┌  ┐┐ ┌              ┐ ⎞            │
│         │ ⎨  nn  ⎬ ⎜ │DISK fn ft    │fm││ │DSN ?         │ ⎟            │
│         │ ⎩  *   ⎭ ⎜ │              │A1││ │DSN q1 [q2...]│ ⎟            │
│         │          ⎜ └              └  ┘┘ └              ┘ ⎟            │
│         │          ⎜      ┌              ┌  ┐┐                ⎟            │
│         │          ⎜DISK  │fn   ft       │fm││                ⎟            │
│         │          ⎜      │FILE ddname   │A1││                ⎟            │
│         │          ⎜      └              └  ┘┘                ⎟            │
│         │          ⎜                                           ⎟            │
│         │          ⎜DUMMY                                      ⎟            │
│         │                ┌                  ┐                              │
│         │ Related Option: │MEMBER membername│                              │
│         │                │CONCAT            │                              │
│         │                └                  ┘                              │
└──────────────────────────────────────────────────────────────────────────┘
```

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 ensure 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 Planning and System Generation Guide.

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 signaling whether or not I/O has been performed. If it has not been done, DMSSEB performs the appropriate device I/O.

When control is received from DMSSEB, the general-purpose registers contain the following information:

```
            GPR2  = Data Control Block (DCB) address
            GPR3  = Base register for DMSSEB
            GPR8  = CMS OPSECT address
            GPR11 = File Control Block (FCB) address
            GPR14 = Return address in DMSSEB
            GPR15 = Auxiliary processing routine address
all other registers = Work registers
```

The auxiliary processing routine must provide a save area in which to save the general registers; this routine must also perform the save operation. DMSSEB does not provide the address of a save area in general register 13, as is usually the case. When control returns to DMSSEB, the general registers must be restored to their original values. Control is returned to DMSSEB by branching to the address contained in general register 14.

GPR15 is used by the auxiliary processing routine to inform to DMSSEB of the action that has been or should be taken with the data block as follows:

Register	Content	Action
GPR15=0	No I/O performed by AUXPROC routine; DMSSEB will perform I/O.
GPR15<0	I/O performed by AUXPROC routine and error was encountered. DMSSEB will take error action.
GPR15>0	I/O performed by AUXPROC routine with residual count in GPR15; DMSSEB returns normally.
GPR15=64K	I/O performed by AUXPROC routine with zero residual count.

DOS /VS Support under CMS

CMS supports interactive program development for DOS/VS Release 31, 32, 33 and 34. 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 available in CMS/DOS only 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
- IBM 3340 Direct Access Storage Facility
- IBM 3344 Direct Access Storage
- IBM 3350 Direct Access Storage

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 Models 1 and 2, 3525, 5203, 5425P, and 5425S
- Disks: 2311

Also, CMS uses the CP spooling facilities and does not support dedicated unit record devices. Each CMS virtual machine supports only one virtual console, one reader, one punch, one printer, four tapes, and 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 32 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, and 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		Checks that a module generated to execute in a specific macro simulation environment (CMS/DOS or CMS) is in the correct environment.

Figure 32. 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.
	DOSLNCNT	Displays the current number of SYSLST lines per page.
	DOS	Displays the current status (active or not active) of CMS/DOS.
	DOSLIB	Displays the names of all CMS/DOS phase libraries currently being searched for executable phases.
	LIBRARY	Displays the names of all CMS/DOS phase libraries to be searched, in addition to the text and macro libraries.
RSERV		Executable only in the CMS/DOS environment. Copies and/or displays modules in a 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.
	DOSLNCNT nn	Specifies the number of SYSLST lines per page.
	UPSI	Executable only in the CMS/DOS environment. Sets the CMS/DOS UPSI byte.
SSERV		Executable only in the CMS/DOS environment. Copies or displays books from the DOS/VS source statement library. Output can also be directed to the virtual printer or punch.

Figure 32. Summary of Changes to CMS Commands to Support CMS/DOS (Part 2 of 2)

LOGICAL UNIT ASSIGNMENT

A logical unit is a symbolic name by which a program may refer to a real I/O device without knowing the device address. Two examples of logical units are SYSRDR and SYSPCH.

The DOS/VS supervisor uses two control blocks, the logical unit block (LUB) and the physical unit block (PUB), to map the symbolic name to the real device address. An entry in the LUB for a particular logical unit, such as SYSRDR, contains a pointer to a PUB entry. The PUB entry contains the address of the reader, X'00C'. Thus, all programs that read from the logical unit SYSRDR actually read from the device at address X'00C'.

On a real 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 NOP. When a logical unit is unassigned, its pointer to the PUB table is removed.

Compiler Input/Output Assignments

The compilers supported by CMS/DOS expect input/output to be assigned to the following devices:

- SYSIN/SYSIPT must be assigned to the device where the input source file resides. Valid device types are reader, tape, or disk.
- The user should assign the following logical units to any of the indicated device types:

  SYSPCH to tape, punch, disk, or IGN
  SYSLST to tape, printer, disk, or IGN
  SYSLOG to terminal
  SYS001, SYS002, and SYS006 to disk.
  SYS003-SYS005 to tape or disk.

The maximum number of work files is six for DOS/VS COBOL Compiler (FCOBOL) and two for DOS PL/I Optimizing Compiler (DOSPLI).

You must assign SYSIN/SYSIPT. If it is unassigned at compilation time, an error message is issued and the FCOBOL or DOSPLI command is terminated.

If SYSPCH or SYSLST are unassigned at compilation time, the FCOBOL or DOSPLI EXEC file directs output to the disk where SYSIN resides if SYSIN is assigned to a read/write CMS disk. Otherwise, output is directed to the CMS read/write disk with the most read/write space. If SYSLOG is unassigned, it is assigned to the terminal. If SYS001 through SYSnnn are unassigned, output is directed to the CMS disk with the most read/write space.

Interrogating I/O Assignments

The current I/O assignments may be displayed on the terminal by entering the CMS/DOS LISTIO command. You can selectively display the system and/or programmer logical units as a group or as a specific unit. With the EXEC option of the LISTIO command you can create a disk file containing the list of assignments.

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, 32, 33, and 34 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 33 lists the physical IOCS macros and describes their support. Figure 34 lists the control program function macros and their support. Refer to VM/370 System Logic and Problem Determination Guide, Volume 2 (CMS) for details of the macros' operation.

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 33. Physical IOCS Macros Supported by CMS/DOS

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 an error message.
REALAD	69	Not supported. Execution terminates with an 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 34. 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 to an AB routine has been established. 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 to an AB routine has been established. 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 an error message.
	(OC)	21	Not supported. Execution terminates with an error message.
	(AB)	95	Return from user's abnormal task termination routine.
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 an error message.

Figure 34. 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 the dump to the printer.
DUMP	--	Provides hexadecimal dump of the partition and general registers. CMS/DOS uses CP DUMP command to direct the 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 an error message.
Multitasking Macros		
ATTACH	38	Not supported. Execution terminates with an error message.
DETACH	39	Not supported. Execution terminates with an error message.
RCB	--	RCB control block generated. However, CMS/DOS does not support the use of Request Control Block.
DEQ	41	No operation performed.
ENQ	42	No operation performed.
WAITM	29	Not supported. Execution terminates with an 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 34. DOS/VS Macros Supported under CMS (Part 3 of 3)

SEQUENTIAL ACCESS METHOD -- DECLARATIVE MACROS

CMS/DOS supports the following declarative macros:

- DTFCD - Types X'02' and X'04'
- DTFCN - Type X'03'
- DTFDI - Type X'33'
- DTFMT - Types X'10', X'11', X'12', and X'14'
- DTFPR - Type X'08'
- DTFSD - Type X'20'

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 35 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 default.
EOFADDR=xxxxxxxx		Name of your end-of-file routine.
ERROPT=xxxxxx	*	IGNORE, SKIP, or name. Applies to 3505 and 3525 only.
FUNC=xxx	*	Not supported.
IOAREA2=xxxxxxxx	*	If two output areas are used, name of second area.
IOREG=(nn)		Register number if two I/O areas were used and GET or PUT does not specify a work area. Omit WORKA.
MODE=xx	*	Only MODE=E is supported.
MODNAME=xxxxxxxx		Name of the logic module that is used with the DTF table to process the file.
OUBLKSZ=nn	*	Not supported.
RDONLY=YES	*	Causes a read-only module to be generated.
RECFORM=xxxxxx		(FIXUNB, VARUNB, UNDEF). If omitted, FIXUNB is default.
RECSIZE=(nn)	*	Register number if RECFORM=UNDEF.

Figure 35. 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 35. 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 36 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 default.
INPSIZE=nnn		Length in bytes for input part of I/O area for PUTR macro usage.
MODNAME=xxxxxxxx		Logic module name for this DTF. If omitted, IOCS generates a standard name. The logic module is generated as part of the DTF.
RECFORM=xxxxxx		(FIXUNB or UNDEF). If omitted, FIXUNB is default.
RECSIZE=(nn)		Register number if RECFORM=UNDEF. General registers 2 through 12, enclosed in parentheses.
TYPEFLE=xxxxxx		(INPUT, OUTPUT, or CMBND). Input processes both input and output. CMBND must be specified for PUTR macro usage. If omitted, INPUT is default.
WORKA=YES		GET or PUT specifies work area.

Figure 36. CMS/DOS Support of DTFCN macro.

DTFDI MACRO - Define the File for Device Independence for System Logical Units

CMS/DOS supports most operands of the DTFDI macro. Figure 37 describes the operands of the DTFDI macro and their support under CMS/DOS. An asterisk in the status column indicates that CMS/DOS support differs from DOS/VS support.

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 on errors.
IOAREA2=xxxxxxxx		If two I/O areas are used, name of second area.
IOREG2=(nn)		Register number. If omitted and two I/O areas are used, register 2 is default. General registers 2 through 12, enclosed in parentheses.
MODNAME=xxxxxxxx		DIMOD name for this DTF. If omitted, IOCS generates a standard name.
RDONLY=YES		Generates a read-only module. Requires a module save area for each routine using the module.
RECSIZE=nnn		Number of characters in record. Default values: 121 (SYSLST), 81 (SYSPCH), 80 (other).
SEPASMB=YES		DTFDI to be assembled separately.
WLRERR=xxxxxxxx		Name of your wrong-length record routine.

Figure 37. 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 tape 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 at 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 38 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 supported only 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 registers 2 through 12, enclosed in parentheses.
LABADDR=xxxxxxxx	*	Name of your label routine if FILABL=NSTD, or if FILABL=STD and user-standard labels are processed. User label routines are supported only 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 38. 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 through 12, enclosed in parentheses.
REWIND=xxxxxx		(UNLOAD or NORWD). Unload on CLOSE or end-of-volume, or prevent rewinding. If omitted, rewind only.
SEPASMB=YES		DTFMT is to be assembled separately.
TPMARK=NO		Prevent writing a tapemark ahead of data records if FILABL=NSTD or NC.
TYPEFLE=xxxxxx		(INPUT, OUTPUT, or WORK). If omitted, INPUT is default.
VARBLD=(nn)		Register number, if RECFORM=VARBLK and records are built in the output area. General registers 2 through 12 are enclosed in parentheses.
WLRERR=xxxxxxxx		Name of wrong-length record routine.
WORKA=YES		GET or PUT specifies a work area. Omit IOREG.

Figure 38. 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 39 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 default.
CONTROL=YES		CNTRL macro used for this file. Omit CTLCHR for this file.
CTLCHR=xxx		(YES or ASA). Data records have control character. YES for S/370 character set; ASA for American National Standards Institute character set. Omit CONTROL for this file.
DEVICE=nnnn	*	(1403, 1443, 3203, or 3211). If omitted, 1403 is default.
ERROPT=xxxxxxxx	*	RETRY or the name of your error routine for 3211. Not allowed for other devices. IGNORE is not supported.
FUNC=xxxx	*	Not supported.
IOAREA2=xxxxxxxx		If two output areas are used, name of second area.
IOREG=(nn)		Register number; if two output areas used and GET or PUT does not specify a work area. Omit WORKA.
MODNAME=xxxxxxxx		Name of PRMOD logic module for this DTF. If omitted, IOCS generates standard name.
PRINTOV=YES		PRTOV macro used for this file.
RDONLY=YES		Generate a read-only module. Requires a module save area for each routine using the module.
RECFORM=xxxxxx		(FIXUNB, VARUNB, or UNDEF). If omitted, FIXUNB is default.
RECSIZE=(nn)		Register number if RECFORM=UNDEF.
SEPASMB=YES		DTFPR is to be assembled separately.
STLIST=YES		Use 1403 selective tape listing feature.
UCS=xxx		(ON) process data checks. (OFF) ignores data checks. Only for printers with the UCS feature or 3203 or 3211. If omitted, OFF is default.
WORKA=YES		PUT specifies work area. Omit IOREG.

Figure 39. 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 40 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. 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, 3330-11, 3340, 3350). If omitted, 2311 is the default used at compilation time. At execution time, when 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 to which the logical unit is assigned. 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. You must specify the DEVICE=nnnn operand correctly for input files residing on DOS disks; otherwise, CMS/DOS issues an error.
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.

Figure 40. CMS/DOS Support of DTFSD Macro (Part 1 of 3)

Operand	Status	Description
		message.
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. 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. If you specify FIXBLK or FIXUNB when creating a CMS file on a CMS disk, CMS writes the file in variable-length format. The LISTFILE command would show the file as V format.
RECSIZE=nnnnn		If RECFORM=FIXBLK, number of characters in record. If RECFORM=SPNUNB, SPNBLK, or UNDEF, register number. Not required for other records.
SEPASMB=YES		DTFSD is to be assembled separately.
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 40. 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 40. 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 that simulate the DOS/VS transients execute in the CMS/DOS 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.

Transient	Function under CMS/DOS
	$$BOPEN checks for supported DTF types, and initializes DTFs in accordance with the device type. In the case of disk files and tape data files, default DLBLs with the NOCHANGE option are issued. (The CMS STATE command is issued to verify the existence of the input files.) $$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 or not 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 Block 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, 32, or 33. 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/DCS provides the necessary path 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, 32, or 33 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 DCS PL/I Optimizing compiler 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 the private core image library.

When you later invoke the compilers from CMS/DOS, the library (system or private) containing the compiler phases must be identified to CMS. You identify all the system libraries to CMS by coding the filemode letter that corresponds to that DOS/VS system disk on the SET DOS CN 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 Guide.

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 through 149 of a 3330 volume labeled DOSRES. And, assume the DOS/VS private libraries are on cylinders 0 through 99 of a 2314 volume labeled DOSPRI. Then, one CMS machine (for example, DOSUSER1) would have the MDISK statements in its directory entry.

