SY28-1133-2
File No. S370-37

Program Product

MVS Diagnostic Techniques

MVS/System Product
JES3 5740-XYN
MVS/System Product
JES2 5740-XYS

IBM

Third Edition (July, 1985)

This is a major revision of, and obsoletes, SY28-1133-1 and Technical Newsletter SN28-0875. See the Summary of Amendments following the Contents for a summary of the changes made to this manual. Technical changes or additions to the text and illustrations are indicated by a vertical line to the left of the change.

This edition with Technical Newsletter SN28-5095 applies to Version 1 Release 3.6 of MVS/System Product 5740-XYN or 5740-XYS and to all subsequent releases until otherwise indicated in new editions or Technical Newsletters. Changes are made periodically to the information herein; before using this publication in connection with the operation of IBM systems, consult the latest *IBM System/370 Bibliography*, GC20-0001, for the editions that are applicable and current.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM program product in this publication is not intended to state or imply that only IBM's program product may be used. Any functionally equivalent program may be used instead.

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

A form for reader's comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, Information Development, Department D58, Building 921-2, PO Box 390, Poughkeepsie, NY 12602. IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Guide for Using This Publication

The following is a list of guidelines for using this publication.

- This publication contains information from *OS/VS2 System Programming Library: MVS Diagnostic Techniques*, GC28-0725-2. It also includes all information from the Newsletters and Supplements issued for GC28-0725-2 through September 15, 1981.

- This publication contains information for OS/VS2 MVS Release 3.8 plus the following:

  - OS/VS2 MVS Processor Support 2
  - OS/VS2 MVS/System Product Release 1
  - OS/VS2 MVS/System Product Release 1 Enhancements
  - OS/VS2 MVS/System Product Release 2
  - OS/VS2 MVS/System Product Release 3
  - OS/VS2 MVS/System Product Release 3.2 (TNL, SN28-5014)
  - OS/VS2 MVS/System Product Release 3.3
  - OS/VS2 MVS/System Product Release 3.4 (TNL, SN28-0875)
  - OS/VS2 MVS/System Product Release 3.5

- Do not use this publication unless you have installed MVS/System Product Version 1 Release 3.5.

- The implied date of this publication, for the purpose of inserting new Newsletters and Supplements, is July 1, 1985. When you are adding pages from different Newsletters and Supplements, always use the page with the latest date (shown at the top of the page).

Preface

This publication describes diagnostic techniques and guidelines for isolating problems on MVS systems. It is intended for the use of system programmers and analysts who understand MVS internal logic and who are involved in resolving MVS system problems.

This publication is intended for use only in debugging. None of the information contained herein should be construed as defining a programming interface.

Note: For JES3 diagnostic information, refer to *JES3 SPL: Diagnosis.*

Organization and Contents

This publication stresses a three-step debugging approach:

1. Identifying the external symptom of the problem.

2. Gathering relevant data from system data areas in order to isolate the problem to the component level.

3. Analyzing the component to determine the cause of the problem.

In support of this approach, the publication has been reorganized into three basic parts consisting of five sections and three appendixes as follows:

Part 1

"Section 1. General Introduction" describes the debugging approach that is used and defines the external symptoms that are used to identify a system problem.

"Section 2. Important Considerations Unique to MVS" describes concepts and functions that should be understood prior to undertaking system diagnosis. Included are: global system analysis, system execution modes and status saving, locking, use of recovery work areas, effects of MP, trace analysis, debugging hints, and general data gathering techniques.

"Section 3. Diagnostic Materials Approach" provides guidelines for obtaining and analyzing storage dumps of data areas affected by the problem.

Part 2

"Section 4. Symptom Analysis Approach" describes how to identify an external symptom (loop, wait state, TP problem, performance degradation, or incorrect output), and provides an analysis procedure for what kind of problem is causing the symptom.

"Section 5. Component Analysis" describes the operating characteristics and recovery procedures of selected system components and provides debugging techniques for determining the cause of a problem that has been isolated to a particular component.

Part 3

Appendixes

A - describes the flow of various MVS processes.

B - provides a step-by-step approach to analyzing a stand-alone dump.

C - provides diagnostic information for SVC dump titles.

D - contains definitions of abbreviations used throughout the publication.

Referenced Publications

The following publications either are referenced in this publication or provide related reading:

System/370 Principles of Operation		GA22-7000
Synchronous Data Link Control General Information		GA27-3093
MVS Interactive Problem Control System (IPCS)		
User's Guide and Reference		GC28-1183
Environmental Record Editing and Printing (EREP) Program		
User's Guide and Reference		GC28-1378
OS/VS2 System Programming Library:		
Initialization and Tuning Guide		GC28-1029
Supervisor		GC28-1046
Job Management		GC28-1303
Service Aids		GC28-0674
SYS1.LOGREC Error Recording		GC28-0677
Debugging Handbook (5 volumes)	LC28-1385 through	LC28-1389
JES3 System Programming Library: Diagnosis		SC23-0043
OS/VS2 TCAM System Programmer's Guide, TCAM Level 10		GC30-2051
OS/VS2 TCAM Debugging Guide, TCAM Level 10		GC30-3040
OS/VS2 MVS VTAM Debugging Guide		GC27-0023
ACF/VTAM Diagnosis Guide		SC27-0615
ACF/VTAM Diagnosis Reference		SC27-0621
Operator's Library:		
OS/VS2 MVS System Commands		GC28-1031
OS/VS2 MVS JES2 Commands		SC23-0048
VTAM Network Operating Procedures		GC27-6997
ACF/VTAM Operation		SC27-0612
OS/VS TCAM Level 10		GC30-3037
JES/3 Operator's Library		SC23-0045