```
USER DOSUSER1 password 320K 2M G
    .
    .
    .
MDISK 331 3330 0 150 DOSRES R rpass
MDISK 231 2314 0 100 DOSPRI R rpass
```

All the other CMS/DOS users would have links to these disks. For example

```
LINK DOSUSER1 331 331 R rpass
LINK DOSUSER1 231 231 R rpass
```

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 DCS 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 library program
- 2K for the PSERV library program
- 2K for the SSERV library 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 the private source statement library.
- You invoke one of the library 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 library services to place the DOS/VS COBOL compiler, DOS PL/I compiler, and ESERV program in a CMS DOSLIB, it is recommended that you do not use this method with VM/370. 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 nct 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 ensure that the end-of-data indication is recognized. Be sure that (1) the program checks for end of data and (2) a /* record follows the last data record.

If there is an error in the way you handle end of data, the DCS program could read the entire batch input stream as its own data. The result is that jobs sent to the batch machine are never executed and the DOS program reads records that are not part of its 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 that 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 or for CMS disk files used as input for Access Method Services. 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
- IBM 3340 Direct Access Storage Facility
- IBM 3344 Direct Access Storage
- IBM 3350 Direct Access Storage

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 41, 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 41. 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 upon device type:

Number of Cylinders Required		
Device Type	DOS User	OS User
2314	10	20
2319	10	20
3330 Model 1	6	12
3330 Model 11	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.

Saving the CMS System

Only named systems can be saved. The NAMESYS macro must be used to name a system. A discussion on creating a named system is found under "Generating 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 this saved system, when activated by a subsequent IPL, is changed as though an IPL of a specific device had occurred. The one exception to this procedure is the file directory for the system disk, which is part of the nucleus. When a user IPLs CMS, CP loads the directory for the CMS residence device from disk into virtual storage. Subsequent updates to this directory modify the copy that is in virtual storage but not the copy that is on disk. To modify the copy that is on disk, save the CMS system after updating the directory.

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 upon 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 among 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 not to 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 loaded (via IPL) 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 disk, in this case.

In addition, for the CMSSEG segment only, you can indicate an alternate segment to be loaded on the IPL command. The format of the IPL command to support this is:

```
IPL   cuu          PARM SEG=segmentname
      systemname
```

Where:

SEG=segmentname indicates the name of the saved segment to be loaded whenever the CMS Editor, EXEC processor, or OS simulation routines are needed. Eight characters must be entered for segmentname; either assign an 8-character segment name when you code the NAMESYS macro for your installation, or be sure that the operator enters trailing blanks if segmentname is less than 8 characters long.

The CMS Batch Facility loads whatever segment is specified on the first IPL command issued for the batch virtual machine. Thus, if the first IPL command for a CMS Batch Facility machine is:

```
IPL CMS PARM SEG=CMSSEG02
```

the same segment name (CMSSEG02) is loaded.

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 that 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.

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

CMS Batch Facility

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 CMS Batch Facility then executes these jobs, freeing user machines for other uses.

If both CMS Batch Facility and the Remote Spooling Communications Subsystem (RSCS) are being executed under the same VM/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 CMS Batch Facility virtual machine is generated and controlled on a userid dedicated to execution of jobs in batch mode. The system operator generates the "batch machine" by loading (via IPL) the CMS subsystem, and then issuing the CMSBATCH command. The CMSBATCH module loads the DMSBTP TEXT S2 file, which is the actual batch processor. After each job is executed, the batch facility will IPL itself, thereby providing a continuously processing batch machine. The batch processor will IPL itself by using the PARM option of the CP IPL command, followed by a character string that CMS recognizes as peculiar to a batch virtual machine performing its IPL. Jobs are sent to the batch machine's virtual card reader from users' terminals and executed sequentially. When there are no jobs waiting for execution, the CMS Batch Facility remains in a wait state ready to execute a user job. See the VM/370 Operator's Guide for more information about controlling the batch machine.

The CMS Batch Facility is particularly useful for compute-bound jobs such as assemblies and compilations and for execution of large user programs, since interactive users can continue working at their terminals while their time-consuming jobs are run in another virtual machine.

The system programmer controls the batch facility virtual machine environment by resetting the CMS Batch Facility machine's system limits, by writing routines that handle special installation input to the batch facility, and by writing EXEC procedures that make the CMS Batch Facility facility easier to use.

Resetting the CMS Batch Facility System Limits

Each job running under the CMS Batch Facility is limited by default to the maximum value of 32,767 seconds of virtual processor time, 32,767 punched cards output, and 32,767 printed lines of output. You can reset these limits by modifying the BATLIMIT MACRO file, which is found in the CMSLIB macro library, and by reassembling DMSBTP.

Writing Routines to Handle Special Installation Input

The CMS Batch Facility can handle user-specified control language and special installation batch facility /JOB control cards. These handling mechanisms are built into the system in the form of user exits from batch; you are responsible for generating two routines to make use of them. These routines must be named BATEXIT1 and BATEXIT2, respectively, and must have a filetype of TEXT and a filemode number of 2 if placed on the system disk or an extension of the system disk. (See the VM/370 CMS User's Guide for information on how to write and use CMS Batch Facility control cards.) The routines you write are responsible for saving registers, including general register 12, which saves addressability for the batch facility. These routines (if made available on the system disk) are included with the CMS Batch Facility each time it is loaded.

BATEXIT1: PROCESSING USER-SPECIFIED CONTROL LANGUAGE

BATEXIT1 is an entry point provided so that users may write their own routine to check non-CMS control statements. For example, a routine could be written to scan for the OS job control language needed to compile, link edit, and execute a FORTRAN job. BATEXIT1 receives control after each read from the CMS Batch Facility virtual card reader is issued. General register 1 contains the address of the batch facility read buffer, which contains the card image to be executed by the batch facility. This enables BATEXIT1 to scan each card it receives as input for the type of control information you specify.

If, after the card is processed by BATEXIT1, general register 15 contains a nonzero return code, the CMS Batch Facility flushes the card and reads the next card. If a zero is returned in general register 15, the batch facility continues processing by passing the card to CMS for execution.

BATEXIT2: PROCESSING THE BATCH FACILITY /JOB CONTROL CARD

BATEXIT2 is an entry point provided so that users can code their own routine to use the /JOB card for additional information. BATEXIT2 receives control before the VM/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 processing of this card is nonzero, an error message is generated and the job is flushed. If general register 15 contains a zero, normal checking is done for a valid userid and the existence of an account number. Finally, execution of this job begins.

EXEC Procedures for the Batch Facility Virtual Machine

You can control the CMS Batch Facility virtual machine using EXEC procedures. For example, you can use an EXEC:

- To produce the proper sequence of CP/CMS commands for users who do not know CMS commands and controls.
- To provide the sequence of commands needed to execute the most common jobs (assemblies and compilations) in a particular installation.

For information on how to use the EXEC facility to control the batch facility virtual machine, see the VM/370 CMS User's Guide.

Data Security under the Batch Facility

After each job, the CMS Batch Facility will load (via IPL) itself, destroying all nucleus data and work areas. All disks to which links were established during the previous job are detached.

At the beginning of each job, the batch facility work disk is accessed and then immediately erased, preventing the current user job from accessing files that might remain from the previous job. Because of this, execution of the PROFILE EXEC is disabled for the CMS Batch Facility machine. You may, however, create an EXEC procedure called BATPROF EXEC and store it on any system disk to be used instead of the ordinary PROFILE EXEC. The batch facility 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 more efficient IPL procedure by using a saved CMS system when processing batch jobs.

This can be accomplished by passing the name of the saved system to the CMS Batch Facility via the optional "sysname" operand in the CMSBATCH command line.

The batch facility saves the name of the saved system until the end of the first job, at which time it stores the name in the IPL command line both as the "device address" and as the PARM character string. The latter entry informs the CMS initialization routine (DMSINS) that a saved system has been loaded and that the name is to be saved for subsequent IPL procedures.

Note: When using the CMS SET command, the BLIP operand is ignored when issued from the CMS batch machine.

Auxiliary Directories

When a disk is accessed, each module that fits the description specified on the ACCESS command is included in the resident directory. An auxiliary directory is an extension of the resident directory and contains the name and location of certain CMS modules that are not included in the resident directory. These modules, if added to the resident directory, would significantly increase its size, thus increasing the search time and storage requirements. An auxiliary directory can reference modules that reside on the system (S) disk; or, if the proper linkage is provided, reference modules that reside on any other read-only CMS disk. To take advantage of the saving in search time and storage, modules that are referenced via an auxiliary directory should never be in the resident directory. The disk on which these modules reside should be accessed in a way that excludes these modules.

How to Add an Auxiliary Directory

To add an auxiliary directory to CMS, the system programmer must generate the directory, initialize it, and establish the proper linkage. Only when all three tasks are completed, can a module described in an auxiliary directory be properly located.

GENERATION OF THE AUXILIARY DIRECTORY

An auxiliary directory TEXT deck is generated by assembling a set of DMSFST macros, one for each module name. The format of the DMSFST macro is:

```
┌──────────────────────────────────────────────────────────────────────┐
│ 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. At directory creation, all modules named in the directory being generated must be on either the A-disk or a read-only extension (that is, not all disks are searched). 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 method(s) of access.

ESTABLISHING THE PROPER LINKAGE

The CMS module, DMSLAD, entry point DMSLADAD, must be called by a user program or interface to initialize the directory search order. The subroutine, DMSLADAD, must be called via an SVC 202 with register 1 pointing to the appropriate PLIST. The disk containing the modules listed in the auxiliary directory must be accessed as the mode specified, or implied, by the GENDIRT command before the call is issued. If the GENDIRT command has not been used, the user will receive the messages: "File not found" or "Error reading file."

The coding necessary for the call is:

```
        LA    R1,PLIST
        SVC   202
        DC    AL4(error return)
```

This call must be executed before the call to any module that is to be located via an auxiliary directory.

The PLIST should be:

```
PLIST   DS    0F
        DC    CL8'DMSLADAD'
        DC    V(directoryname)
        DC    F'0'
```

The auxiliary directory is copied into nucleus free storage. The Active Disk Table (ADT) for the targetmode expressed or implied by the GENDIRT command is found and its file directory address chain (ADTFDA) is modified to include the nucleus copy of the auxiliary directory. A flag, ADTPSTM, in ADTFLG2 is set to indicate that the directory chain has been modified.

The address of the nucleus copy of the auxiliary directory is saved in the third word of the input parameter list and the high order byte of the third word is set to X'80' to indicate that the directory search chain was modified and that the next call to DMSLADAD is a clear request.

To reset the directory search chain, a second call is made to DMSLADAD using the modified PLIST. DMSLADAD removes the nucleus copy of the auxiliary directory from the chain and frees it. DMSLADAD does not, however, restore the caller's PLIST to its initial state.

Error Handling and Return Codes

An error handling routine should be coded to handle nonzero return codes in register 15. When register 15 contains 1 and the condition code is set to 2, the disk specified by the targetmode operand of the GENDIRT command was not accessed as that mode.

When register 15 contains 2 and the condition code is set to 2, the disk specified by the targetmode operand of the GENDIRT command has not previously had its file directory chains modified; therefore, a call to DMSLADAD to restore the chain is invalid.

An Example of Creating an Auxiliary Directory

Consider an application called PAYROLL consisting of several modules. It is possible to put these modules in an auxiliary directory rather than in the resident directory. It is further possible to put the auxiliary directory on a disk other than the system disk. In this example, the auxiliary directory 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 auxiliary 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 fo
   .                    each payroll module called by PAYROLL.)
   .
   .
LOADMOD PAYROLL
INCLUDE PAYSHIFT
GENMOD PAYSHIFT

LOADMOD PAYROLL
GENDIRT PAYDIRT Y
GENMOD PAYROLL MODULE A2
```

When it is time to execute the PAYROLL program, the 194 disk must be accessed as the Y disk (the same mode letter as specified on the GENDIRT command). Also, the 194 disk is accessed in a way that includes the PAYROLL control program in the resident directory but not the other payroll modules. This is done by specifying a mode number of 2 on the ACCESS command.

```
ACCESS 194 Y/S * * Y2
```

Now, a request for a payroll module, such as PAYOVERT, can be successfully fulfilled. The auxiliary directory will be searched and PAYOVERT will be found on the Y disk.

Note: A disk referred to by an auxiliary directory must be accessed as a read-only disk.

Assembler Virtual Storage Requirements

The minimum size virtual machine required by the assembler is 256K bytes. However, better performance is generally achieved if the assembler is run in 320K bytes of virtual storage. This size is recommended for medium and large assemblies.

If more virtual storage is allocated to the assembler, the size of buffers and work space can be increased. The amount of storage allocated to buffers and work space determines assembler speed and capacity. Generally, as more storage is allocated to work space, larger and more complex macro definitions can be handled.

You can control the buffer sizes for the assembler utility data sets (SYSUT1, SYSUT2, and SYSUT3), and the size of the work space used during macro processing, by specifying the BUFSIZE assembler option. Of the storage given, the assembler first allocates storage for the ASSEMBLE and CMSLIB buffers according to the specifications in the DD statements supplied by the FILEDEF for the data sets. It then allocates storage for the modules of the assembler. The remainder of the virtual machine is allocated to utility data set buffers and macro generation dictionaries according to the BUFSIZE option specified:

BUFSIZE(STD): 37 percent is allocated to buffers, and 63 percent to work space. This is the default if you do not specify any BUFSIZE option.

BUFSIZE(MIN): Each utility data set is allocated a single 790-byte buffer. The remaining storage is allocated to work space. This allows relatively complex macro definitions to be processed in a given virtual machine size, but the speed of the assembly is substantially reduced.

Overlay Structures

An overlay structure can be created in CMS in two different ways, although CMS has no overlay supervision. For descriptions of all the CMS commands mentioned, see the VM/370 CMS Command and Macro Reference.

PRESTRUCTURED OVERLAY

A prestructured overlay program is created using the LOAD, INCLUDE, and GENMOD commands. Each overlay phase or segment is a nonrelocatable core-image module created by GENMOD. The phases may be brought into storage with the LOADMOD command.

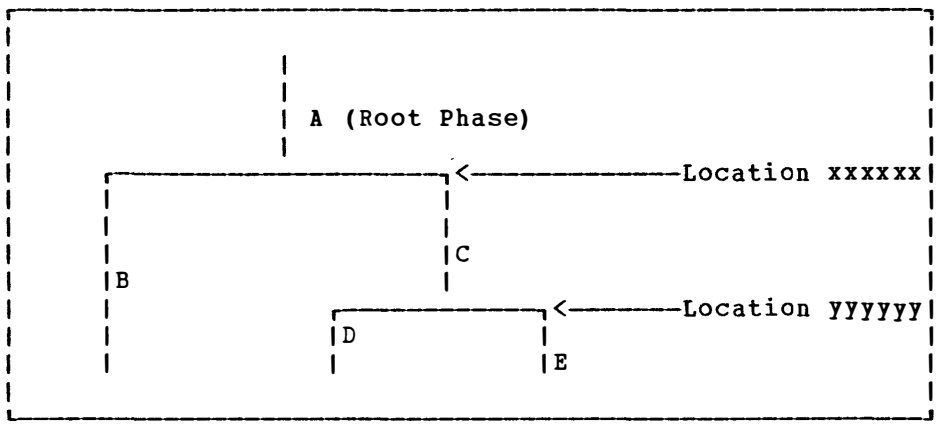


Figure 42. An Overlay Structure

The overlay structure shown in Figure 42 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 41, locations xxxxxx and yyyyyy). The following sequence of commands could be used:

```
LOAD     A B
GENMOD   SECOND (FROM B)
INCLUDE  C D    (ORIGIN xxxxxx)
GENMOD   THIRD  (FROM C TO D)
GENMOD   FOURTH (FROM D)
INCLUDE  E      (ORIGIN yyyyyy)
GENMOD   FIFTH  (FROM E)
LOAD     A B
INCLUDE  C D    (ORIGIN xxxxxx)
INCLUDE  E      (ORIGIN yyyyyy)
GENMOD   ROOT   (FROM A TO C STR)
```

The ORIGIN option of the INCLUDE command is used to cause the included file to overlay a previously loaded file. The address at which a phase begins must be a doubleword boundary. For example, if the root phase were X'2BD' bytes long, starting at virtual storage location X'20000', then location xxxxxx would be the next doubleword boundary, or X'202C0'.

The STR option, which is specified in the GENMOD of the root phase, specifies that whenever that module is brought into storage with the LOADMOD command, the Storage Initialization routine should be invoked. This routine initializes user free storage pointers.

At execution time of the prestructured overlay program, each phase is brought into storage with the LOADMOD command. The phases can call LOADMOD. The OS macros LINK, LOAD, and XCTL normally invoke the INCLUDE command, which loads TEXT files. These macros will invoke LOADMOD if a switch, called COMPSWT, in the CMS Nucleus Constant area, NUCON, is turned on.

With COMPSWT set, overlay phases that use LINK, LOAD, and XCTL must be prestructured MODULE files.

DYNAMIC LOAD OVERLAY

The dynamic load method of using an overlay structure is to have all the phases in the form of relocatable object code in TEXT files or members of a TEXT library, filetype TXTLIB. The OS macros, LINK, LOAD, and XCTL may then be used to pass control from one phase to another. The XCTL macro causes the calling program to be overlayed by the called program except when it is issued from the root phase. When issued from the root phase, CMS treats XCTL as it would a LINK macro, adding the new code at the end of the root phase.

The COMPSWT flag in OSSFLAGS must be off when the dynamic load method is used.

Part 4. Remote Spooling Communications Subsystem (RSCS)

Part 4 contains the following information:

- Introduction to RSCS
- Structure of RSCS virtual storage
- Functional information
- Logging I/O activity

Introduction to RSCS

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 for 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 43; for a full description and the format of each, refer to "Appendix A: Remote Spooling" Communications Subsystem Commands" in the VM/370 Remote Spooling Communications Subsystem (RSCS) User's Guide.

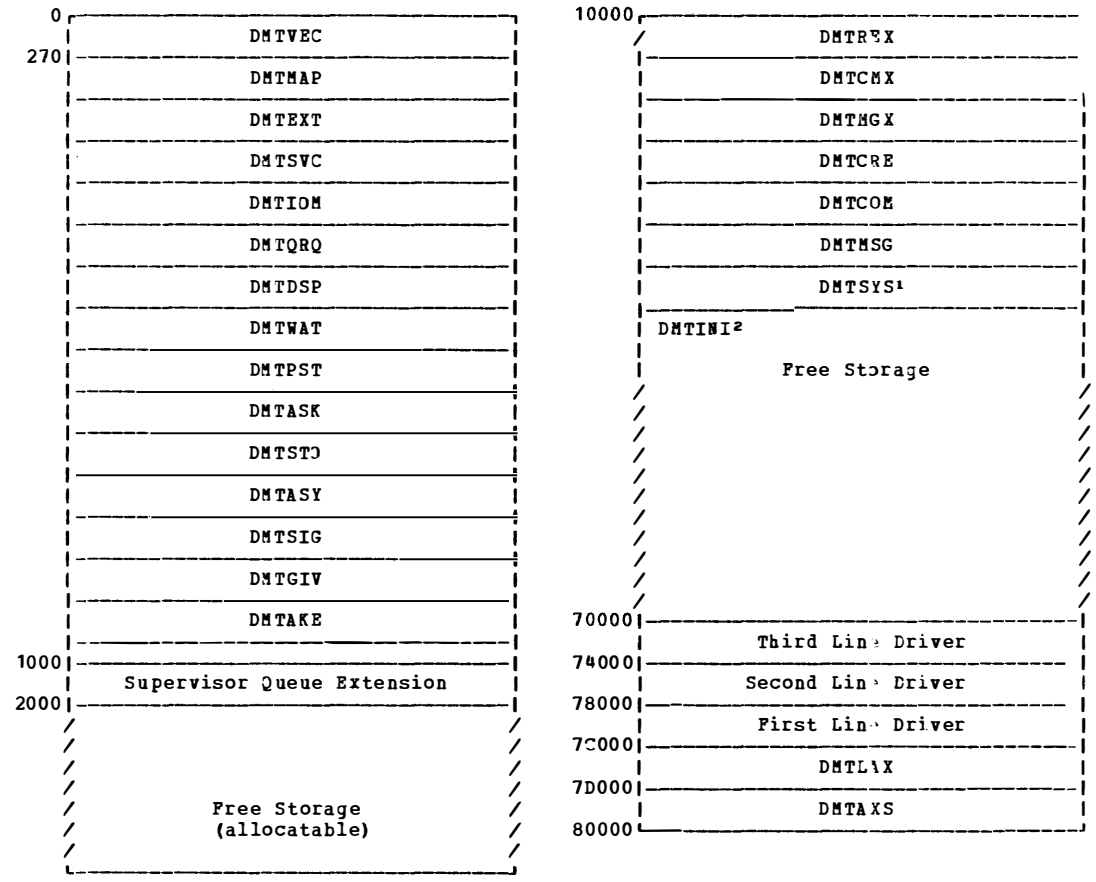
Command Name	Function
BACKSPAC	Restart or reposition, in a backward direction, the file currently being transmitted.
CHANGE	Alter one or more attributes of a file owned by RSCS.
CMD	Control certain functions performed by a remote system, or control the logging of I/O activity on a specified link.
DEFINE	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 the system in general.
START	Activate a specified communication link.
TRACE	Monitor line activity on a specified link.

Figure 43. 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 affect file processing at 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 44 shows how RSCS storage is allocated.



[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 44. RSCS Storage Allocation

RSCS Supervisor

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, as follows:

Routine	Function
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

The supervisor queue extension is a chain of 16-byte elements that provide an extension to the supervisor queue located in DMTMAP.

Free Storage

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

The system control task consists of five executable and two nonexecutable modules, as follows:

Module	Function
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

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

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

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.

Functional Information

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, DMTSTC. 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 throughout 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.

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 it is, 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 then 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 synchronous 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 it is 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 synchronous lock for this transaction.

67–68 The sense byte.

71–86 The CCW associated with the I/O operation.

```
SAMPLES OF READ AND WRITE RECORDS FOR SML



1070                                          0779C80C00018E 800000  00  0207100720000190
1070                                          0779C80C00018E 000000  00  0107100760000002
1002808FCF9094000026                          0779C80C000186 800000  00  0207100720000190
100 28 18FCFA0940000                          0779C80C000186 800000  00  0107100760000009
100 28 18FCF9491C140009483C140009483C1400094  0779C80C00003C 800000  00  0207100720000190
1070                                          0779C80C00003C 800000  00  0107119F60000002
                                              0779C80E000190 800000  01  0207119F20000190
1002828FCF9483C8C6C9D3C57A40C4E787C4C5E7C5    0779C80E00005C 800000  02  0207119F20000190
323D                                          0779C80E00005C 000000  00  0107119F60000002
100 2828FCF9483E4C4C5E2E37A40C8D6E2E3D3C9D5   0779C80C00000C 800000  00  0207119F20000190
1070                                          0779C80C00000C 000000  00  0107100760000002
1002838FCF9481CC50D5E4D4C2C5D9407E4050F100    0779C80C000008 800000  00  0207100720000190
1070                                          0779C80C000008 000000  00  0107119F60000002
1002848FCF9481FF5C5C5C40C3C1E4E2C5E240E3C8    0779C80C000003 800000  00  0207119F20000190
1070                                          0779C80C000003 000000  00  0107100760000002
100 2858FCF9481C7C3D740D84007C6009481E350E3   0779C80C0000E7 800000  00  0207100720000190
1070                                          0779C80C0000E7 000000  00  0107119F60000002

                                                SML INTERNAL SYNCH SENSE       CCW
                                                   BUFFER    LOCK  BYTE

21 BSC, MULTILEAVING AND DATA BYTES                  - OR -

         TP BUFFER                           ADDR STATUS COUNT
                                                  BYTES

                                                   CSW
```

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 operation.

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
|                          |  |          | |            |  A  |              |
└──────────V───────────────┘  └─────V─────┘ └─────V──────┘  |  └──────V───────┘
    First 14 bytes            Last 6 bytes      CSW         |         CCW
|                                          |
└──────────────────V───────────────────────┘                SENSE
               TP Buffer                                    BYTE
```

Note: The dashes in record positions 1 and 2 indicate that there was no data transfer for that I/O transaction.

Appendixes

- Appendix A: System/370 Information
- Appendix B: MULTI-LEAVING
- Appendix C: VM Monitor Tape Format and Content

Appendix A: System/370 Information

Control Registers

The control registers are used to maintain and manipulate control information that resides outside the PSW. There are sixteen 32-bit registers for control purposes. The control registers are not part of addressable storage.

At the time the registers are loaded, the information is not checked for exceptions, such as invalid segment-size or page-size code or an address designating an unavailable or a protected location. The validity of the information is checked and the errors, if any, indicated at the time the information is used.

Figure 45 is a summary of the control register allocation and Figure 46 lists the facility associated with each control register.

Figure 47 is a description of the EC (Extended Control) PSW.

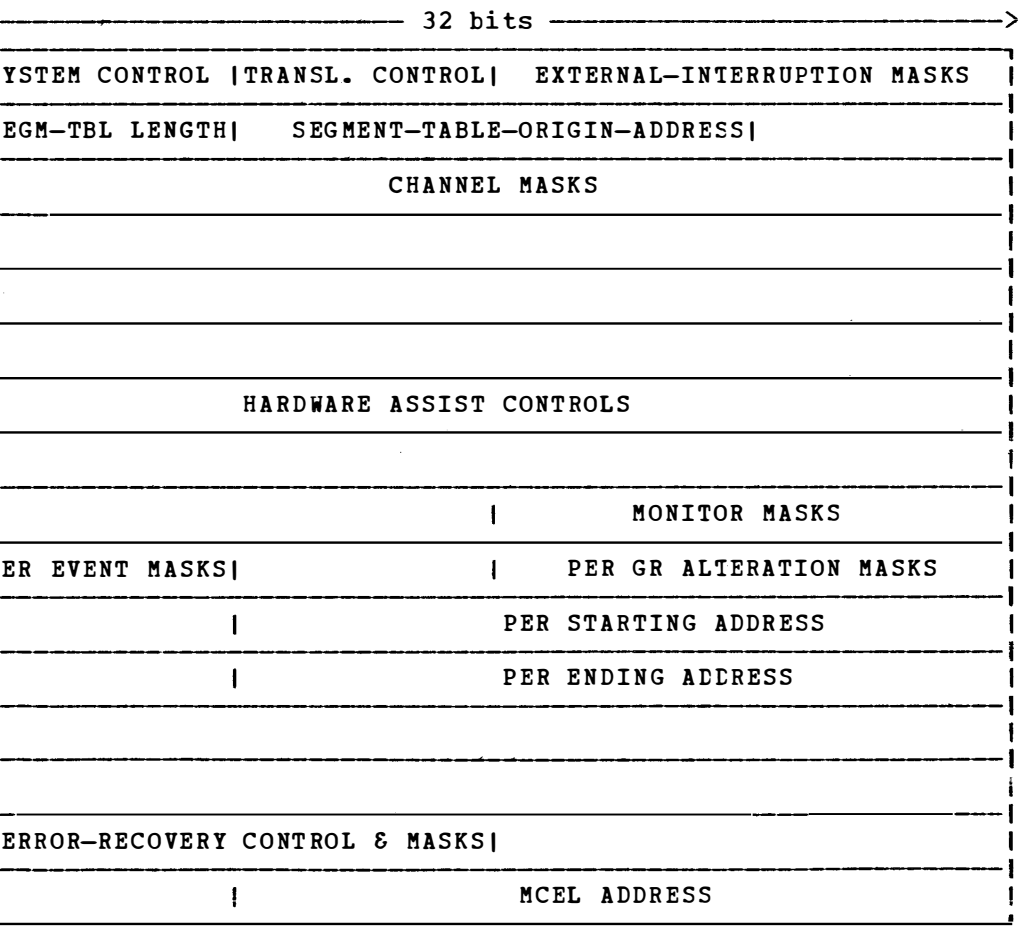


Figure 45. Control Register Allocation

Word	Bits	Name of Field	Associated with	Initial Value
0	0	Block-Multiplex Mode	Block-Multiplex Control	1
0	1	SSM Suppression	Extended Control	0
0	2	TOD Clock Synchronous Ctrl.	Attached Processing	0
0	8-9	Page Size[1]	Dynamic Addr. Translation	10
0	10	Reserved	Dynamic Addr. Translation	0
0	11-12	Segment size[1]	Dynamic Addr. Translation	00
0	16	Malfunction Alter Mask	Attached Processing	1
0	17	Emergency Signal Mask	Attached Processing	1
0	18	External Call Mask	Attached Processing	1
0	19	TOD Synchronous Check Mask	Attached Processing	1
0	20	Clock Comparator Mask	Clock Comparator	1
0	21	Processor Timer Mask	Processor Timer	0
0	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 varies with the type of virtual machine.
1	8-25	Segment Table Address	Dynamic Addr. Translation	
2	0-31	Channel Masks	I/O Interruptions	FFFFFFFF. Set to zero on the attached processor in attached processor systems
6	0	VM Assist	Hardward Assist	Value depends upon virtual machine
6	1	VM Problem State	Hardware Assist	Value depends upon virtual machine
6	2	ISK & SSK	Hardware Assist	Value depends upon virtual machine
6	3	S/360 or S/370 instructions	Hardware Assist	Value depends upon virtual machine
6	4	Virtual SVC Interrupts	Hardware Assist	Value depends upon virtual machine
6	5	Shadow Table Fixup	Hardware Assist	Value depends upon virtual machine
6	6	CP Assist	Hardware Assist	Value depends upon virtual machine
6	7	Virtual Interval Timer	Hardware Assist	Value depends upon virtual machine
6	8-28	Real address of VM pointer list	Hardware Assist	Value depends upon virtual machine
8	16-31	Monitor Masks	Monitoring	Value depends on upon virtual machine
9	0-3	PER[2] Event Masks	Program-Event Recording	Value depends upon virtual machine.
9	16-31	PER GR Alteration Masks	Program-Event Recording	
10	8-31	PER Starting Address	Program-Event Recording	Value depends upon virtual machine.

Explanation:
The fields not listed are unassigned.
The initial value of unassigned register positions is unpredictable.

[1] The initial value varies depending upon whether virtual storage is supported in the virtual machine.
[2] PER means program-event recording.

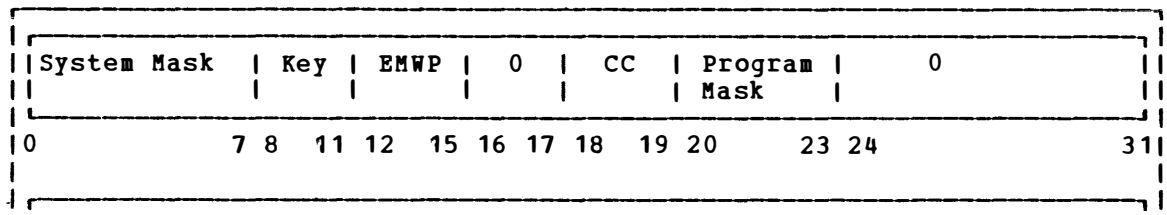
Figure 46. Control Register Assignments (Part 1 of 2)

Word	Bits	Name of Field	Associated with	Initial Value
11	8–31	PER Ending Address	Program-Event Recording	Value depends upon virtual machine.
14	0	Check-Stop Control	Machine-Check Handling	Value depends upon machine check handler for the virtual machine.
14	1	Synchronous MCEL[3] Control	Machine-Check Handling	
14	2	I/O Extended Logout Control	Channel-Check Handling	
14	4	Recovery Report Mask	Machine-Check Handling	
14	5	Degradation Report Mask	Machine-Check Handling	
14	6	External Damage Report Mask	Machine-Check Handling	
14	7	Warning Mask	Machine-Check Handling	
14	8	Asynchronous MCEL Control	Machine-Check Handling	
14	9	Asynchronous Fixed Log Ctrl.	Machine-Check Handling	
15	8–28	MCEL Address	Machine-Check Handling	Points to extend I/O logout area

Explanation:
The fields not listed are unassigned.
The initial value of unassigned register positions is unpredictable.

[3] MCEL means machine-check extended logout.

Figure 46. Control Register Assignments (Part 2 of 2)



The fields of the PSW are:

Bits	Contents
0	Must be zero.
1	PER (Program Event Recording) enabled.
2–4	Must be zero.
5	Address translation.
6	Summary I/O mask.
7	Summary extension.
8–11	The protection key determines if information can be stored or fetched from a particular location.
12	Extended control mode.
13	The machine check flag is set to 1 if machine check interruptions are enabled.
14	The wait state flag is set to 1 when the CPU is in the wait state.
15	The problem state flag is set to 1 when the CPU is operating in the problem rather than the supervisor state.
16–17	Must be zero.
18–19	The condition code reflects the result of a previous arithmetic, logical, or I/O operation.
20–23	The program mask indicates whether or not various program exceptions are allowed to cause program interrupts.
24–32	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 47. 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 (BSC) 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 (for example, SYZ123&#!).

   ---and---

2. A variable number of identical characters (for example, sssssssss).

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 48. 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 48), 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 addition 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--		
O	1	Non-EOT RCB
III0000		III is control information:
III	000	Reserved
	001	Request to initiate a function transmission (prototype RCB for function in SRCE)
	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--		
O	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:

O	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:

O	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

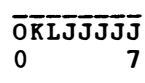
```
 ________
 OTTTTTTT
 0      7
```

Usage: To indicate the destination of a terminal message

Bits:

O	1	
TTTTTTT	0000000	Broadcase to all remote systems
	NNNNNNN	Remote system number (1-99) or remote system group (100-127)

STRING CONTROL BYTE (SCB)



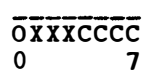
Usage: Control field for data character strings

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)



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:

O...O	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/370 Monitor receives control and collects data appropriate for the particular class and code of MONITOR CALL. (Or, for USER, PERFORM, or DASTAP classes, VM/370 Monitor gets control at periodic intervals to collect data.) The data is formatted into records that are collected sequentially in the order that each interrupt occurred. The tape data format is standard Variable Blocked (VB) format. Data is written at the default tape drive density. Maximum block and record lengths are 4096 bytes. The formats and contents of all the kinds of data records for the currently implemented classes and codes of MONITOR CALL are listed below.

All values described in the following records are binary unless otherwise noted.

[1]Indicates that the field is EBCDIC.

[2]Indicates that the field is in special timer format described below.

[3]See VM/370 Data Areas and Control Block Logic 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 fullwords in storage, with the rightmost 12 bits zeros. The rightmost 2 bytes and the leftmost byte are ignored, giving 16-microsecond accuracy instead of 1-microsecond.

The first 4 bytes of this header are the standard variable-format record field.