MVS/370 Message Library:	
JES2 Messages	GC28-1354
System Messages (2 volumes)	GC38-1374, GC28-1375
System Codes	GC38-1008
OS/VS2 System Logic Library (11 volumes) - Volume 1	SY28-0713
OS/VS2 I/O Supervisor Logic	SY26-3823
OS/VS2 System Initialization Logic	LY28-1050
OS/VS2 MVS Service Aids Logic	SY28-0643
OS/VS2 MVS Global Resource Serialization Logic	LY28-1059
JES2 Logic	LY24-6006
OS/VS2 MVS JES3 Logic	LY24-6005
Resource Access Control Facility (RACF): Program Logic Manual	LY28-0730
OS/VS2 MVS Programmed Cryptographic Facility: Program Logic Manual	LY28-0958
OS/VS2 MVS Resource Measurement Facility (RMF) Version 2 Program Logic Manual	LY28-0923
MVS Input/Output Configuration Program Logic	LY28-1033
OS/VS2 VTAM Data Areas	SY27-7267
ACF/VTAM Data Areas	LY38-3054
OS/VS2 VTAM Logic	SY28-0621
ACF/VTAM Logic	LY27-8034
OS/VS2 VSAM Logic	SY26-3825
OS/VS2 Catalog Management Logic	SY26-3826
OS/VS2 Access Method Services Logic	SY35-0010
OS/VS2 MVS Mass Storage System Communication (MSSC) Logic	SY35-0013
OS/VS2 SAM Logic	SY26-3832
OS/VS2 BDAM Logic	SY26-3831
OS/VS2 MVS VTIOC and TCAS Logic	SY27-7269
OS/VS2 TSO Command Processor Logic, Volume IV	SY28-0652
OS/VS2 Open/Close/EOV Logic	SY26-3827
OS/VS2 CVOL Processor Logic	SY35-0011
OS/VS2 VIO Logic	SY26-3834
OS/VS2 TCAM Level 10 Logic	SY30-3032
IBM 3704 and 3705 Communications Controllers NCP/VS Logic	SY30-3013
3704/3705 Communications Controllers Principles of Operation	GC30-3004
IBM 3704/3705 Communications Controllers Emulation Program Generation and Utilities Guide and Reference Manual	GC30-3008
IBM 3704/3705 Communications Controller NCP/VS Generation and Utilities Guide and Reference Manual	GC30-3007

Contents

Section 1. General Introduction 1-1
Basic MVS Problem Analysis Techniques 1-1
IPCS - Interactive Problem Control System 1-4

Section 2. Important Considerations Unique to MVS 2-1
Global System Analysis 2-2
  Global Indicators that Determine the Current System State 2-2
  Work Queues, TCBs and Address Space Analysis 2-5
    TCB Summary 2-5
    SRB Dispatching Queues 2-5
    Address Space Analysis 2-6
    Task Analysis 2-7
    Summary 2-9
System Execution Modes and Status Saving 2-10
  System Execution Modes 2-10
    Task Mode 2-10
    SRB Mode 2-11
    Physically Disabled Mode 2-12
    Locked Mode 2-13
  Determining Execution Mode From a Stand-alone Dump 2-13
    LCCA Indicators 2-13
    PSA Indicators 2-13
    ASCB Indicators 2-14
  Locating Status Information in a Storage Dump 2-15
    Task and SRB Mode Interruptions 2-15
    Locally Locked Task Suspension 2-17
    SRB Suspension 2-17
    SMF Suspension 2-19
Locking 2-21
  Categories of Locks 2-21
  Types of Locks 2-22
  Locking Hierarchy 2-23
  Determining Which Locks Are Held On a Processor 2-24
  Content of Lockwords 2-25
    Global Spin Lockword 2-25
    Global Suspend Lockword (Cross Memory Services Locks) 2-25
    Local Suspend Lockword (Local Lock) 2-25
  How To Find Lockwords 2-26
  Results of Requests For Unavailable Locks 2-26
    Global Spin Locks 2-26
    Local Locks 2-27
    Cross Memory Services Locks 2-28
  Intersect 2-31

Determining if Intersects are Held on a Processor 2-31
Requesting the Intersect 2-32
Use of Recovery Work Areas For Problem Analysis 2-33
SYS1.LOGREC Analysis 2-34
Listing the SYS1.LOGREC Data Set 2-34
SDWAVRA Key-Length-Data Format 2-35
Important Considerations About SYS1.LOGREC Records 2-36
SYS1.LOGREC Recording Control Buffer 2-38
Formatting the LOGREC Buffer 2-38
Finding the LOGREC Recording Control Buffer 2-38
Format of the LOGREC Recording Control Buffer 2-38
FRR Stacks 2-40
Extended Error Descriptor (EED) 2-41
RTM2 Work Area (RTM2WA) 2-41
Formatted RTM Control Blocks 2-42
System Diagnostic Work Area (SDWA) Use in RTM2 2-42
Effects of Multiprocessing On Problem Analysis 2-43
Features of an MP Environment 2-43
MP Dump Analysis 2-45
Data Areas Associated With the MP Environment 2-45
Parallelism 2-46
General Hints For MP Dump Analysis 2-48
Inter-Processor Communication 2-49
Direct Services 2-50
Remote Pendable Services 2-51
Remote Immediate Services 2-52
MP Debugging Hints 2-58
MVS Trace Analysis 2-61
Trace Entries 2-61
Trace Entry for Service Processor Call SVC 2-64
Trace Examples 2-64
Notes For Traces 2-66
Tracing Procedure 2-66
Bypassing GTF Lost Events 2-68
Cautionary Notes 2-69
Master Trace 2-71
Master Trace Table 2-72
The Message Processing Facility Table (MPFT) 2-74
Miscellaneous Debugging Hints 2-75
Alternate CPU Recovery (ACR) Problem Analysis 2-75
Pattern Recognition 2-77
Low Storage Overlays 2-78
Common Bad Addresses 2-80
OPEN/CLOSE/EOV ABENDs 2-80
Debugging Machine Checks 2-81
Debugging Problem Program Abend Dumps 2-86
Debugging From Summary SVC Dumps 2-89
SUMDUMP Output For SVC-Entry SDUMP 2-90
SUMDUMP Output For Branch-Entry SDUMP 2-91
Started Task Control ABEND and Reason Codes 2-93
SWA Manager Reason Codes 2-94
Additional Data Gathering Techniques 2-95
Using the CHNGDUMP, DISPLAY DUMP and DUMP Operator Commands 2-95

How to Print Dumps 2-97
How to Automatically Establish System Options For SVC Dump 2-99
How to Copy PRDMP Tapes 2-99
How to Rebuild SYS1.UADS 2-100
How to Print SYS1.DUMPxx 2-101
How to Clear SYS1.DUMPxx Without Printing 2-102
How to Print the SYS1.COMWRITE Data Set 2-103
How to Print an LMOD Map of a Module 2-103
How to Re-Create SYS1.STGINDEX 2-103
Software LOGREC Recording 2-104
Using the PSA as a Patch Area 2-104
Using the SLIP Command 2-105
SLIP Event Qualifier Keywords 2-105
Using the ACTION Keyword 2-112
Dump Tailoring 2-117
Examples of Using the SLIP Command 2-118
Example of SLIP Command From TSO Terminal 2-124
Designing an Effective SLIP Trap 2-125
Controlling SLIP Traps 2-125
Placement of PER Traps 2-127
SLIP Command Keyword Summary 2-129
System Stop Routine 2-133
How to Expand the Trace Table 2-133

Section 3. Diagnostic Materials Approach 3-1
Stand-alone Dumps 3-2
SVC Dumps 3-4
How to Change the Contents of an SVC Dump Issued by an Individual Recovery Routine 3-5
SDUMP Parameter List 3-5
SYSABENDs, SYSMDUMPs, and SYSUDUMPs 3-7
Software-Detected Errors 3-7
Hardware-Detected Errors 3-8

Section 4. Symptom Analysis Approach 4-1
Waits 4-2
Characteristics of Disabled Waits 4-2
Analysis Approach For Disabled Waits 4-3
Characteristics of Enabled Waits 4-4
Analysis Approach For Enabled Waits 4-5
Stage 1: Preliminary Global System Analysis 4-6
Stage 2: Key Subsystem Analysis 4-8
Stage 3: System Analysis 4-14
Loops 4-15
Common Loop Situations 4-15
Analysis Procedure 4-16
TP Problems 4-20
Message Flow Through the System 4-20
Types of Traces 4-21
GTF Traces 4-21
ACF/VTAM Traces 4-22
ACF/TCAM Traces 4-22
NCP and EP Traces 4-22
Performance Degradation 4-23

Operator Commands 4-23
Dump Analysis Areas 4-24
Incorrect Output 4-29
Initial Analysis Steps 4-29
Isolating the Component 4-29
Analyzing System Functions 4-30
Summary 4-31

Section 5. Component Analysis 5-1
Supervisor Control 5-2
Dispatcher 5-2
Important Dispatcher Entry Points 5-2
Dispatchable Units and Sequence of Dispatching 5-4
Dispatchability Tests 5-13
Miscellaneous Notes About the Dispatcher 5-14
Dispatcher Recovery Considerations 5-15
Dispatcher Error Conditions 5-16
SRB/SSRB Pool Manager 5-17
SRB/SSRB Pool Manager Entry Points 5-17
SRB/SSRB Pool Manager Recovery Considerations 5-18
SRB/SSRB Pool Manager Error Conditions 5-19
Stop/Reset Services 5-19
Stop/Reset Entry Points 5-19
Stop/Reset Recovery Considerations 5-20
Stop/Reset Error Conditions 5-21
SUSPEND/RESUME/TCTL Services 5-22
SUSPEND/RESUME/TCTL Entry Points 5-22
RESUME/TCTL Recovery Considerations 5-23
SUSPEND/RESUME/TCTL Error Conditions 5-24
IOS 5-25
Front-End Processing 5-25
Back-End Processing 5-25
IOS Problem Analysis 5-25
IOS ABEND Codes 5-28
Loops 5-28
IOS WAIT States 5-29
General Hints For IOS Problem Analysis 5-30
IOS Diagnostic Aids 5-32
Table of EXCP Abend Codes 5-32
EXCP Debugging Area (XDBA) 5-33
SDWA Variable Recording Area 5-34
Output of IOS Recovery Procedures 5-34
Informative IOSB Fields 5-43
Table of IOS Messages 5-46
IOS Wait State Codes 5-47
Table of IOS Return Codes 5-47
Error Recovery Procedures (ERPs) 5-48
IOS and ERP Processing 5-48
Identifying ERP Module Names 5-49
How ERP Transfers Control 5-49
Abnormal End Appendages 5-49
Retry/Restart the Channel Program 5-50
Error Interpreter 5-50
ERP Messages and Logging 5-51

Intercept Conditions 5-51
Unit Check on Sense Command 5-52
Compound Errors 5-52
Diagnostic Approach 5-52
Program Manager 5-55
Functional Description 5-55
Program Manager Organization 5-55
Program Manager Control Blocks 5-55
Program Manager Queues 5-55
Queue Validation 5-58
System Initialization 5-58
Basic Functional Flow 5-60
LINK 5-60
ATTACH 5-61
XCTL 5-61
LOAD 5-64
DELETE 5-64
Exit Resource Manager 5-64
SYNCH 5-65
IDENTIFY 5-65
ABEND Resource Manager 5-66
806 Abend 5-66
APF Authorization 5-70
Module Subpools 5-71
FETCH/Program Manager Work Area (FETWK) 5-72
RB Extended Save Area (RBEXSAVE) 5-72
CDE Pool Control 5-72
Virtual Fetch 5-74
Functional Description 5-74
Module Organization 5-74
Functional Flow 5-75
Control Blocks 5-78
Recovery Processing 5-79
Error During Initialization Processing 5-79
Errors During Build, Find, and Get Processing 5-79
Debugging Hints 5-80
VSM 5-82
Address Space Initialization 5-84
Step Initialization/Termination (IEAVPRT0 - GETPART/FREEPART) 5-86
Virtual Storage Allocation (GETMAIN/FREEMAIN) 5-87
GETMAIN's Functional Recovery Routine - IEAVGFRR 5-90
VSM Cell Pool Management 5-92
Miscellaneous Debugging Hints 5-92
Real Storage Manager (RSM) 5-97
Major RSM Control Blocks 5-97
PCB (Page Control Block) 5-99
SPCT (Swap Control Table) 5-100
PFTE (Page Frame Table Entry) 5-101
Page Stealing 5-101
Reclaim 5-102
Relate 5-103
RSM Recovery 5-104
Real Storage Management ABEND Reason Codes 5-105