Data Records

Class Zero - Codes for Tape Header, Trailer, and Data Suspension Records

Monitor Code	Data Item	Number of Bytes	CP Variable Name	DSECT Variable Name
97	Tape header record			
	CPU serial/model number	8	CPUID	MN097CPU
	Software version number[1]	8	DMKCPEID	MN097LEV
	Date of data collection session[1]	8	TOD clock	MN097DAT
	Time of data collection session[1]	8	TOD clock	MN097TIM
	Userid of monitor controller[1]	8	VMUSER	MN097UID
	CR8 mask of enabled classes	4	DMKPRGC8	MN097CR8
	Size of CP nucleus	4	Derived by CP	MN097NUC
	Size of Free/Fret pools	4	Derived by CP	MN097FSS
	Size of dynamic paging area	4	Derived by CP	MN097DPA
	Size of trace table	4	Derived by CP	MN097TTS
	Size of V=R area (if any)	4	Derived by CP	MN097VR
	CPU logical address	2	LPUADDR	MN097CPL
	APU logical address	2	LPAUDDRX	MN097APL
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 main processor idle time[3]	8	IDLEWAIT	MN000WID
	Total main processor page wait[3]	8	PAGEWAIT	MN000WPG
	Total main processor time I/O wait[3]	8	IONTWAIT	MN000WIO
	Total main processor problem time[3]	8	PROBTIME	MN000PRB
	Total paging start I/Os	4	DMKPAGPS	MN000PSI
	Total page I/O requests	4	DMKPAGCC	MN000CPA
	Current page frames on free list	4	DMKPTRFN	MN000NFL
	Pages being written, due for free list	4	DMKPTRSW	MN000PSN
	Total pages flushed, but reclaimed	4	DMKPTRPR	MN000PRC
	Number of reserved pages	4	DMKPTRRC	MN000RPC
	Number of shared system pages	4	DMKPTRSC	MN000SPC
	Total number of times free list empty	4	DMKPTRF0	MN000FLF
	Total number of calls to DMKPTRFR	4	DMKPTRFC	MN000CPT
	Total pages stolen from in-queue users	4	DMKPTRSS	MN000SS
	Number of pages examined in stealing pages	4	DMKPTRRF	MN000PRF
	Number of pages swapped from the flush list	4	DMKPTRFF	MN000PFF
	Number of full scans done in stealing pages	4	DMKPTRCS	MN000PCS

Monitor Code	Data Item	Number of Bytes	CP Variable Name	DSECT Variable Name
	Total real external interrupts to main processor	4	DMKPSANX	MN000NXR
	Total calls to DMKPRVLG	4	DMKPRVNC	MN000CPR
	Total calls to DMKVIOEX	4	DMKVSICT	MN000CVI
	Total calls to CCWTRANS from DMKVIO	4	DMKVSICW	MN000CCW
	Total virtual interval timer interrupts reflected	4	DMKDSPIT	MN000ITI
	Total virtual CPU timer interrupts reflected	4	DMKDSPPT	MN000PTI
00	Total virtual clock comparator interrupts reflected	4	DMKDSPCK	MN000CKI
	Total virtual SVC interrupts simulated by main processor	4	PSASVCCT	MN000CSV
	Total virtual program interrupts handled	4	DMKPRGCT	MN000CPG
	Total I/O interrupts handled	4	DMKIOSCT	MN000CIO
	Total calls to dispatch (main)	4	DMKDSPCC	MN000CDS
	Total fast reflects in dispatch	4	DMKDSPAC	MN000CDA
	Total dispatches for new PSW	4	DMKDSPBC	MN000CDB
	Total calls to schedule	4	DMKSCHCT	MN000CSC
	Count of virtual machine SSK simulated	4	DMKPRVEK	MN000EK
	Count of virtual machine ISK simulated	4	DMKPRVIK	MN000IK
	Count of virtual machine SSM simulated	4	DMKPRVMS	MN000MS
	Count of virtual machine LPSW simulated	4	DMKPRVLP	MN000LP
	Count of virtual machine diagnose instructions	4	DMKPRVDI	MN000DI
	Count of virtual machine SIO simulated	4	DMKVSISI	MN000SI
	Count of virtual machine SIOF simulated	4	DMKVSISF	MN000SF
	Count of virtual machine TIO simulated	4	DMKVSITI	MN000TI
	Count of virtual machine CLRIO simulated	4	DMKVSICI	MN000CI
	Count of virtual machine HIO simulated	4	DMKVSIHI	MN000HI
	Count of virtual machine HDV simulated	4	DMKVSIHD	MN000HD
	Count of virtual machine TCH simulated	4	DMKVSITC	MN000TC
	Count of virtual machine STNSM simulated	4	DMKPRVMN	MN000MN
	Count of virtual machine STOSM simulated	4	DMKPRVMO	MN000MO
	Count of virtual machine LRA simulated	4	DMKPRVLR	MN000LR
	Count of virtual machine STIDP simulated	4	DMKPRVCP	MN000CP
	Count of virtual machine STIDC simulated	4	DMKPRVCH	MN000CH
	Count of virtual machine SCK simulated	4	DMKPRVTE	MN000TE
	Count of virtual machine SCKC simulated	4	DMKPRVCE	MN000CE
	Count of virtual machine STCKC simulated	4	DMKPRVCT	MN000CT
	Count of virtual machine SPT simulated	4	DMKPRVPE	MN000PE
	Count of virtual machine STPT simulated	4	DMKPRVPT	MN000PT
	Count of virtual machine SPKA simulated	4	DMKPRVEP	MN000EP
	Count of virtual machine IPK simulated	4	DMKPRVIP	MN000IP
	Count of virtual machine PTLB simulated	4	DMKPRVPB	MN000PB
	Count of virtual machine RRB simulated	4	DMKPRVRR	MN000RR
	Count of virtual machine STCTL simulated	4	DMKPRVTC	MN000TCL
	Count of virtual machine LCTL simulated	4	DMKPRVLC	MN000LCL
	Count of virtual machine CS simulated	4	DMKPRVCS	MN000CS
	Count of virtual machine CDS simulated	4	DMKPRVCD	MN000CD
	Count of virtual machine diagnose disk I/O	4	DMKHVCDI	MN000HDI
	Number of users dialed to virtual machines	4	DMKSYSND	MN000NDU
	Number of users logged on	4	DMKSYSNM	MN000NAU
	Number of page reads by main processor	4	PGREAD	MN000PRD
	Number of page writes by main processor	4	PGWRITE	MN000PWR
	Number of system pageable pages	4	DMKDSPNP	MN000NPP
	Sum of working sets of in-queue users	4	DMKSCNPU	MN000SWS
	Number of users in interactive queue (Q1)	4	DMKSCHN1	MN000Q1N
	No. of users in compute-bound queue (Q2)	4	DMKSCHN2	MN000Q2N

Monitor Code	Data Item	Number of Bytes	CP Variable Name	DSECT Variable Name
	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
	Count of mini IOB stack depletes	4	DMKIOSNM	MN000ISD
	Count of mini IOB enqueues	4		MN000GTM
	Count of mini IOB dequeues	4		MN000DQM
	Count of SIOs on alternate paths	4		MN000SWP
	Count of FREE/FRET extends	4	DMKFRENP	MN000EXT
	Count of FREE/FRET unextends	4		MN000NXT
	Count of attempts to split subpool	4		MN000ATT
	Count of SUBPOOL SPLITS	4		MN000CNT
01	Internal statistics for attached processor			
	Total attached processor idle wait time	8	IDLEWAIT	MN001WID
	Total attached processor page wait time	8	PAGEWAIT	MN001WPG
	Total attached processor I/O wait time	8	IONTWAIT	MN001WIO
	Total attached processor problem time	8	PROBTIME	MN001PRB
	Total real external interrupts for attached processor	4	DMKPSANX	MN001NXR
	Total SVCs reflected by attached processor	4	PSASVCCT	MN001CSV
	Page reads by attached processor	4	PGREAD	MN001PRD
	Page writes by attached processor	4	PGWRITE	MN001PWR
	Total time spin on system lock	4	DMKLOKSY+8	MN001SSY
	Number of spins on system lock	4	DMKLOKSY+12	MN001NSY
	Total time spin on DMKFRE lock	4	DMKLOKFR+8	MN001SFR
	Number of spins on DMKFRE lock	4	DMKLOKFR+12	MN001NFR
	Total time spin on RUNLIST lock	4	DMKLOKRL+8	MN001SRN
	Number of spins on RUNLIST lock	4	DMKLOKRL+12	MN001NFR
	Total time spin on timer request lock	4	DMKLOKTR+8	MN001STM
	Number of spins on timer request lock	4	DMKLOKTR+12	MN001NTM
	Total time spin on dispatcher queue lock	4	DMKLOKDS+8	MN001SDP
	Number of spins on dispatcher queue lock	4	DMKLOKDS+12	MN001NDP
	Number of times CPFRELK set	4		MN001NFL
	Number of times CPFRESW set	4		MN001NFS
	Number of times system lock deferred	4	LOKSYSCT	MN001NSD
	Number of times VMBLOK lock deferred	4	LOKVMCT	MN001NVD
	Number of DMKDSPRU entries	4		MN001NRU

Class One - RESPONSE

Monitor Code	Data Item	Number of Bytes	CP Variable Name	DSECT Variable Name
00	Read command sent to terminal			
	Userid	8	VMUSER	MN10XUID
	Line address	2		MN10XADD
01	Terminal output line			
	userid	8	VMUSER	MN10XUID
	Line address	2		MN10YADD
	Byte count	1		MN10YCNT
	Line of data	Variable		MN10YIO
02	Edited terminal input line			
	userid	8	VMUSER	MN10XUID
	Line address	2		MN10XADD
	Byte count	1		MN10YCNT
	Line of data[1]	Variable		MN10YIO
03	Sleep issued with time out			
	Userid	8	VMUSER	MN10XUID
	Line address	2		MN10XADD
04	Terminal logged on			
	Userid	8	VMUSER	MN10XUID
	Line address	2		MN10XADD
05	Terminal logged off			
	Userid	8	VMUSER	MN10XUID
	Line address	2		MN10XADD

Note that the line addresses for the 370X in NCP mode appear as the base address.

These records are created at the time that DMKQCN handles the console I/O request. This may reflect a slightly different time than that of the SIO or the I/O interrupt. If DMKQCN is called to write a line that is longer than Terminal line size, more than one MC is issued, resulting in more than one record. Input and output terminal data collected is limited to 128 bytes. Longer lines are truncated.

Class Two - SCHEDULE

Monitor Code	Data Item	Number of Bytes	CP Variable Name	DSECT Variable Name
02	User dropped from dispatch queue			
	Userid[1]	8	VMUSER	MN20XUID
	Number of system pageable pages	4	DMKDSPNP	MN20XNPP
	Sum of working sets of in-queue users	4	DMKSCHPU	MN20XSWS
	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
	Processor address	1	---	MN20XPRC
	Accumulated user CP simulation time[3]	8	VMTTIME	MN20YTTI
	Accumulated user virtual time[3]	8	VMVTIME	MN20YVTI
	Externally assigned dispatch priority	2	VMQPRIOR	MN20YPRI
	Pages read while in queue	2	VMPGREAD	MN202PGR
	Sum of pages resident at all reads	2	VMPGRINQ	MN202APR
	Current number of pages resident	2	VMPAGES	MN202RES
	Number of pages stolen while in queue	2	VMSTEALS	MN202PST
	User total virt non-spool device SIO count	4	VMIOCNT	MN202IOC
	User total virtual cards punched	4	VMPNCH	MN202PNC
	User total virtual lines printed	4	VMLINS	MN202LIN
	User total virtual cards read	4	VMCRDS	MN202CRD
	User last executed on this processor	1	VMLSTPRC	MN202LPR
03	User added to dispatch queue			
	Userid	8	VMUSER	MN20XUID
	Number of system pageable pages	4	DMKDSPNP	MN20XNPP
	Sum of working sets of in-queue users	4	DMKSCHPU	MN20XSWS
	Number of users in interactive queue (Q1)	4	DMKSCHN1	MN20XQ1N
	No. of users in compute-bound queue (Q2)	4	DMKSCHN2	MN20XQ2N
	Number of users eligible for Q1	2	DMKSCHW1	MN20XQ1E
	Number of users eligible for Q2	2	DMKSCHW2	MN20XQ2E
	User's projected working set size	2	VMWSPROJ	MN20XWSS
	Queue being added to	1	gen reg 15	MN20XQNM
	Processor address (main or attached)	1	---	MN20XPRC
04	User added to eligible list			
	Userid	8	VMUSER	MN20XUID
	Number of system pageable pages	4	DMKDSPNP	MN20XNPP
	Sum of working sets of in-queue users	4	DMKSCHPU	MN20XSWS
	Number of users in interactive queue (Q1)	4	DMKSCHN1	MN20XQ1N
	No. of users in compute-bound queue (Q2)	4	DMKSCHN2	MN20XQ2N
	Number of users eligible for Q1	2	DMKSCHW1	MN20XQ1E
	Number of users eligible for Q2	2	DMKSCHW2	MN20XQ3E
	User's projected working set size	2	VMWSPROJ	MN20XWSS
	Queue being added to	1	VMQ1	MN20XQNM
	Processor address (main or attached)	1	---	MN20XPRC
	Accumulated user CP simulation time	8	VMTTIME	MN20YTTI
	Accumulated user virtual time	8	VMVTIME	MN20YVTI
	Eligible list priority	2	VMEPRIOR	MN20YPRI

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
	User operating status	1	VMOSTAT	MN400OST
	User queuing status	1	VMQSTAT	MN400QST
	User processing status	1	VMPSTAT	MN400PST
	User control status	1	VMESTAT	MN400EST
	User tracing control	1	VMTRCTL	MN400TST
	User message level	1	VMMLEVEL	MN400MLV
	User queue level	1	VMQLEVEL	MN400QLV
	User command level	1	VMCLEVEL	MN400CLV
	User timer level	1	VMTLEVEL	MN400TLV
	Interrupt pending summary	1	VMPEND	MN400PND
	User's externally assigned priority	1	VMUPRIOR	MN400UPR
	Reserved	1	---	MN4RSV1
00	Current number of pages resident	2	VMPAGES	MN400RES
	Current working set size estimate	2	VMWSPROJ	MN400WSS
	Page frames allocated on drum	2	VMPDRUM	MN400PDR
	Page frames allocated on disk	2	VMPDISK	MN400PDK
	Monitor sampling interval (seconds)	2	DMKPRGTI	MN400INT

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	MN500CVH

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	2	RDEVADDR+ RCUADDR+ 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 0 record is collected when the MONITOR START TAPE command is entered. Thereafter, all DASTAP records are collected with a monitor code of 1.

02	number of high frequency samples in interval	2	MONCHPTR	MN602SAM
	Device Address	2	Derived by CP	MN602CHB
	Times channel busy during interval	2	MCNCHPTR+2	MN602CHB
	Times control unit busy during interval	2	MNCUBSY	MN602CUB
	Times device busy during interval	2	MNCUBST+2	MN602DVB
	Number of I/O tasks queued on channel	2	RCHQCNT	MN602CHQ
	Number of I/O tasks queued on control unit	2	RCUQCNT	MN602CUQ
	Number of I/O tasks queued on device	1	RDEVQCNT	MN602DVQ

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 cylinder address
X'01' seeking to higher cylinder address

Class Eight - SYSPROF additional data for system profile class

Monitor Code	Data Item	Number of Bytes	CP Variable Name	DSECT Variable Name
02	Additional data at add queue, drop queue times			
	Number of 4-byte device block counts which follow	2	---	MN802NUM
	For each device ...count of I/O's	4	RDEVIOCT	---
	After device counts ...			
	Current number of users logged on	4	DMKSYSNM	MN802NAU
	Total system page reads	4	PGREAD	MN802PGR
	Total system page writes	4	PGWRITE	MN802PGW
	Current number of pageable pages	4	DMKDSPNP	MN802NPP
	Total system idle time	8	IDLEWAIT	MN802WID
	Total system page wait time	8	PAGEWAIT	MN802WPG
	Total system I/O wait time	8	IONTWAIT	MN802WIO
	Total system problem time	8	PROBTIME	MN802PRB

Index

The entries in this Index are accumulative. They list additions to this publication by the following VM/370 System Control Program Products:

- VM/370 Basic System Extensions, Program Number 5748-XX8
- VM/370 System Extensions, Program Number 5748-XE1

However, the text within the publication is not accumulative; it only relates to the one SCP program product that is installed on your system. Therefore, there may be topics and references listed in this Index that are not in the body of this publication.

$$BCLOSE transient 304-305
$$BDUMP transient 304-305
$$BOPEN transient 304-305
$$BOPENR transient 304-305
$$BOPNLB transient 304-305
$$BOPNR2 transient 304-305
$$BOPNR3 transient 304-305

A
abend (see abnormal termination (abend))
ABEND macro 275
abnormal termination (abend) 4
(see also problem types)
CMS abend
debugging 20
reason for 69
reasons for 20
recovery 21-22
collect information 47-48,70-72
CP abend
debugging 18-19
reason for 18-19,46-47
recovery 46-47
CP dump 44-45
dump (see also CMS (Conversational Monitor System), dump)
dump (see also CP (Control Program), dump)
dump 44-45,45-46,66-68
attached processor 45-46
in CMS 4-6
in CP 4
in DOS 4-6
in OS 4-6
internal trace table 47-48
messages 4-6
of system routine 20
OS (operating system), debugging 23
program check in CP 18
program interrupt 86
program interrupt (5748-XX8) 86.1
program interrupt (5748-XE1) 86.1
reason for 18,20,69
register usage 48
save area conventions 48-49
SVC 0 18,45-46
system 20
SYSTEM RESTART button 19
virtual machine abend, debugging 23
ACCESS command, accessing OS data sets 281
access method, OS, support of 278-279
account number, replacing directory entry 206-210
accounting
ACCTOFF routine 130-131
ACCTON routine 130-131
cards, generating 197-198.2
records
created by user 130-130
for AUTOLOG, LOGON, and LINK journaling 128-129
format for dedicated devices 127-131
format for dedicated devices (5748-XX8) 126
format for dedicated devices (5748-XE1) 127-131
format for virtual machines 127-131
when to punch 130-130
user options 130-131
activating the TOD-clock accounting interface (5748-XE1) 204-204.1
Active Disk Table (ADT) 72,320-321
Active File Table (AFT) 72
address, stop 40
ADSTOP command 40
summary 32
ADT (see Active Disk Table (ADT))
affinity, in attached processor mode 97-98,181
AFT (see Active File Table (AFT))
allocating, storage 250
altering storage 41
alternate path support 102-102.1
ASSGN command 288-288.1
assigning, dedicated channels to virtual machine 79-80
ATTACH macro 276
attached processor
abnormal termination, dump 45-46
examine real storage 184
improving performance of (5748-XE1) 102.2-102.3
special code in CP 178
TOD clock 176
virtual machine I/O management 79-80
attaching, virtual devices 79-80
AUTHORIZE, VMCF subfunction 153

AUTOLOG command, journaling 229
auxiliary directories
  creating 319
    example 321-322
  error handling 321
  establishing linkage 320-321
  generating 319
  initializing 319-320
  initializing (5748-XX8) 320
  initializing (5748-XE1) 320
  saving resources 319

B
BACKSPAC command, 3800 printer 228
BALRSAVE (BAL register save area) 19,49
batch, facility (see CMS Batch Facility)
BATEXIT1 317
BATEXIT2 317
BATLIMIT 317
BDAM
  restrictions on 280
  support of 278-279
BEGIN command, summary 32
BLDL macro 274
blocks
  control
    CMS 70
    CP 51
BPAM, support of 278-279
BSAM/QSAM, support of 278-279
BSP macro 277
buffers
  forms control 216-217
  print 216-217

C
calculating, dispatching priority 84-85
CANCEL, VMCF subfunction 154
CAW
  operand, of DISPLAY command 34
  subcommand, of DEBUG command 34
CAW (Channel Address Word), displaying 34
CHANGE command, 3800 printer support 227
Channel Address Word (see CAW (Channel Address Word))
channel check 23
channel program, modification 194-195
Channel Status Word (see CSW (Channel Status Word))
CHAP macro 276
character arrangement tables, 3800 printer 226-228
character modification, 3800 printer 226-228
CHECK macro 277
CHKPT macro 276
class
  device 60-62
  privilege 82
clock, comparator 177
CLOSE
  command
    usage 23,44-45
CLOSE/TCLOSE macros 275
CMNDLINE (command line) 71
CMS (Conversational Monitor System) (see also virtual machines)
  ABEND macro 20
  abnormal termination 8-13,15-17
    collect information 70-72
    messages 4-6
    procedure 19-23,21-22,69
    reason for 69
    recovery 21-22
  auxiliary directories 319
  Batch Facility (see CMS Batch Facility)
  called routine table 264
  command language 233
  command processing 261-262
  commands (see CMS commands)
  control blocks, relationships 70
  development facilities 235
  devices supported 242
  DEVTAB (Device Table) 241
  display PSW 22
  DMSABN macro description 20-21
  DMSFREE 241-243
    free storage management 247-250
    macro description 247-250
    service routines 252-255
  DMSFRES macro description 252-255
  DMSFRET macro description 250-251
  DMSFST macro description 319
  DMSITS 256,264-267
  DMSNUC 241-243
  dump
    at abnormal termination 66-68
    examine low storage 69
    format 66-68
    message 69
    register usage 72
  examine low storage 22
  file system 234-235
  free storage management 244-246
    DMSFREE 247-250
    GETMAIN 244-246
  function table 269
    reserved names 269
  functional information 240
  Halt Execution (HX) 20-21
  how to approach problem 3
  how to save it 313
  interface with display terminals 267-268
  interrupt handling 236-239
  introduction 231-233
  load map 22,66-68
  loader tables 243
  low storage 22
  nucleus 243
  nucleus load map 66-68
  program, exception 20
  register usage 72,240
  returning to calling routine 264-267
  sample load map 66-68
  saved system restrictions 315
  simulation of DOS/VS functions 285
  storage
    dump 22,69
    map 245
    structure 241-243

structure of DMSNUC 240
SVC handling 256,264-267
symbol references 240
system, abend 20-21
system save area modification 264-267
transient area 242,263
transient area (5748-XX8) 242.1
transient area (5748-XE1) 242.1
user
area 263
program area 243
USERSECT (User Area) 241
CMS Batch Facility
BATEXIT1 317
BATEXIT2 317
BATLIMIT MACRO file 316
data security 318
EXEC procedures 318
installation input 317
/JOB control card 317
remote input 316
system limits 316
resetting 316
user control cards 317
CMS blip facility (5748-XX8) 85-86
CMS blip facility (5748-XE1) 86
CMS commands 64.2-66
ACCESS 281
ASSGN 288-288.1
DDR 24-25
DEBUG 20-21,64.2-66
FILEDEF 45-46,281-282
GENDIRT 319-320
how to add one 269
LISTFILE 66-68
MODMAP 66-68
MOVEFILE 45-46
PRINT 66-68
SVCTRACE 29-30,35-36,64.2-66
VMFDUMP 45-46
ZAP 41
CMSCB (OS control blocks) 71
CMS/DOS
command summary 286-287
considerations for execution 309
control blocks used by 305-306
DOS/VS volumes needed 308-309
environment, defined 285
generating 306-307
library volume directory entries 307
performance 309
restrictions 309
storage requirements 308
support
DOS/VS macros under CMS 291-293
for declarative macros 294
for DTFCD macro 294-296
for DTFCN macro 296
for DTFDI macro 297
for DTFMT macro 298-300
for DTFPR macro 300-301
for DTFSD macro 302-304
for EXCP 305-306
for imperative macros 304
for transient routines 304-305
hardware devices 285
of DOS/VS functions 285
of DOS/VS supervisor and I/O macros 284
of physical IOCS macros 284
tape label processing (5748-XX8) 309
tape label processing (5748-XE1) 309
user responsibilities 305-306
CMSDOS discontiguous saved segment 135-136
CMSSEG
discontiguous saved segment 313
usage options 314
coding conventions
addressing 211-213
constants 211-213
CP 211-213
error messages 211-213
loadlist requirements 214
module names 211-213
register usage 211-213
command
language
CMS 233
RSCS 330
processing, RSCS 337-338
summary, RSCS 331
commands (see CMS commands, CP commands and RSCS commands)
common segment facility (5748-XE1) 102.1-102.2
communication, between virtual machines 143
COMND macro 215
compiler input/output assignments 289
completion code X'00B' 86
completion code X'00B' (5748-XX8) 86.1
completion code X'00B' (5748-XE1) 86.1
console, function (see CP (Control Program))
control
blocks
locating 40
used by CMS/DOS routines 305-306
registers, displayed by DISPLAY command 33
Control Program (see CP)
control tables
3800 printer
creating and modifying 227-228
displaying current values 228
Conversational Monitor System (see CMS)
copy modification, 3800 printer 226-228
COPYV command, for MSS volumes 174
CP (Control Program)
abnormal termination 15-17
messages 4-6
procedure 18-19,19,45-46
with automatic restart 8-13
without automatic restart 8-13
attached processor mode 178
coding conventions 211-213
commands (see CP commands)
concurrent execution of virtual machines 76
console functions, how to add one 215
control block relationships 51
debugging CP on virtual machine 42-43
disabled loop 8-13
procedure 25-26

disabled wait 8-13
procedure 15-17,27-29
dump
at abnormal termination 45-46
attached processor 45-46
examine abend code 45-46
examine low storage 45-46
format 45-46
on disk 45-46
on printer 45-46
on tape 45-46
printing disk dump 45-46
printing tape dump 45-46
VMFDUMP command usage 45-46
enabled wait 8-13
procedure 15-17,29
errors encountered by warmstart program 4-6
examine low storage 19
how to approach problem 3
identifying and locating pageable module 63-64.1
internal trace table 19,44,47-48
(see also CP trace table)
I/O management on virtual machine 79-80
load map 19
looping condition 15-17
low storage 19
machine check 19
page zero handling 77-78
privileged instruction simulation 76
problem state execution 76
program check 18
in checkpoint program 4-6
in dump program 4-6
PSA, Prefix Storage Area 19
real control blocks 19
register usage 48
restrictions 24-25
RMS (Recovery Management Support) 86
RMS (Recovery Management Support) (5748-XX8) 86.1
RMS (Recovery Management Support) (5748-XE1) 86.1
save areas 48-49
small CP option (5748-XX8) 78.1
spooling 80-82
storage dump 18,45-46
SVC 0 18
SYSTEM RESTART button 19,29
trace table entries 44
(see also CP trace table)
unexpected results 8-13,15-17
procedure 24-25
virtual control blocks 19
virtual machine interrupt handling 76
wait state status messages 4-6
CP assist 99-100.1
CP commands 82
ADSTOP 32,40
CLOSE 23,44-45
DCP 38-42
DISPLAY 22,26-27,33,38-42
DMCP 38-42
DUMP 26-27,29-30,32,38-42
how to add command 215
INDICATE 103-109
E privilege class 103-109
G privilege class 103-109
INDICATE FAVORED
E privilege class (5748-XX8) 109
E privilege class (5748-XE1) 109
IPL 314
LOCATE 40
MIGRATE command (5748-XE1) 109
MONITOR 110-125
MONITOR (5748-XX8) 110.1
MONITOR (5748-XE1) 110.1
QUERY 39
SET 39,44-45
STCP 41
STORE 34-35,41
SYSTEM 39
TRACE 23,24-25,26-27,29-30,35-36,40-41
CP trace table 19
allocation 44
clear channel instruction 44
entries 44
restarting tracing 44
size 44
terminating tracing 44
usage 44,47-48
CPABEND (abend code) 45-46
CPEREP program 19
CPSTAT (CP running status) 47-48
CSW
operand, of DISPLAY command 34
subcommand, of DEBUG command 34
CSW (Channel Status Word), displaying 34
CVTSECT (CMS Communications Vector Table) 72
cylinder faults, MSS, VM/370 processing 173

D
DASD Dump Restore (DDR) program 24-25
DASD I/O function 190-191
data
records, VM Monitor 357-365
security, batch 318
data set control block (DSCB) 278-279
data sets
OS
accessing 281
defining 281-282
reading 280.1-281
VSAM
compatibility considerations 312-313
compatibility considerations (5748-XX8) 310-312.1
compatibility considerations (5748-XE1) 310-312.1
DCB macro 277
DCP, command 38-42
DDR command, usage 24-25
deadline priority
definition (5748-XX8) 85-86
definition (5748-XE1) 85-86
dispatch list (5748-XX8) 85-86
dispatch list (5748-XE1) 85-86
eligible list (5748-XX8) 85-86
eligible list (5748-XE1) 85-86

deadline priority (5748-XX8) 85-86
DEBUG command
  BREAK subcommand, summary 32
  CAW subcommand, summary 34
  CSW subcommand, summary 34
  DUMP subcommand
    summary 32
    usage 26-27
  GO subcommand, summary 32
  GPR subcommand, summary 33
  messages 64.2-66
  PSW subcommand
    summary 33
    usage 22
  rules for using 64.2-66
  SET CAW subcommand, summary 35
  SET CSW subcommand, summary 35
  SET GPR subcommand, summary 34-35
  SET PSW subcommand, summary 35
  STORE subcommand, summary 34-35
  usage 20-21
  X (Examine) subcommand, summary 33
debugging
  analyzing problem 13
  applying PTF 14
  comparison of CP and CMS facilities 37
  how to start 3,15-17
  identifying
    abnormal termination 15-17
    looping condition 15-17
    looping condition in virtual machine 7
    problem 7
    unexpected results 15-17
    wait 15-17
    wait state in virtual machine 7
  introduction 1-2
  on virtual machine 23
  procedure
    for abnormal termination 15-17
    for CMS abnormal termination 19-23
    for CP abend without dump 19
    for CP abnormal termination 18-19
    for CP disabled loop 25-26
    for CP disabled wait 27-29
    for CP enabled wait 29
    for CP unexpected results 24-25
    for looping condition 15-17
    for RSCS disabled wait 30-31
    for unexpected results 15-17
    for virtual machine abnormal termination 22
    for virtual machine disabled loop 26-27
    for virtual machine disabled wait 29-30
    for virtual machine enabled loop 26-27
    for virtual machine enabled wait 30
    for virtual machine unexpected results 24-25
    for wait 15-17
  recognizing problem 4
  summary of VM/370 debugging tools 32-36
  unproductive processing time 7
  VM/370 commands 38-42
    ADSTOP 40
    DCP 38-42
    DISPLAY 38-42
    DMCP 38-42
    DUMP 38-42
    LOCATE 40
    MONITOR 40
    QUERY 39
    SET 39
    STCP 41
    STORE 41
    SYSTEM 39
    TRACE 40-41
    ZAP 41
  with VM/370 facilities 8-31
declarative macros 294
dedicated, channel, assigning to virtual machine 79-80
DELETE macro 274
demand paging 76-77
DEQ macro 276
DETACH, macro 276
detaching, virtual devices 79-80
determining, virtual machine storage size 201
DEVICE (last I/O interrupt) 22
devices
  class codes 60-62
  CMS-supported 242
  feature codes 60-62
  model codes 60-62
  sense information 86
  sense information (5748-XX8) 86.1
  sense information (5748-XE1) 86.1
  supported, for VSAM under CMS 310
  type codes 60-62
DEVTAB (Device Table) 241
DEVTYPE macro 275
DIAGNOSE instruction 182
  activating the TOD-clock accounting interface (5748-XE1) 204-204.1
  channel program modification 194-195
  clean-up after virtual IPL by device (5748-XX8) 197
  clean-up after virtual IPL by device (5748-XE1) 197
  clear I/O recording 191
  DASD I/O function 190-191
  define function of PA2 function key 199
  determine virtual machine storage size 138-139,201
  device type and features 192-194
  directory update in-place 206-210
  display data on 3270 console screen 199-200
  error message editing 200-201
  error message editing (5748-XX8) 200.4
  error message editing (5748-XE1) 200.4
  examine real storage 184
  find address of discontiguous saved segment 138-139
  FINDSYS function 138-139,202
  general I/O function 191-192
  generate accounting cards 197-198.2
  generate accounting records for the virtual user (5748-XX8) 198
  generate accounting records for the virtual user (5748-XE1) 198
  input spool file manipulation 187-190

issue SVC 76 from a second level virtual machine 197
load discontiguous saved segment 138-139
LOADSYS function 138-139,201-202
MSS communication 205-206
MSS mount and demount processing 173
page release function 186
pseudo timer 186
purge discontiguous saved segment 138-139
PURGESYS function 138-139,202
read LOGREC data 196
read system dump spool file 196-197
read system symbol table 197
save 3704/3705 control program 199
saving or loading a 3800 named system 204-205
saving or loading a 3800 named system (5748-XE1) 204.1-205
special diagnose for shadow table maintenance (5748-XE1) 204
start of LOGREC area 195-196
store extended-identification code 183
update user directory 197
virtual console function 184-186
VMCF function 143,159,203-204
data transfer error codes 167-169
return codes 167-169
VMCPARM parameter list 159-162
3270 virtual console interface 199-200
full screen interactions (5748-XX8) 199-200.4
full screen interactions (5748-XE1) 199-200.4
full screen mode (5748-XX8) 199-200.4
full screen mode (5748-XE1) 199-200.4

directory
entries for CMS/DOS library volumes 307
hooks 206-210
replacing entries 206-210
update in-place 206-210

discontiguous saved segments 135-136
loading 201-202
purging 202

discontiguous shared segments
defined via NAMESYS macro 137-138
user requirements 136

dispatch list
use in deadline priority (5748-XX8) 85-86
use in deadline priority (5748-XE1) 85-86

dispatching
interactive users 84-85
noninteractive users 84-85
priority, calculating 84-85
scheme, for virtual machines 84-85
virtual machines
from queue 1 84-85
from queue 2 84-85

dispatching priority, replacing directory entry 206-210

DISPLAY
command 38-42
summary 33
usage 22,26-27

display terminals, CMS interface 267-268

displaying
data on 3270 console screen 199-200
floating-point registers, DISPLAY command 33
general registers
DISPLAY command 33
GPR subcommand of DEBUG command 33
PSW
DISPLAY command 33
PSW subcommand of DEBUG command 33
storage
DISPLAY command 33
X subcommand of DEBUG command 33