RSM Debugging Tips 5-107
Converting a Virtual Address to a Real Address 5-108
Example: Converting a Virtual Address to a Real Address 5-110
PCB Trace Facility 5-111
Auxiliary Storage Manager (ASM) 5-112
Component Functional Flow 5-113
Saving an LG 5-113
Requesting I/O 5-114
Requesting Swap I/O 5-115
Component Operating Characteristics 5-117
System Mode 5-117
Address Space, Task, and SRB Structure 5-117
Storage Considerations 5-117
Interfaces With Other Components 5-118
Register Conventions 5-118
Footprints and Traces 5-118
General Debugging Approach 5-119
Paging Interlocks 5-119
Incorrect Pages 5-120
Unusable Paging Data Sets 5-125
Page/Swap Data Set Errors 5-127
Error Analysis Suggestions 5-127
Validity Checking 5-128
ASM Serialization 5-129
Recovery Considerations 5-131
Recovery Traces 5-132
Recovery Structure 5-132
Recovery As a Debugging Tool 5-133
Recovery Footprints 5-133
ASM Diagnostic Aids 5-134
C0D ABEND Meanings for ASM 5-134
ASM Recovery Control Blocks 5-135
Additional ASM Data Areas 5-139
System Resources Manager (SRM) 5-141
SRM Objectives 5-141
Address Space States 5-142
SRM Indicators 5-143
System Indicators 5-143
Individual User Indicators 5-147
Other Indicators 5-148
SRM Error Recovery 5-148
SRM SDWA Data 5-149
Module Entry Point Summaries 5-149
VTAM 5-150
Note to Readers 5-150
VSAM 5-151
Record Management 5-151
RPL 5-151
PLH 5-152
BUFC 5-152
Record Management Debugging Aids 5-153
Open/Close/End-Of-Volume 5-155
O/C/EOV Debugging Aids 5-155
I/O Manager 5-157

I/O Manager Debugging 5-157
Catalog Management 5-158
Major Registers and Control Blocks 5-158
How to Find Registers 5-158
Major Registers 5-159
Major Control Blocks 5-159
Module Structure 5-164
VSAM Catalog Recovery Logic 5-165
Establishing/Releasing a Recovery Environment 5-165
Maintaining a Pushdown List End Mark 5-166
Tracking GETMAIN/FREEMAIN Activity 5-166
CMS Function Gate 5-167
Recovery Routine Functions 5-167
Diagnostic Output (Function 4) 5-167
Backout (Function 7) 5-168
Drop Catalog Orientation (Function 9) 5-168
Storage Freeup (Function 10) 5-168
DEFINE/DELETE Backout (Function 12) 5-169
Debugging Aids 5-170
Allocation/Unallocation 5-172
Functional Description 5-172
Allocation 5-172
Unallocation 5-173
Batch Initialization and Control 5-173
Dynamic Initialization and Control 5-173
JFCB Housekeeping 5-174
Common Allocation 5-174
Common Unallocation 5-175
Volume Mount and Verify 5-176
General Debugging Aids 5-176
Allocation Module Naming Conventions 5-176
Registers and Save Areas 5-177
Common Allocation Control Block Processing 5-177
ESTAE Processing 5-180
Unit Allocation Status Recording 5-180
ALLOCAS Recovery Considerations 5-183
ALLOCAS Debugging Hints 5-183
Debugging Hints 5-192
Allocation Serialization 5-192
Subsystem Allocation Serialization 5-193
Device Selection Problems (Non-Abend) 5-193
Address Space Termination 5-194
0B0 Abend 5-194
0C4 Abend in IEFAB4FC 5-195
Volume Mount and Verify (VM&V) Waiting Mechanism 5-195
Allocation/Unallocation Reason Codes 5-197
Common and Batch Allocation and JFCB Housekeeping Reason Codes 5-197
Common and Batch Unallocation Reason Codes 5-200
Dynamic Allocation Reason Codes 5-200
JES2 5-201
Note to Readers 5-201
Subsystem Interface (SSI) 5-202
System Initialization Processing 5-202

Subsystem Interface Major Control Blocks 5-203
Requesting Subsystem Services 5-206
Invoking the Subsystem Interface 5-206
Logic Flow Examples 5-208
Notifying a Single Subsystem 5-208
Notifying All Active Subsystems 5-210
Debugging Hints 5-211
Event Notification Facility (ENF) 5-212
Requests for ENF Services 5-212
Listen and Signal Exit Routines 5-214
ENF Control Blocks 5-215
ENF Initialization 5-216
ENF Processing 5-217
ENF Return Codes 5-217
ENF Logic Flow Examples 5-217
ENF Recovery Routines 5-220
Recovery Termination Manager (RTM) 5-221
Functional Description 5-221
Work Areas 5-221
Major RTM Modules 5-221
Process Flow 5-222
Hardware Error Processing 5-222
Normal Task Termination 5-224
Abnormal Task Termination 5-225
Retry 5-226
Cancel 5-227
Address-Space Termination 5-229
PER Activation/Deactivation 5-230
Error ID 5-232
SVC Dump Debugging Aids 5-233
Important SVC Dump Entry Points 5-233
SVC Dump Error Conditions 5-234
SYS1.LOGREC Entries Produced for SVC Dump Errors 5-234
Control Blocks Used to Debug SVC Dump Errors 5-237
Resource Cleanup for SVC Dump 5-238
SLIP Processor Debugging Aids 5-238
SLIP Command Processor Recovery 5-239
SLIP Processor Recovery 5-239
PER Activation/Deactivation Recovery 5-240
Control Blocks Used by SLIP 5-242
Communications Task 5-244
Functional Description 5-244
Communications Task Control Blocks 5-246
Debugging Hints 5-248
Console Not Responding to Attention 5-248
Enabled Wait State 5-248
Disabled Wait State 5-249
Messages or Replies Lost 5-249
No Messages on One Console 5-250
Messages Routed to Wrong Console 5-250
Truncated Messages 5-250
Console Switching 5-251
Action Message Retention Facility Debugging Aids 5-251
DIDOCS Trace Table 5-252