DISPSW macro display terminals, DISPSW macro 267-268

distribution word, replacing directory entry 206-210

DMCP, command 38-42

DMKCFC (console function) support 215

DMKDDR (see DASD Dump Restore (DDR) program)

DMKSNT (system name table) 132-142

DMSABN (abend routine) 71

DMSABN macro 20-21
operands 20-21

DMSEXS 255-256

DMSFREE 241-243
allocating nucleus free storage 250-251
allocating user free storage 250
error codes 254-255
operands 247-250
service routines 252-255
storage management 247-250

DMSFRES 252-255
error codes 254-255
operands 252-255

DMSFRET 252
error codes 254-255
operands 252
releasing storage 252

DMSINA 259

DMSINT 259

DMSIOW 238

DMSITE 239

DMSITI 236-237

DMSITP 238

DMSITS 236-237,256,264-267

DMSKEY 255-256

DMSLADAD, entry for auxiliary directory 320-321

DMSNUC 240,241-243

DOS (Disk Operating System)
abnormal termination
messages 4-6
procedure 23

DOS/VS
functions simulated by CMS 285
macros
supervisor 289-290
supported under CMS 291-293

DOS/VSE
  CMS support
    control blocks simulated (5748-XX8) 305-306
    functions supported (5748-XX8) 285
    hardware supported (5748-XX8) 285
    supervisor and I/O macros supported (5748-XX8) 289-290
    VSAM macros supported (5748-XX8) 310
DOS/VSE CMS support
  control blocks simulated (5748-XE1) 305-306
  functions supported (5748-XE1) 285
  hardware supported (5748-XE1) 285
  supervisor and I/O macros supported (5748-XE1) 289-290
  VSAM macros supported (5748-XE1) 310-312.1
DOS/VSE transient routines (5748-XE1) 304
DSCB 278-279
DTFCD macro 294-296
DTFCN macro 296
DTFDI macro 297
DTFMT macro 298-300
DTFPR macro 300-301
DTFSD macro 302-304
dump (see also CP (Control Program), dump and CMS (Conversational Monitor System), dump)
DUMP
  command 38-42
    summary 32
    usage 26-27,29-30
dump, used in problem determination 18
dumping
  storage
    at printer 37
    at terminal 37
  to real printer 45-46
DUMPSAVE (DMKDMP save area) 49
dynamic load overlay 325
dynamic SCP transition to or from native mode
  advantages of (5748-XE1) 102.3-102.4
  command used for (5748-XE1) 102.3-102.4
  overview of how to use (5748-XE1) 102.3-102.4
  performance impact of (5748-XE1) 102.3-102.4
  precautions when using (5748-XE1) 102.3-102.4
  purpose of (5748-XE1) 102.3-102.4
  systems supported (5748-XE1) 102.3-102.4

E
EC (Extended Control) mode 26-27
EC (Extended Control) PSW 348
ECMODE option 176
ECPS (Extended Control-Program Support) 99-100.1
  CP assist 99-100.1
  expanded virtual machine assist 99-100.1
  restricted use 101-102
  virtual interval timer assist 101,175-176
ECRLOG (control registers) 71
editing
  error messages 200-201
  error messages (5748-XX8) 200.4
  error messages (5748-XE1) 200.4
efficiency, of VM/370 performance options 88
eligible list
  use in deadline priority (5748-XX8) 85-86
  use in deadline priority (5748-XE1) 85-86
ENQ macro 276
environment, of VM/370, system load 124-125
error codes 254-255
  DMSFREE 254-255
  DMSFRES 254-255
  DMSFRET 254-255
error messages
  editing 200-201
  editing (5748-XX8) 200.4
  editing (5748-XE1) 200.4
error recording cylinders, clear 191
EXCP, CMS/DOS support for 305-306
expanded virtual machine assist 99-100.1
Extended Control mode (see EC (Extended Control) mode)
Extended Control-Program Support (ECPS) (see ECPS)
extended-identification code 183
external interrupt
  BLIP character 239
  external console interrupt 87
  HNDEXT macro 239
  in CMS 239
  in RSCS 338
  in VMCF 143,164-167
    message header 164-167
  interval timer 87
  timer 239
EXTOPSW (external old PSW) 69
EXTRACT macro 275
EXTSECT (external interrupt work area) 71

F
faults, MSS cylinder, VM/370 processing 173
favored execution option 93-94
FCB (see forms control buffer)
FCB (File Control Block) 240
FCBTAB (file control block table) 71
features, device 60-62
FEOV macro 275
fetch storage protection 77-78
file
  management
    CMS 234-235
    RSCS 336

File Status Table 319
FILEDEF command 282-283
  AUXPROC option 282-283
  defining OS data sets 281-282
  usage 45-46
files, OS format, support of 278-279
FIND, macro 274
finding
  address of discontiguous saved segment 138-139
  saved systems 202
flashing (forms overlay), 3800 printer 226-228
FOB (font offset buffer)
  for 3289 Model 4 printer
    adding FOBs (5748-XX8) 222
    adding FOBs (5748-XE1) 222-225
    macro instruction (5748-XX8) 222
    macro instruction (5748-XE1) 222-225
    names of supplied FOBs (5748-XX8) 222
    names of supplied FOBs (5748-XE1) 222
    purpose of FOB (5748-XX8) 222
    purpose of FOB (5748-XE1) 222-225
  for 3289 Model 4 printer (5748-XX8) 222
  for 3289 Model 4 printer (5748-XE1) 222-225
font offset buffer (see FOB)
forms control buffer
  FCB 216-217
    examples 222-223
    examples (5748-XX8) 222.1
    examples (5748-XE1) 222-225
    macro 222-223
    macro (5748-XX8) 222.1
    macro (5748-XE1) 222-225
  index feature 222-223
    example 222-223
  index feature (5748-XX8) 222.1
  index feature (5748-XE1) 222-225
  3800 printer 226-228
forms overlay (flashing), 3800 printer 226-228
FPRLOG (floating-point registers) 71
free storage
  management
    CMS 244-246
    RSCS 333
FREEDBUF macro 276
FREEMAIN macro 273
FREEPOOL macro 274
FREESAVE (DMKFRE register save area) 19,49

G
GENDIRT command
  creating auxiliary directory 319-320
  format 320-321
generating, CMS/DOS 306-307
GENIMAGE utility program 227-228
GET macro 278-279
GETMAIN
  free element chain 245
  GETMAIN/FREEMAIN macros 274
  macro 273
  simulation 245
GETPOOL macro 274
GPRLOG (general registers) 71

H
handling
  OS files
    on CMS disks 270
    on OS or DOS disks 271
hardware assist 99-100.1
header, record, VM Monitor 357-365

I
IBM 3800 Printing Subsystem (see 3800 printer)
identification bits for program products (5748-XE1) 183
IDENTIFY
  VMCF protocol 152
  VMCF subfunction 155
IDENTIFY macro 276
IIP (ISAM Interface Program) 312-313
IIP (ISAM Interface Program) (5748-XX8) 312.1
IIP (ISAM Interface Program) (5748-XE1) 312.1
IMAGELIB utility program 227-228
imperative macros 304
INDICATE command 103-109
  described 103-109
  format
    E privilege class 103-109
    G privilege class 103-109
INDICATE FAVORED command
  format
    E privilege class (5748-XX8) 109
    E privilege class (5748-XE1) 109
indicators, of system load 103-109
input/output (see I/O)
interrogating input/output assignments 289
interrupt handling 85-86
  attached processor
    real I/O interrupts 87
    synchronous interrupts 87
  CMS 236-239
    input/output interrupts 237
    SVC interrupts 236
    terminal interrupts 238
  DMSITS 236-239
  external interrupts 87,239
  I/O interrupts 79-80
  machine check interrupts 86,239
  machine check interrupts (5748-XX8) 86.1
  machine check interrupts (5748-XE1) 86.1
  program interrupts 86,238
  program interrupts (5748-XX8) 86.1
  program interrupts (5748-XE1) 86.1
  reader/punch/printer interrupts 238
  RSCS 338
  SVC interrupts 87
  user-controlled device interrupts 238
interrupt handling (5748-XX8) 86.1
interrupt handling (5748-XE1) 86.1

interval timer 101,175-176
INTSVC 256
I/O
  assignments
    compiler 289
    interrogating 289
  function
    DASD 190-191
    general 191-192
  in attached processor mode 179
  interrupt
    in CMS 237
    in RSCS 339
  logging, RSCS 340
  management 79-80
  overhead in CP, reducing 89-90
  virtual machines 89-90
I/O errors, recovery from 228
ICBLOK 19
IOSECT (I/O interrupt work area) 72
IPL, of a VMSAVE system (5748-XE1) 78.2
IPL command, loading alternate saved segment 314
IPL device, replacing directory entry 206-210
issuing, SVC 76, from a second level virtual machine 197

J
journaling
  accounting records 128-129
  LOGON, AUTOLOG, LINK commands 229

L
LASTCMND (last command) 22,71
LASTEXEC, last EXEC procedure 71
LASTEXEC (last EXEC procedure) 22
LASTLMOD (last module loaded) 22,71
LASTTMOD (last module in transient area) 71
LASTTMOD (last transient loaded) 22
library volumes, CMS/DOS, directory entries for 307
LINK, macro 273
LINK command
  journaling 229
  password suppression 230
LIOCS routines supported by CMS/DOS 304-305
LISTFILE command 66-68
load
  environments of VM/370 124-125
  indicators 103-109
LOAD, macro 274
load map
  CMS 66-68
  how to print CMS load map 66-68
loader tables, (CMS) 243
loading
  and saving discontiguous saved segments 138-139
  discontiguous saved segments 138-139,201-202
loadlist
  requirements
    CP 214
    SPB card 214
LOCATE, command 40
locked pages option 91-92
locking, in attached processor mode 180-181
LOCKSAVE (LOCK macro save area) 50
LOG record
  NPT 342
  SML 341
logical editing symbols, replacing directory entry 206-210
logical units
  assignment of 288-288.1
  defined 288-288.1
  programmer assigned 288-288.1
  system assigned 288-288.1
LOGON command
  journaling 229
  password suppression 230
LOGREC area
  getting starting address 195-196
  reading 196
LOKSAVE (DMKLOK save area) 50
loop 25-26
  (see also problem types)
  disabled
    CP 25-26
    virtual machine 26-27
  enabled, virtual machine 26-27
low address protection (5748-XE1) 102.1-102.2
LOWSAVE (debug save area) 71
LUB (Logical Unit Block) table 288-288.1

M
machine check
  CP 19
  during start-up 86
  during start-up (5748-XX8) 86.1
  during start-up (5748-XE1) 86.1
  interrupt 86
    in CMS 239
  interrupt (5748-XX8) 86.1
  interrupt (5748-XE1) 357-365
  not diagnosed 19
  on attached processor 23
  unrecoverable 19
macros
  declarative 294
  DOS/VSE macros supported by CMS/DOS (5748-XX8) 289-290
  DOS/VSE macros supported by CMS/DOS (5748-XE1) 289-290
  imperative 304
  OS (see OS (Operating System), macros)
  VSAM, supported under CMS 311-312
Mass Storage System (see MSS)
MCKOPSW (CMS machine check old PSW) 69
messages, controlling 40
MFASAVE (DMKMCT save area) 50
MIGRATE command (5748-XE1) 109
minidisk link mode, replacing directory entry 206-210

minidisk multiple password, replacing
directory entry 206-210
minidisk read password, replacing directory
entry 206-210
minidisk write password, replacing
directory entry 206-210
minidisks 79-80
model, device 60-62
modifying modules 41-42
MODMAP command 66-68
MONITOR command 110-125
described 110-125
described (5748-XX8) 110.1
described (5748-XE1) 110.1
format 110-125
implemented classes 119-121
monitoring, recommendations 124
MOVEFILE command, usage 45-46
MSS (Mass Storage System)
communication 205-206
cylinder faults, VM/370 processing 173
mount and demount processing
173,205-206
mount processing, asynchronous 173
VM/370 access 173
volumes
backup copies 174
I/O management 79-80,89-90
MULTI-LEAVING
block control byte (BCB) 351-356
character string 349
control fields
record control byte (RCB) 351-356
string control byte (SCB) 351-356
sub-record control byte (SRCB)
351-356
description of 349
function control sequence (FCS) 351-356
in VM/370 349
transmission block 350-351
multiple channel errors 6
multiple copy printing, 3800 printer
226-228
multiple shadow table support (5748-XE1)
98-98.1
multiprocessing systems, improving
performance of (5748-XE1) 102.2-102.3
MVS/system extensions support
common segment facility (5748-XE1)
102.1-102.2
enabling (5748-XE1) 102.1-102.2
low address protection (5748-XE1)
102.1-102.2
special operations and instructions
(5748-XE1) 102.1-102.2

N
named systems
allocating DASD space 132-142
generating 132-142
SPB card 132-142
using NAMESYS macro 132-142
saved system 132-142
SAVESYS command 134
saving or loading a 3800 204-205
saving or loading a 3800 (5748-XE1)
204.1-205
shared segments 135
system name table (DMKSNT) 132-142
NAMESYS macro
discontiguous saved segments 137-138
for saved systems 132-142
native mode, switching to or from
(5748-XE1) 102.3-102.4
NOTE macro 277
NPT LOG record 342
nucleus (CMS) 243
NUCON (nucleus constant area) 70-72

O
OPEN/OPENJ macros 275
options
performance
affinity 97-98
favored execution 93-94
locked pages 91-92
priority 95
reserved page frames 92,95
small CP (5748-XX8) 78.1
virtual=real 77-78,92-93,95-97
virtual machine 93
OS (Operating System)
abnormal termination
messages 4-6
procedure 23
data management simulation 270
data sets, reading 280.1-281
formatted files 278-279
handling
files on CMS disks 270
files on OS or DOS disks 271
macros
ABEND 275
ATTACH 276
BLDL 274
BSP 277
CHAP 276
CHECK 277
CHKPT 276
CLOSE/TCLOSE 275
DCB 277
DELETE 274
DEQ 276
descriptions of 272-277
DETACH 276
DEVTYPE 275
ENQ 276
EXTRACT 275
FEOV 275
FIND 274
FREEDBUF 276
FREEMAIN 273
FREEPOOL 274
GET 278-279
GETMAIN 273
GETMAIN/FREEMAIN 274
GETPOOL 274
IDENTIFY 276
LINK 273
LOAD 274

NOTE 277
OPEN/OPENJ 275
POINT 277
POST 273
PUT 278-279
PUTX 278-279
RDJFCB 276-277
READ 278-279
SNAP 276
SPIE 274
STAE 276
STAX 277
STIMER 276
STOW 275
SYNADAF 277
SYNADRLS 277
TCLEARQ 277
TGET/TPUT 277
TIME 275
TTIMER 276
under CMS 270
WAIT 273
WRITE 278-279
WTO/WTOR 275
XCTL 274
XDAP 273
overhead, CP, reducing for I/O 89-90
overlay structures under CMS 323
overlaying
dynamic load 325
example 324
prestructured 323-325