DIDOCS-In-Operation Indicator 5-252
DIDOCS Locking 5-252
K Q Command Debugging Aids 5-253
Master Trace Debugging Aids 5-254
Recovery Management Support (RMS) 5-256
MCH Diagnostic Aids 5-256
MCH Return Codes 5-256
Processor Work Area (PWA) 5-257
PWF Diagnostic Aids 5-257
PWF Return Codes 5-257
PWF Data Areas 5-257
Dump Footprint Table 5-265
Appendage (ICFBDF00) Footprint Table 5-265
LOGREC Recording 5-266
CCH Diagnostic Aids 5-266
Message IGF002I 5-266
PCCA Fields Showing CCH Footprints 5-267
DDR Diagnostic Aids 5-268
DDR Tasks 5-268
DDR Communication Table (DDRCOM) 5-268
DDR Error Recovery Parameter List (DERPLIST) 5-269
DDR Return Codes 5-272
Software Recording 5-272
DDR Storage Dumps 5-272
MIH Diagnostic Aids 5-273
MIH Process 5-273
MIH Work Area 5-273
Software Recording 5-276
MIH Storage Dumps 5-276
Service Processor Call SVC and MSSFCALL DIAGNOSE Instruction 5-277
Service Processor Call SVC (SVC 122) Used With the MSSF 5-277
MSSFCALL Data Block 5-278
SVC Abend and Return Codes With the MSSF 5-279
SVC Processing and Control Blocks With the MSSF 5-279
SVC Control Blocks Used with the MSSF 5-280
MSSFCALL DIAGNOSE Instruction 5-281
MSSFCALL DIAGNOSE Instruction Condition Codes 5-282
Service Processor Call SVC and SERVICE CALL Instruction 5-283
Service Processor Call SVC (SVC 122) Used With the Service Processor Architecture 5-283
Service Call Control Block (SCCB) 5-284
SVC Abend and Return Codes With the Service Processor Architecture 5-284
SVC Processing With the Service Processor Architecture 5-285
SVC Control Blocks Used With the Service Processor Architecture 5-286
SERVICE CALL Instruction 5-287
SERVICE CALL Instruction Condition Codes 5-287
Cross Memory Services 5-288
PC/AUTH Services 5-288
Module Structure 5-289
Process Flow 5-290
Control Block Structure 5-291
Control Block Formats 5-293
Recovery Considerations 5-296

Debugging Hints 5-299
SLIP Traps 5-301
PCLINK Services 5-302
STKE Control Block 5-302
Module Structure 5-304
Debugging Hints 5-304
Global Resource Serialization 5-305
Functional Overview 5-306
Ring Processing 5-306
Command Processing 5-306
Dump Support 5-307
Resource Request Processing (Mainline and Fast Path) 5-307
CTC Processing 5-307
WTO/WTOR Message Processing 5-307
Initialization 5-307
Queue Scanning Services 5-308
Storage Management 5-308
Control Blocks 5-308
Control Block Overviews 5-310
Module Flow Diagrams 5-320
Diagnostic Aids 5-343
Check on Enabled Wait During IPL 5-343
System Indicators 5-343
Probe Points 5-344
CTC Processing Debugging Hints 5-344
Ring Processing Debugging Hints 5-345
ENQ/DEQ/RESERVE Processing Debugging Hints 5-345
Storage Management Debugging Hints 5-348
Serialization 5-350
Recovery Routines 5-351
SYS1.LOGREC Recording 5-351

Appendix A. Process Flows A-1
RSM Processing for Page Faults A-2
IEAVPIX Tests A-2
IEAVGFA Tests A-2
IEAVPIOP Tests A-3
IEAVIOCP Tests A-6
Swapping A-7
Swap-In Process A-7
Swap-Out Process A-9
EXCP/IOS A-12
GETMAIN/FREEMAIN A-15
GETMAIN Processing A-15
FREEMAIN Processing A-16
VTAM Process A-18
TSO A-21
Time Sharing Initialization A-21
LOGON Processing A-24
LOGON Scheduling Diagnostic Aids A-32
TSO Line Drop Processing A-34
TMP and Command Processor Interface A-37
TSO Command Processor Recovery A-41
TSO Terminal I/O Overview A-42

Terminal Output Flow A-43
Terminal Input Flow A-44
TSO/TIOC Terminal I/O Diagnostic Techniques A-45
TSO Attention Processing A-46

Appendix B. Stand-Alone Dump Analysis B-1
Overview B-1
Analysis Procedure B-6

Appendix C. SVC DUMP Title Directory C-1
System-Defined SVC Dump Titles C-2
Operator- and Caller-Defined SVC Dump Titles C-82
SVC Dumps Without Titles C-83
Module to SVC Dump Title Cross-Reference C-85

Appendix D. Abbreviations D-1

Index X-1

Figures

2-1. Definition and Hierarchy of MVS Locks 2-22
2-2. Bit Map to Show Locks Held on a Processor 2-24
2-3. Classification and Location of Locks 2-26
2-4. Cross Memory Services Lock Suspend Queues 2-30
2-5. Example of SDWAVRA in Key-Length-Data Format 2-35
2-6. Format of the LOGREC Recording Control Buffer 2-39
2-7. Format of Records Within the LOGREC Recording Control Buffer 2-39
2-8. SIGP Return Codes 2-50
2-9. External Call (XC) Process Flow 2-54
2-10. Emergency Signal (EMS) Process Flow 2-56
2-11. How to Locate the Trace Table 2-62
2-12. Types of Trace Entries 2-63
2-13. MVS Trace of a Page Fault Without I/O (Formatted by SNAP in SYSUDUMP/SYSABEND) 2-65
2-14. MVS Trace of a Page Fault With I/O (Unformatted) 2-65
2-15. GTF Trace of a Page Fault Without I/O 2-66
2-16. GTF Trace of a Page Fault With I/O 2-66
2-17. SLIP Command Summary 2-130
4-1. JES2 Commands for Status Information 4-24
4-2. System Use of Hardware Components 4-26
5-1. SRB Queue Structure and Control Block Relationships 5-6
5-2. Local SRB Queue Structure and Control Block Relationships 5-7
5-3. Dispatcher Processing Overview 5-10
5-4. I/O Processing Overview 5-26
5-5. Major IOS and EXCP Control Block Relationships 5-27
5-6. Program Manager Modules 5-56
5-7. Program Manager Control Blocks and Work Areas 5-57
5-8. Program Manager Queues 5-57
5-9. IEAVNP05 Initialization 5-59
5-10. New PRB Initialization - LINK 5-61
5-11. New RB Initialization - XCTL 5-62
5-12. XCTL RB Manipulation 5-63
5-13. CDE Initialization by IDENTIFY 5-66
5-14. Module Search Sequence for LINK, ATTACH, XCTL and LOAD 5-68
5-15. Module Search Sequence of Private Libraries 5-69
5-16. CDE Allocation 5-70
5-17. Virtual Fetch Modules 5-75
5-18. Virtual Fetch Control Blocks 5-78
5-19. Virtual Storage Management's View of MVS Storage 5-83
5-20. Virtual Storage Management's Control Block Usage 5-85
5-21. Virtual Storage Management's Global Data Area (GDA) 5-89
5-22. SDWAVRA Error Indicators 5-91

5-23. VSM Cell Pool Management 5-93
5-24. Major RSM Control Blocks and Their Functions 5-97
5-25. Relationship of Critical RSM Control Blocks 5-98
5-26. Page Stealing Process Flow 5-102
5-27. Converting Virtual Addresses to Real Addresses 5-109
5-28. Relationship of Important ASM Control Blocks 5-116
5-29. Locating An LSID From An LPID 5-122
5-30. Relating the Virtual Address to the PART and PAT 5-124
5-31. Page/Swap Data Set Error Action Matrix 5-127
5-32. SRM Control Block Overview 5-145
5-33. Relationship of the Six Major Functions of Allocation/Unallocation 5-172
5-34. Common Allocation Input 5-178
5-35. Common Allocation Control Blocks After Construction of Volunit Table and EDLs 5-179
5-36. ALLOCAS Control Block Structure 5-182
5-37. ALLOCAS Dump 5-187
5-38. VM&V Control Block Structure 5-196
5-39. Subsystem Interface Control Block Usage 5-205
5-40. Control Block Usage With Synonyms 5-206
5-41. Control Block Structure for Invoking Subsystem Interface 5-207
5-42. Finding the SSIB for a Job When SSOB Pointer is Zero 5-208
5-43. ENF Event Parameter List (ENFPM) 5-213
5-44. ENF Control Block Summary 5-215
5-45. ENF Control Block Structure 5-216
5-46. Sequence of Communications Task Processing 5-245
5-47. Communications Task Control Block Structure 5-247
5-48. Typical DDRCOM Chains 5-269
5-49. DDR Error Recovery Parameter List 5-270
5-50. MIH Work Area 5-275
5-51. Overview of SVC Processing With the MSSF 5-280
5-52. SVC Control Block Structure With the MSSF 5-281
5-53. Overview of SVC Processing With the Service Processor Architecture 5-286
5-54. SVC Control Block Structure With the Service Processor Architecture 5-287
5-55. PC/AUTH Control Block Structure 5-292
5-56. PC/AUTH Recovery Areas 5-297
5-57. PCLINK Control Block Structure 5-303
5-58. TCBs in the Global Resource Serialization Address Space 5-311
5-59. CTC Processing - Control Block Overview 5-312
5-60. Ring Processing - Control Block Overview 5-313
5-61. Command Processing - Control Block Overview 5-314
5-62. ENQ/DEQ Processing - Local Resources - Control Block Overview 5-314
5-63. ENQ/DEQ Processing - Global Resources - Control Block Overview 5-315
5-64. Queue Scanning Services - Local Resources - Control Block Overview 5-316
5-65. Queue Scanning Services - Global Resources - Control Block Overview 5-317
5-66. Storage Management - Control Block Overview 5-318
5-67. WTO/WTOR Message Processing - Control Block Overview 5-319
5-68. Module Flow Overview and Directory 5-321

5-69. Module Flow for CTC Processing - Handle Arrival of Immediate-CCW 5-322
5-70. Module Flow for CTC Processing - Handle Arrival of RSA or RSAIRCD 5-322
5-71. Module Flow for CTC Processing - Send a RSA or RSAIRCD 5-323
5-72. Module Flow for Ring Processing - Send/Receive a RSA 5-324
5-73. Module Flow for Ring Processing - Send a RSAIRCD or Immediate-CCW (Requested by ISGBCI) 5-325
5-74. Module Flow for Ring Processing - Send a RSAIRCD (Requested by ISGBTC) 5-326
5-75. Module Flow for Ring Processing - Handle Arrival of RSAIRCD (Not Requested by This System) 5-327
5-76. Module Flow for Ring Processing - SNAPSHOT Function 5-328
5-77. Module Flow for Ring Processing - SENDCMD (RSCRADDS) Function 5-329
5-78. Module Flow for Ring Processing - SENDCMD (RSCRSNAD) Function 5-330
5-79. Module Flow for Command Initialization and Cleanup 5-331
5-80. Module Flow for DISPLAY GRS 5-332
5-81. Module Flow for VARY GRS(x), PURGE 5-332
5-82. Module Flow for VARY GRS(x), QUIESCE to Another System 5-333
5-83. Module Flow for VARY GRS(x), QUIESCE by a System to Quiesce Itself 5-334
5-84. Module Flow for VARY GRS(x), RESTART to Restart Another System 5-335
5-85. Module Flow for VARY GRS(ALL), RESTART to Restart All Systems 5-336
5-86. Module Flow for VARY GRS(x), RESTART by a System Not in the Main Ring 5-337
5-87. Module Flow for Join Processing at Initialization Time 5-338
5-88. Module Flow for ENQ/DEQ Mainline - Local Resource Request 5-339
5-89. Module Flow for ENQ/DEQ Mainline - Global Resource Request 5-340
5-90. Module Flow for the Termination Resource Manager 5-341
5-91. Module Flow for Queue Scanning Services 5-342
5-92. Module Flow for Dump Support - SVC Dump 5-342
A-1. Page Fault Process Flow A-4
A-2. Swap-In Process Flow A-8
A-3. Swap-out Process Flow A-10
A-4. IOS/EXCP Process Flow A-13
A-5. VTAM SEND Process Flow A-19
A-6. Overview of Logon Processing A-22
A-7. TCAM Organization After a TSO Logon A-26
A-8. Logon Work Area A-28
A-9. LOGON Work Area Bits That Indicate the Currently Executing Module A-32
A-10. LOGON Scheduling Post Codes A-33
A-11. Overview of TSO Line Drop Process A-35
A-12. Summary of Command Processor Recovery Activity A-42
A-13. TSO Attention Flow A-47
B-1. Stand-alone Dump Analysis Flowchart B-5