P
page
allocation, RSCS 335
exceptions, effects of 90-91
frames 76-77
reserved 77-78,92
locking 91-92
releasing
contiguous storage 186
discontiguous storage 202
SPB (Set Page Boundary) card 214
table 76-77
zero, restrictions 77-78
pageable module
identifying 63-64.1
locating 63-64.1
paging 76-77
by demand 76-77
considerations 90-91
password
replacing directory entry 206-210
suppression on command line 230
PA2 program function key, defining function of 199
performance 88
avoiding IPL 132-142
CMS/DOS 309
for mixed mode foreground/background systems 126
for time-shared multi-batch virtual machines 126
measurement 103
options
affinity 97-98
dynamic SCP transition to or from native mode (5748-XE1) 102.3-102.4
favored execution 93-94
locked pages 91-92
priority 95
reserved page frames 92,95
single processor mode (5748-XE1) 102.2-102.3
small CP (5748-XX8) 78.1
virtual=real 77-78,92-93,95-97
virtual machine 93
PGMOPSW (program old PSW) 69
PGMSECT (program check interrupt work area) 72
PLIST (parameter list) 240
POINT macro 277
POST macro 273
Prefix Storage Area (see PSA (Prefix Storage Area))
prestructured overlays 323-325
PREVCMND (previous command) 22,71
PREVEXEC (previous EXEC procedure) 22,71
print buffers
adding new images 217-218
LOADBUF command 217-218
print chain image 217-218
UCB macro 218-219
UCBCCW macro 220
UCC, examples 224-225
UCC macro 224
UCCCCW macro 224
UCS
examples 218-219
macro 217-218
1403 216-217
UCSB
associative fields 220
examples 221-222
3211 216-217
UCSCCW macro 218
PRINT command 66-68
printer
IBM 3800 (see 3800 printer)
interruptions 238
priority
messages 153,156-157,157
of execution 76-77
performance option 95
privilege classes 82
replacing directory entry 206-210
privileged instructions 88
problem
programs, unexpected results 15-17
types
abnormal termination 8-13
loop 8-13
unexpected results 8-13
wait 8-13
processor
attached
affinity 97-98,181
I/O handling 179
locking 180-181
machine check 23
PSA 178
real I/O interrupts 87

signaling 180
synchronous interrupts 87
resources 84-85
timer 176
utilization 84-85
program
check
in checkpoint program 4-6
in dump program 4-6
interruption
in CMS 238
problem state 86
problem state (5748-XX8) 86.1
problem state (5748-XE1) 86.1
supervisor state 86
supervisor state (5748-XX8) 86.1
supervisor state (5748-XE1) 86.1
states 83
program product identification bits (5748-XE1) 183
Program Status Word (see PSW (Program Status Word))
programmer logical units 288-288.1
PROPSW (program old PSW) 45-46
protection keys 77-78
protection of shared segments 140-142
PSA (Prefix Storage Area) 19
ARIOCH (address of first RCHBLOK) 56
ARIOCU (address of first RCUBLOK) 57
ARIODV (address of first RDEVBLOK) 57-59
in attached processor mode 178
pseudo timer 178,186
PSW (Program Status Word) 47-48
interruption code 22
keys, CMS 255-256
PTFs (program temporary fixes), applying 3,14
PUB (Physical Unit Block) table 288-288.1
punch, interruptions 238
purging, discontiguous saved segment 138-139,202
PUT macro 278-279
PUTX macros 278-279

Q
QUERY, command 39,286-287
QUERY command, 3800 printer 228
QUERY SRM command (5748-XX8) 109
QUERY SRM command (5748-XE1) 110
querying and setting paging variables (5748-XX8) 110
querying and setting paging variables (5748-XE1) 110
querying and setting SRM variables (5748-XX8) 110
querying and setting SRM variables (5748-XE1) 110
queue 1, dispatching virtual machines from 84-85
queue 2, dispatching virtual machines from 84-85
queue 3 (5748-XX8) 85-86
queue 3 (5748-XE1) 86
QUIESCE, VMCF subfunction 154
Q1 (see queue 1)
Q2 (see queue 2)
Q3 (see queue 3) (5748-XE1)
Q3 (see queue 3) (5748-XX8)

R
RCHBLOK 56
RCHADD (address) 56
RCHFIOB (first IOBLOK pointer) 56
RCHSTAT (status) 56
RCHTYPE (type) 56
RCUBLOK 57
RCUADD (address) 57
RCUCHA (primary RCHBLOK) 57
RCUCHB (first alternate RCHBLOK) 57
RCUCHC (second alternate RCHBLOK) 57
RCUCHD (third alternate RCHBLOK) 57
RCUFIOB (first IOBLOK pointer) 57
RCULIOB (last IOBLOK pointer) 57
RCUSTAT (status) 57
RCUTYPE (type) 57
RDEVBLOK 57-59
RDEVADD (address) 57-59
RDEVAIOB (IOBLOK pointer) 57-59
RDEVATT (attached virtual address) 57-59
RDEVCKPT (address of enable CKPBLOK) 57-59
RDEVEPDV (address of EP free list) 57-59
RDEVFLAG (device dependent flags) 57-59
RDEVIOER (address of IOERBLOK) 57-59
RDEVMAX (highest valid NCP name) 57-59
RDEVNCP (reference name of active 3705 NCP) 57-59
RDEVNICL (address of network control list) 57-59
RDEVSPL (RSPLCTL pointer) 57-59
RDEVSTAT (status) 57-59
RDEVTFLG (flags) 57-59
RDEVTMCD (terminal flags) 57-59
RDEVTYPC (class) 57-59
RDEVUSER (dedicated user) 57-59
RDJFCB macro 276-277
READ macro 278-279
reader, interruptions 238
reading, OS data sets 280.1-281
real, printer dumping to 45-46
real storage
examine 184
in attached processor environment 184
optimizing use of 76-77
REALTIMER option 175-176
RECEIVE, VMCF subfunction 158
recording, real machine system events 40
records
accounting
created by user 130-130
for AUTOLOG, LOGON, and LINK journaling 128-129
format for dedicated devices 127-131
format for dedicated devices (5748-XX8) 126
format for dedicated devices (5748-XE1) 127-131
format for virtual machines 127-131

RECOVERV command, for MSS volumes 174
reduction
  of CP overhead, for virtual machine I/O 89-90
  of paging activity 90-91
  of SIO operation 89-90
reenterable code, usage 90-91
registers, usage, CMS 240
REJECT, VMCF subfunction 155
releasing
  allocated storage 252
  storage 252
Remote Spooling Communications Subsystem (see RSCS (Remote Spooling Communications Subsystem))
REPLY, VMCF subfunction 158
RESERVE, operand 77-78
reserved page frames 77-78
  performance option 92,95
resources, processor 84-85
responses, VM Monitor, to unusual tape conditions 121-122
responsibilities, user, for CMS/DOS 305-306
restrictions
  BDAM 280
  CMS, saved system 315
  CMS/DOS 309
  multiple path support 44
resume
  execution
    BEGIN command 32
    GO subcommand of DEBUG command 32
RESUME, VMCF subfunction 155
RSCS (Remote Spooling Communications Subsystem)
  command language 330
  command processing 337-338
  command summary 331
  disabled wait 8-13
    procedure 30-31
    X'001' 30-31
    X'007' 30-31
    X'011' 30-31
  DMTMAP 333
  DMTVEC 333
  enabled wait 8-13,31
  external interrupts 338
  file management 336
  free storage 333
  functional information 335
  interrupt handling 338
  I/O
    interrupts 339
    logging activity 340
    logging output 340
    logging record 340
  line allocation task 334
  line driver storage 334
  links 329
    definition 329
    table 329
  locations 329
  message handling 338
  nonprogrammable remote terminals 329
  page allocation 335
  programmable remote stations 329
  queue element management 335
  remote stations 329
  spool file
    access 337
    access task 334
  storage
    allocation 331
    structure 331
  supervisor 333
    service routines 333
  supervisor queue 333
    extension 333
  SVC interrupts 339
  system control task 334
  tag slot queues 336
  task-to-task communications 337
  virtual storage management 335
  VM/370 spool system interface 330
RUNUSER (current user) 47-48

S
save area
  BALRSAVE 19,49
  CMS system 264-267
  CMS system save area format 264-267
  DMKLOK 50
  DUMPSAVE 49
  FREESAVE 19,49
  LOCKSAVE 50
  MFASAVE 50
  SAVEAREA 19,49
  SIGSAVE 49
  SVCREGS 50
  SWTHSAVE 50
  user save area format 264-267
SAVEAREA (active save area) 19,49
saved systems
  CMS 312-313
  described 132-142
  SAVESYS command 134
  when to save system 134
  when to save systems (5748-XX8) 132
  when to save systems (5748-XE1) 131
SAVESEQ priority value (5748-XX8) 78.2
SAVESEQ priority value (5748-XE1) 78.2
SAVESYS command 134
saving, storage information 41
segment
  alternate, loading on IPL command 314
  shared (see shared segments)
segment table 76-77
SEND
  VMCF protocol 148-149
  VMCF subfunction 156-157
SEND/RECV
  VMCF protocol 150
  VMCF subfunction 156-157
SENDX
  VMCF protocol 151
  VMCF subfunction 157
SET
  command 39,286-287
    usage 44-45
SET SRM command (5748-XX8) 110
SET SRM command (5748-XE1) 110
SETKEY command, described 138-139
setting, address stops 37

shadow table bypass (5748-XE1) 98-98.1
shared segments 135
  described 135
  discontiguous 135-136
  protected 140-142
  special considerations 135
  unprotected 140-142
  virtual machine operation 142
signaling, in attached processor mode 180
SIGSAVE (DMKEXT save area) 49
simulation 88
  of DOS/VS functions by CMS 285
single processor mode
  advantages of (5748-XE1) 102.2-102.3
  commands used with (5748-XE1) 102.2-102.3
  performance impact of (5748-XE1) 102.2-102.3
  purpose of (5748-XE1) 102.2-102.3
  systems supported (5748-XE1) 102.2-102.3
  use of the V=R machine (5748-XE1) 102.2-102.3
single-instruction mode 82
SIO (see Start I/O (SIO) instruction)
small CP option
  effect on performance (5748-XX8) 78.1
  purpose of (5748-XX8) 78.1
SML LOG record 341
SMSG command, overview of 171-172
SNAP macro 276
spanned records, usage 278-279
SPB (Set Page Boundary) card 214
special diagnose for shadow table maintenance (5748-XE1) 204
special message facility 171-172
  authorization 171-172
  buffer length 171-172
  description 171-172
  introduction to 171-172
  sending or receiving messages 171-172
  SMSG command 171-172
  using 171-172
special message flag
  purpose of 171-172
  turning on or off 171-172
SPIE macro 274
SPOOL command, 3800 printer support 227
spool file
  access, RSCS 337
  manipulation 187-190
  recovery
    after checkpoint start 81
    after force start 82
    after warm start 81
spooling
  described 80-82
  terminal input 82
  terminal output 82
  via RSCS 80-82
STAE macro 276
START command, 3800 printer 227
Start I/O (SIO) instruction
  handling 89-90
  reducing 89-90
STAX macro 277
STCP, command 41
STIMER macro 276
stop execution 40
  ADSTOP command 32
  BREAK subcommand of DEBUG command 32
stop tracing
  SVCTRACE command 35-36
  TRACE command 35-36
storage
  allocation 250
    RSCS 331
  CMS 245
  dump
    CMS 22
    CP 18
  dynamic paging 90-91
  protection
    fetch 77-78
    storing 77-78
  releasing 252
  requirements
    assembler 323
    for CMS support of VSAM 312-313
    for CMS/DOS 308
storage size
  maximum, replacing directory entry 206-210
  virtual machine, replacing directory entry 206-210
STORE command 41
  summary 34-35
storing
  data
    into CAW, SET CAW subcommand of DEBUG command 35
    into control registers, STORE command 34-35
    into CSW, SET CSW subcommand of DEBUG command 35
    into floating-point registers, STORE command 34-35
    into general registers, SET GPR subcommand of DEBUG command 34-35
    into general registers, STORE command 34-35
    into PSW, SET PSW subcommand of DEBUG command 35
    into PSW, STORE command 35
    STORE command 34-35
    STORE subcommand of DEBUG command 34-35
  information 37
  storage protection 77-78
STOW macro 275
STRINIT macro 244-246
structure, of RSCS storage 331
SVC
  handling
    by user 258
    commands entered from terminal 259-260
    invalid SVCs 258-259
    linkage 257-258
    OS and DOS/VS SVC simulation 258
    type of SVC 257-258
  interrupt
    CMS internal linkage SVCs 236
    other CMS SVCs 236-237

problem state 87
supervisor state 87
interrupts, RSCS 339
SVC 202 257-258
search hierarchy 259
SVC 203 257-258
SVC 76, issuing from a second level virtual machine 197
SVCOPSW (SVC old PSW) 69
SVCREGS (SVC interrupt save area) 50
SVCSECT (SVC interrupt work area) 72
SVCTRACE command 64.2-66
summary 35-36
usage 29-30
SWTHSAVE (DMKSTK save area) 50
SYNADAF macro 277
SYNADRLS macro 277
SYSJRL macro instruction 229
system
abend 20
dump spool file, reading 196-197
logical units 288-288.1
performance
for mixed mode foreground/background systems 126
measurement 103
routine, abnormal termination of 20
symbol table, reading 197
SYSTEM command 39
system name table (DMKSNT) 132-142
System/370
control registers
allocation 345
assignments 346-347
extended control (EC) PSW 348
information 345

T
tape label processing in CMS/DOS (5748-XX8) 309
tape label processing in CMS/DOS (5748-XE1) 309
TCLEARQ macro 277
terminal interruptions, in CMS 238
TGET/TPUT macros 277
TIME, macro 275
time management 76
time slice 84-85
time-of-day (TOD) clock 176
in attached processor environment 176
timers
clock comparator 177
interval timer 101,175-176
processor timer 176
pseudo timer 177
Time of Day (TOD) clock 176
TOD-clock accounting interface (5748-XE1) 204-204.1
TRACCURR (current trace table entry) 47-48
TRACE
command 40-41
summary 35-36
usage 23,24-25,26-27,29-30
TRACEND (end of trace table) 47-48
tracing 40-41
all user I/O operations, TRACE command 35-36
branches
TRACE command 35-36
CCWs, TRACE command 35-36
clear channel instruction 44
CP trace table 44
external interrupts, TRACE command 35-36
information 37
instructions
TRACE command 35-36
interrupts 44
TRACE command 35-36
I/O 44
interrupts, TRACE command 35-36
NCP BTU 44
privileged instructions, TRACE command 35-36
program interrupts, TRACE command 35-36
queue drop 44
real machine events, MONITOR command 35-36
run user requests 44
scheduling 44
storage management 44
SVC interrupts
SVCTRACE command 35-36
TRACE command 35-36
user operations, TRACE command 35-36
TRACSTRT (start of trace table) 47-48
transient area 263
transient area (CMS) 242
transient routines supported by CMS/DOS 304-305
TTIMER macro 276
type (device) 60-62

U
UCS (Universal Character Set)
adding buffer images 217-218
supplied images 216-217
UNAUTHORIZE, VMCF subfunction 154
unexpected results 6
(see also problem types)
reason for 24-25
unit record, devices, sharing 80-82
Universal Character Set (see UCS)
Universal Character Set (see UCS)
unproductive processing time 6
user directory
reading 197
updating 197
user doubleword, VMCF function 164-167
user options, replacing directory entry 206-210
user-controlled device interrupts 238
USERSECT (User Area) 241