Summary of Amendments

Summary of Amendments
for SY28-1133-2
as Updated December 27, 1985
by Technical Newsletter SN28-5095

This Technical Newsletter, which supports Version 1 Release 3.6 of MVS/System Product, includes HASPRAS as the issuing module of a $SDUMP macro instruction for JES2.

Additionally, several technical corrections were made.

Summary of Amendments
for SY28-1133-2
MVS/System Product Version 1 Release 3.5

This major revision consists of maintenance changes and changes to support MVS/System Product 1.3.5.

The changes include:

- The Service Processor Call SVC (SVC 122), which provides MVS support for:
  - Processor complexes with the Service Processor Architecture. For these processors, MVS issues the SERVICE CALL instruction to communicate with the service processor.
  - Processor complexes with the monitoring and system support facility (MSSF). (The Service Processor Call SVC was formerly called the MSSFCALL SVC.) For these processors, MVS issues the MSSFCALL DIAGNOSE instruction to communicate with the MSSF.
- Bit 19 in the machine check interruption code (MCIC).
- SVC dump titles for the scheduler, master scheduler, and TSO.
- Minor technical and editorial changes throughout.

- Information in "Section 5. Component Analysis" for the following components is deleted from this book because the information is obsolete or duplicated in other books.
  - JES2 - see the *JES2 Logic* book for diagnostic information.
  - VTAM - see the *ACF/VTAM Diagnosis Guide* and *ACF/VTAM Diagnosis Reference* for diagnostic information.

Summary of Amendments
for SY28-1133-1
as Updated December 30, 1983
by Technical Newsletter SN28-0875

This Technical Newsletter, which supports Version 1 Release 3.4 of MVS/System Product, contains information for the functional subsystem interface (FSI).

Also, changes have been made throughout this publication to reflect maintenance changes.

Section 1. General Introduction

This section introduces basic MVS problem analysis and provides an overview of the interactive problem control system (IPCS).

Basic MVS Problem Analysis Techniques

Problem isolation and determination are significantly more complex in MVS than in previous operating systems because of:

- ***Enabled System Design*** which has made the internal and environmental status-saving functions more extensive than those of previous systems.
- ***Multiprocessing (MP)*** which potentially allows the execution of code in sequences not encountered in a uniprocessing (UP) environment. MP can also cause contention for serially reusable resources. (In this manual, MP refers to multiprocessing on both multiprocessors and attached processors.)
- ***Locking Mechanism*** which facilitates enabled system design and multiprocessing functions and maintains data integrity.
- ***Subsystems*** which are responsible for processing work requested from the system. They maintain their own work queues, control block structures and dispatching mechanisms - all of which must be understood in order to effectively pursue problems in the MVS operating system.
- ***Software Recovery*** which attempts to keep the system available despite errors.
- ***The number of components*** which provide new functions and whose internal logic must be understood for effective problem determination.

As a result of this complexity, MVS problem solvers have made two adjustments in their diagnostic outlook:

- Rather than learning the system logic at an instruction or module level, they have learned the system in terms of component interactions at the interface level.
- They have learned that the most effective problem analysis at a system level is obtained from a disciplined, almost formal, diagnostic approach.

This publication contains those debugging techniques and guidelines that have proven the most useful to problem solvers with several years experience in analyzing MVS system problems. These techniques are presented in terms of a debugging "approach" that can be summarized in three steps:

1. Identifying the external symptom of the problem.

2. Gathering relevant data from system data areas in order to isolate the problem to a component.

3. Analyzing the component to determine the cause of the problem.

The most important step in this approach is often the first - correctly identifying the external symptom of a problem. To do this, it is best to get a description of the problem as it was perceived by an eyewitness. You will want a description that provides a context from which to start, such as:

"System is looping; can't get in from console."
"Job abended with 213."
"I/O error on 251."
"Console locked out."
"Terminal hung, keyboard locked."
"System in wait, nothing running."
"Bad output."
"Job won't cancel."
"System degrading. Very slow."
"System died."
"0C4 in component abc."

The list is endless, of course. Your objective is to fit one (or more) of these descriptions to one of the following external symptoms.

- ***Enabled wait*** - The system is not executing any work and when it takes interrupts, nothing happens. Something appears to be stuck.

- ***Disabled wait*** - The system freezes with a disabled PSW that has the wait bit on. This can be either an explicit and intentional disabled wait or a situation that occurs because the PSW area has been overlaid.

- ***Disabled Loop*** - This is normally a small (fewer than 50 instructions) loop in disabled code.

- ***Enabled loop*** - This is normally a large loop in enabled code (and may include disabled portions - loops as a result of interrupts).

- ***Program check*** - The program is automatically cancelled by the system, usually because of improper specification or incorrect use of instructions or data in the program. If a SYSABEND, SYSMDUMP, or SYSUDUMP DD statement was included in the JCL for the job, a dump of the problem program will be taken.

- ***ABEND*** - The system issues an SVC 13 with a specific code from 1 to 4095 to indicate an abnormal situation.