V
V=R machine, used with single processor mode (5748-XE1) 102.2-102.3
VCHBLOK 53-54
  VCHADD (virtual channel address) 53-54
  VCHSTAT (status) 53-54
  VCHTYPE (type) 53-54
VCUBLOK 53-54
  VCUADD (virtual control unit address) 53-54
  VCUSTAT (status) 53-54
  VCUTYPE (type) 53-54
VDEVBLOK 54-56
  VDEVADD (virtual device address) 54-56
  VDEVCFLG (virtual console flags) 54-56
  VDEVCSW (virtual CSW) 54-56
  VDEVEXTN (virtual spool extension) 54-56
  VDEVFLAG (device dependent information) 54-56
  VDEVFLG2 (Reserve/Release flags) 54-56
  VDEVIOB (active IOBLOK pointer) 54-56
  VDEVREAL (real device block address) 54-56
  VDEVRRB (address of VRRBLOK) 54-56
  VDEVSFLG (virtual spooling flags) 54-56
  VDEVSTAT (status) 54-56
verifying existence of saved systems 202
virtual
  block multiplexer channel option 102
  console functions, DIAGNOSE instruction 184.1
  operator's console 76
  processor 76
virtual=real option 77-78,92-93,95-97
virtual console, operator 76
virtual devices, I/O 76
virtual interval timer assist 101,175-176
virtual machine assist feature
  described 98-99
  described (5748-XE1) 98.2
  restrictions for use of 98-99
  usage 98-99
  used to reduce real supervisor state time 98-99
  used to reduce real supervisor-state time (5748-XE1) 98.2
Virtual Machine Communication Facility (VMCF) (see VMCF)
  introduction to 143
Virtual Machine Facility/370 (VM/370)
  CMS 231-233
  control program 73-230
  device types in 192-194
  DIAGNOSE instruction in 182
  directory 76
  load environment 124-125
  program states 83
  RSCS 327-342
virtual machine storage size
  maximum, replacing directory entry 206-210
  replacing directory entry 206-210
virtual machines
  abend dump 23
  abnormal termination 8-13,15-17,23
  creation 76
  described 76
  DIAGNOSE instruction usage 182
  directory 76
  disabled loop 8-13,15-17
    procedure 26-27
  disabled wait 8-13
    procedure 15-17,29-30
  dispatching scheme 84-85
  enabled loop 8-13,15-17
    procedure 26-27
  enabled wait 8-13
    procedure 15-17,30
    with real timer option 30
    without real timer option 30
  interrupt, handled by CP 76
  I/O management
    dedicated devices 79-80
    directory 79-80
    mass storage volumes 79-80
    shared devices 79-80
    spooled devices 79-80
  I/O operation 89-90
  operating system 76
  performance
    for time-shared multi-batch machines 126
    options 93
  PSW 83
  shared segment operation 142
  storage management
    directory 76-77
    virtual storage 76-77
  time management
    conversational user 76
    nonconversational user 76
    priority of execution 76
  timers 175-176
  unexpected results 8-13,15-17
    procedure 24-25
Virtual Reserve/Release support, virtual machine I/O management 79-80
virtual storage 76
  management
    CP 76-77
    RSCS 335
virtual storage preservation
  purpose of (5748-XE1) 78.1
  SAVESEQ priority value (5748-XX8) 78.2
  SAVESEQ priority value (5748-XE1) 78.2
  VMSAVE option (5748-XX8) 78.1
  VMSAVE option (5748-XE1) 78.1
virtual storage preservation (5748-XX8) 78.1
VM Monitor 110-125
  collection mechanism 110-125
  collection mechanism (5748-XX8) 110.1
  collection mechanism (5748-XE1) 110.1
  considerations 122-123
  data records 357-365
  data volume and overhead 123
  header record 357-365
  monitor classes 110-125
  monitor classes (5748-XX8) 110.1
  monitor classes (5748-XE1) 110.1
  responses to unusual tape conditions 121-122
  tape format and content 357-365

VMBLOK 19,29,50-53
VCUSTRT (address of VCUBLOK table) 53-54
VMCHSTRT (address of VCHBLOK table) 53-54
VMCOMND (last command) 50-53
VMDSTAT (dispatching status) 50-53
VMDVSTRT (address of VDEVBLOK table) 54-56
VMEXTINT (external interrupts) 50-53
VMIOACTV (active channel mask) 50-53
VMIOINT (I/O interrupts) 50-53
VMPEND (interrupts pending) 50-53
VMPSW (virtual PSW) 50-53
VMRSTAT (running status) 50-53
VMCF (Virtual Machine Communication Facility) 143
DIAGNOSE instruction 143,159,203-204
data transfer error codes 167-169
return codes 167-169
external interrupt 164-167
invoking subfunctions 159
protocol 148-149
IDENTIFY 152
SEND 148-149
SEND/RECV 150
SENDX 151
VMCF (virtual machine communication facility), special message facility 143
VMCF (Virtual Machine Communication Facility)
subfunctions 153-169
AUTHORIZE 153
CANCEL 154
IDENTIFY 155
PRIORITY option 153,156-157,157
QUIESCE 154
RECEIVE 158
REJECT 155
REPLY 158
RESUME 155
SEND 156-157
SEND/RECV 156-157
SENDX 157
VMCF (virtual machine communication facility), subfunctions, special message facility 153
VMCF (Virtual Machine Communication Facility)
subfunctions
SPECIFIC option 153
UNAUTHORIZE 154
table of subfunctions 144
user doubleword 164-167
using 145
applications 145
general considerations 148
performance considerations 147
security 146-147
VMCPARM parameter list 159-162
VMFDUMP, command, usage 45-46
VMSAVE areas (5748-XX8) 78.2
VMSAVE areas (5748-XE1) 78.2
VMSAVE option (5748-XX8) 78.1
VMSAVE option (5748-XE1) 78.1
VM/370 (see Virtual Machine Facility/370 (VM/370))
Volume Table of Contents (VTOC), support of 278-279
VSAM
CMS support for 310
data sets
compatibility considerations 312-313
compatibility considerations (5748-XX8) 310-312.1
compatibility considerations (5748-XE1) 310-312.1
device support under CMS 310
macros supported under CMS 311-312
storage requirements for use under CMS 312-313
support of 278-279

W
WAIT macro 273
wait state 27-29
CP
disabled wait 27-29
enabled wait 29
RSCS
virtual machine disabled wait 30-31
virtual machine enabled wait 31
virtual machine
disabled wait messages 29-30
enabled wait procedure 30
WRITE macro 278-279
WTO/WTOR macros 275

X
XCTL macro 274
XDAP macro 273

Z
ZAP, command 41

3
3203 Model 4 and 5 Printer, forms control and print buffer 223
3262
FCB (5748-XX8) 222.1
FCB (5748-XE1) 216.1,222-225
UCSB buffer images (5748-XX8) 216,221
UCSB buffer images (5748-XE1) 211-213,216.1
3270
virtual console interface 199-200
attribute bytes, how to supply (5748-XX8) 197
attribute bytes, how to supply (5748-XE1) 199-200.4
full screen interactions (5748-XX8) 199-200.4
full screen interactions (5748-XE1) 199-200.4
full screen mode (5748-XX8) 199-200.4

full screen mode (5748-XE1) 199-200.4
selector pen limitations (5748-XX8) 199
selector pen limitations (5748-XE1) 199-200.4
3289
font offset buffer
adding FOBs (5748-XX8) 222
adding FOBs (5748-XE1) 222-225
names of supplied FOBs (5748-XX8) 222
names of supplied FOBs (5748-XE1) 222-225
purpose of (5748-XX8) 222
purpose of (5748-XE1) 222-225
font offset buffer (5748-XX8)
3704/3705 control program, saving 199
3800 printer 226-228
as a dedicated device 226
as a real spooling device 227
CHANGE command 227
creating and modifying control tables 227-228
loading control tables 228
SPOOL command 227
START command 227
storing control tables 228
features
character arrangement tables 226-228
character modification 226-228
copy modification 226-228
FCB 226-228
forms overlay (flashing) 226-228
multiple copy printing 226-228

Technical Newsletter

This Newsletter No. GN25-0829
Date April 1, 1981

Base Publication No. GC20-1807-7
File No. S370-36 (VM/370 Release 6 PLC 17)
Prerequisite Newsletters/ Supplements GN25-0492

REC'D JUL 16 1981

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

© Copyright IBM Corp. 1972, 1973, 1974, 1975, 1976, 1977, 1979, 1981

This Technical Newsletter contains replacement pages for VM/370 System Programmer's Guide to support Release 6 PLC 17 of IBM Virtual Machine Facility/370.

Before inserting any of the attached pages into the VM/370 System Programmer's Guide, read carefully the instructions on this cover. They indicate when and how you should insert pages.

Pages to be Removed	Attached Pages to be Inserted*
Title, Edition Notice	Title, Edition Notice
Preface iii-vi	Preface iii-vi
Contents vii-xii	Contents vii-xiv
Summary of Amendments xiii-xvi	Summary of Amendments xv-xviii
5-6	5-6
15-16	15-16
19-20	19-20
41-42	41-42
45-46	45-46
59-64	59-64.2
71-72	71-72
79-80	79-80
91-92	91-92
99-102	99-102.2
107-108	107-108
111-112	111-112
115-116	115-116
127-128	127-128
133-136	133-136
139-148	139-148
153-154	153-154
157-166	157-166.2
171-172	171-172.2
181-190	181-190
193-194	193-194
197-198	197-198.2
201-212	201-212
219-222	219-222
225-228	225-228
245-252	245-252
255-256	255-256
261-262	261-262
273-274	273-274
277-280	277-280.2
287-294	287-294
297-298	297-298
311-314	311-314
321-322	321-322
361-362	361-362
Index 367-384	Index 367-384

IBM Corporation, Programming Publications, Department G60,
PO Box 6, Endicott, New York 13760

© Copyright IBM Corp. 1981

Printed in U.S.A.

Type of Event	Module	Identification Code (hexadecimal) (See Note 1)	Format of Trace Table Entry
External interrupt	DMKPSA	01	X'01' (0) · X'0000000000' (1) · Interrupt Code (6) · External Old PSW (8–15)
SVC interrupt	DMKSVC	02	X'02' (0) · GR14 or GR15 (See Note 2) (1) · Instruction Length Code (4) · Interrupt Code (6) · SVC Old PSW (8–15)
Program interrupt	DMKPRG	03	X'03' (0) · First 3 bytes of VMPSW (1) · Instruction Length Code (4) · Interrupt Code (6) · Program Old PSW (8–15)
Machine Check Interrupt	DMKMCH	04	X'04' (0) · Address of VMBLOK (1) · First 4 bytes of 8-byte Interrupt Code (4) · Machine Check Old PSW (8–15)
I/O interrupt	DMKIOS	05	X'05' (0) · X'00' (1) · Device Address (2) · I/O Old PSW + 4 (4) · CSW (8–15)
Free Storage (FREE)	DMKFRE	06	X'06' (0) · Address of VMBLOK (1) · GR 0 at entry (4) · GR 1 at exit (8) · GR 14 (12–15)
Return storage (FRET)	DMKFRE	07	X'07' (0) · Address of VMBLOK (1) · GR 0 at entry (4) · GR 1 at entry (8) · GR 14 (12–15)
Enter Scheduler	DMKSCH	08	X'08' (0) · Address of VMBLOK (1) · Value of VMRSTAT, VMDSTAT, VMOSTAT, and VMQSTAT (4) · VMQLEVEL VMTLEVEL (8) · VMIOINT (10) · VMPEND (12) · GR 14 (13–15)
Queue drop	DMKSCH	09	X'09' (0) · Address of VMBLOK (1) · X'0000' (4) · New Priority (6) · Number of Resident Pages (8) · Projected Working Set (10) · Current page load (PSA) (12) · Current Page rate (14–15)
Run user	DMKDSP	0A	X'0A' (0) · X'000000' (1) · RUNUSER value from PSA (4) · RUNPSW value from PSA (8–15)
StartI/O	DMKCNS DMKIOS DMKCPI	0B	X'0B' (0) · Condition Code (1) · Device Address (2) · Address of IOBLOK (4) · CAW (8) · For CC = 1, CSW + 4 otherwise this field is not used (12–15)
Unstack I/O interrupt	DMKDSP	0C	X'0C' (0) · X'00' (1) · Virtual Device Address (2) · Address of VMBLOK (4) · Virtual CSW (8–15)
Virtual CSW store	DMKVSI	0D	X'0D' (0) · Instruction Operation Code (1) · Virtual Device Address (2) · Address of VMBLOK (4) · Virtual CSW (8–15)
TestI/O	DMKCPI DMKIOS	0E	X'0E' (0) · Condition Code (1) · Device Address (2) · Address of IOBLOK (4) · CAW (8) · For CC = 1, CSW + 4 otherwise this field is not used (12–15)
Halt Device	DMKCNS DMKIOS DMKVSI DMKCPI	0F	X'0F' (0) · Condition Code (1) · Device Address (2) · Address of IOBLOK (4) · CAW (8) · For CC = 1, CSW + 4 otherwise this field is not used (12–15)
Unstack IOBLOK or TROBLOK	DMKDSP	10	X'10' (0) · Address of VMBLOK (1) · Value of VMRSTAT, VMDSTAT, VMOSTAT, and VMQSTAT (4) · Address of IOBLOK or TROBLOK (8) · Interrupt Return Address (12–15)
NCP BTU (See Note 3)	DMKRNH	11	X'11' (0) · X'00' (1) · CONSRID (2) · CONDEST (4) · CONRTAG (6) · CONSYSR CONEXTR (8) · CONTCMD (10) · CONFUNC CONDFLG (12) · CONDCNT (14–15)
Spinning on lock	DMKLOK	12	X'12' (0) · @ of VMBLOK (1) · Lockword Address (4) · Return Address (8) · Lockword Contents (12–15)
SIGP issued	DMKEXT	13	X'13' (0) · Return Address (1) · 00 (4) · Condition Code (5) · Real Processor Address (6) · Function Code (8) · Order Code (10) · Status of Condition Code = 1 (12–15)
Clear Channel issued	DMKVSI	14	X'14' (0) · Condition Code (1) · Device Address (2) · Address of VMBLOK (4) · Virtual CSW (8–15)

Notes:

1. If the installation is running in attached processor mode, the identification code will be OR'd with an X'40' if the activity occurred on the attached processor. If the installation is running ECPS, the identification code is OR'ed with an X'80' if the activity occurred in microcode.
2. If the interrupt code (bytes 6 and 7) is 0C, the contents of GR14 are displayed. For all other interrupt codes, the contents of GR15 are displayed.
3. Bytes 2 through 15 of a code 11 trace record represent a Basic Transmission Unit, sent or received by a **3704/3705**. If **CONSYSR/CONEXTR** are zero, the **BTU** was transmitted to the **3704/3705**. If they are non-zero, the **BTU** was received. If **CONTCMD** equals **X'7700'**, this is an unsolicited **BTU** response.

Figure 8. CP Trace Table Entries

Abend Dumps

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

*If you are inserting pages from different Newsletters/Supplements and identical page numbers are involved, always use the pages with the latest date (shown in the slug at the top of the page). The page with the latest date contains the most complete information.

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

Summary of Amendments

This Technical Newsletter incorporates changes reflecting minor technical and editorial changes.

Note: Please file this cover letter at the back of the publication to provide a record of changes.

Technical Newsletter

This Newsletter No.	GN25-0492
Date	August 1, 1979
Supplement No.	
Base Publication No.	GC20-1807-7
File No.	S370-36 (VM/370 Release 6 PLC 4)
Prerequisite Newsletters/ Supplements	None

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

© Copyright IBM Corp. 1972, 1973, 1974, 1975, 1976, 1977, 1979

This Technical Newsletter contains replacement pages for VM/370 System Programmer's Guide to support Release 6 PLC 4 of IBM Virtual Machine Facility/370.

Before inserting any of the attached pages into the VM/370 System Programmer's Guide, read carefully the instructions on this cover. They indicate when and how you should insert pages.

Pages to be Removed	Attached Pages to be Inserted*
Title, Edition Notice	Title, Edition Notice
Contents vii-xii	Contents vii-xii
Summary of Amendments xiii-xvi	Summary of Amendments xiii-xvi
23-24	23-24
31-32	31-32.2
37-40	37-40.2
63-64	63-64.2
99-102	99-102
195-196	195-196
Index 367-384	Index 367-384

*If you are inserting pages from different Newsletters/Supplements and identical page numbers are involved, always use the pages with the latest date (shown in the slug at the top of the page). The page with the latest date contains the most complete information.

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

Summary of Amendments

This Technical Newsletter incorporates the following changes reflecting:

- CP dump services for virtual machines
- 3031AP Extended Control-Program Support

Note: Please file this cover letter at the back of the base publication to provide a record of changes.

IBM Corporation, Publications Development, Department D58, Building 706-2, PO Box 390, Poughkeepsie, New York 12602

Printed in U.S.A.

READER'S COMMENT FORM

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

Order No. GC20-1807-7

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

Reader's Comment Form

Cut or Fold Along Line

Fold and tape | Please Do Not Staple | Fold and tape

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE:

International Business Machines Corporation
Department D58, Building 706-2
PO Box 390
Poughkeepsie, New York 12602

Attn: VM/370 Publications

Fold and tape | Please Do Not Staple | Fold and tape

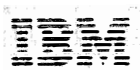
IBM VM/370 System Programmer's Guide Printed in U.S.A. GC20-1807-7

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, N.Y. 10604

IBM World Trade Americas/Far East Corporation
Town of Mount Pleasant, Route 9, North Tarrytown, N.Y., U.S.A. 10591

IBM World Trade Europe/Middle East/Africa Corporation
360 Hamilton Avenue, White Plains, N.Y., U.S.A. 10601

IBM VM/370: System Programmer's Guide
Printed in U.S.A.
GC20-1807-7



International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, N.Y. 10604

IBM World Trade Americas/Far East Corporation
Town of Mount Pleasant, Route 9, North Tarrytown, N.Y., U.S.A. 10591

IBM World Trade Europe/Middle East/Africa Corporation
360 Hamilton Avenue, White Plains, N.Y., U.S.A. 10601