- ***Incorrect output*** - The system is not producing expected output. Incorrect output can be categorized as: missing records, duplicate records or invalid data that has sequence errors, incorrect values, format errors, or meaningless data. If a program has apparently executed successfully, incorrect results will not be detected until the data is used at some future time.

- ***Performance degradation*** - A bottleneck or system failure (hardware or software) has severely degraded job execution and throughput.

- ***TP problem*** - A problem, usually detected by the operator or terminal user, that indicates malfunctions are affecting one or more terminals, lines, etc.

The chapters in Section 4 (Symptom Analysis Approach) will help you identify these symptoms. The main rule at this stage of your analysis is to proceed carefully. When first screening a problem, do not assume too much. Don't even assume that the original eye witness description was correct. Keep all initial information about the problem as a reference for your later analysis.

In the course of identifying the correct external symptom, you will begin gathering data that will lead you to other sections of the publication. Specific data gathering techniques are contained in Sections 2 and 3. Section 2 describes the major MVS debugging areas such as LOGREC records and recovery work areas. Section 3 describes how to use a storage dump effectively as your main source of diagnostic material.

Eventually you should have gathered enough data to isolate the problem to a particular component or process. Section 5 and Appendix A provide techniques for analyzing system components and processes so that you can determine the cause of the problem. Appendix B contains a step-by-step procedure that can be used as a guide for analyzing a stand-alone dump.

Note: Before you begin using this publication for problem analysis, scan through it to find out where the various types of information are located. Depending on your current debugging skill level, various sections will be more important than others.

Always keep in mind that trouble-shooting a system of the internal complexity of MVS is not always an "If A, then B" procedure. The guidelines and techniques presented in this publication define "generally" what the analyst will discover. The nature of the debugging process is such that the problem solver does not perform the same analysis for every problem.

IPCS - Interactive Problem Control System

The Interactive Problem Control System (IPCS) provides MVS installations with expanded capabilities for diagnosing software failures and facilities for managing problem information and status.

IPCS includes facilities for:

- Online examination of storage dumps.
- Analysis of key MVS system components and control blocks.
- Online management of a directory of software problems that have occurred in the user's system.
- Online management of a directory of problem-related data, such as dumps or the output of service aids.

IPCS runs as a command processor under TSO, allowing the user to make use of existing TSO facilities from IPCS, including the ability to create and execute command procedures (CLISTs) containing the IPCS command and its subcommands.

IPCS supports three forms of MVS storage dumps:

- High-speed stand-alone dumps produced by AMDSADMP.
- Virtual dumps produced by MVS SDUMP on SYS1.DUMP data sets.
- Virtual dumps produced by MVS SDUMP on data sets specified by the SYSMDUMP DD statement.

Dumps on data sets specified by the SYSABEND or SYSUDUMP DD statements cannot be analyzed using the IPCS facilities.

For information about IPCS, refer to the *MVS Interactive Problem Control System (IPCS) User's Guide and Reference.*

Section 2. Important Considerations Unique to MVS

This section describes concepts and functions that are unique to the MVS environment and useful to problem analysis. It also contains miscellaneous debugging hints and general data gathering techniques.

The chapters in this section are:

- Global System Analysis
- System Execution Modes and Status Saving
- Locking
- Use of Recovery Work Areas in Problem Analysis
- Effects of Multiprocessing on Problem Analysis
- MVS Trace Analysis
- Miscellaneous Debugging Hints
- Additional Data Gathering Techniques

Global System Analysis

In trying to isolate a problem to an internal symptom, a global system analysis often uncovers enough data to provide a starting point for the actual problem isolation and debugging. This chapter discusses the main considerations the analyst should be aware of when analyzing a stand-alone dump, including:

- The system areas that should be inspected to understand the current system state at the time of a dump

- The system areas that should be examined to understand the current state of the work in the system and the current disposition of storage and tasks

Global Indicators that Determine the Current System State

The following areas should be examined to help determine the current state of the system:

1. **PSA** - occupies the first 4K bytes of real storage for each processor. Note that absolute 0 is not used during normal system operation on a machine with the MP feature - this is true whether the system is operating in MP or UP. (The one exception is a control program that is system generated with ACRCODE=NO.) During NIP processing the PSA(s) for the processor(s) are initialized and the prefix register(s) are initialized to point to them.

   Special Notes About Stand-alone Dumps:

   - If you IPL the stand-alone dump program from the system control (SC) frame on the 3033 or the CC012 frame of the 3081, it is not necessary to perform the STORE STATUS operation as noted in the following paragraphs. Status is automatically stored when stand-alone dump is invoked from either of these frames and automatic store status is on.

   - Before taking a stand-alone dump, it is necessary to perform a STORE STATUS operation. This hardware facility does *not* use prefixing; instead it stores values such as the current PSW, registers, CPU timer, and clock comparator in the unprefixed PSA (the one used before NIP initialized the prefix register) at absolute address 100. The dump program subsequently saves these values and, in an MP environment, issues a SIGP instruction to other processors requesting a STORE STATUS operation. As a result, these values in the unprefixed PSA are *overlaid* by another processor's values.

     Therefore, in an MP environment the status in the unprefixed PSA is always that of a non-IPLed processor, not the one on which the stand-alone dump was IPLed.

   - In a machine not equipped with the MP feature and therefore without prefixing, the IPLing of the stand-alone dump program causes low storage (0-X'18') to be overlaid with CCWs. You should be aware of this and not consider it as a low storage overlay.

- In an MP environment, the STORE STATUS operation must be performed only from the processor to be IPLed for the stand-alone dump program.

- IPLing the stand-alone dump program twice causes the storage dump to contain a dump of the dump program itself because it was read in for the first IPL. This causes the dump program to overlay a certain portion of the nucleus (generally starting at X'7000') and the general purpose registers to contain values associated with the stand-alone dump program and *not* MVS.

- If the operator does not issue the STORE STATUS instruction before IPLing a stand-alone dump, the message "ONLY GENERAL PURPOSE REGS VALID" might appear on the formatted dump. The PSW, control registers, etc., are not included. This greatly hampers the debugger's task.

2. **Registers and PSW** - The print dump program formats the current PSW and the general, floating point, and control registers associated with each processor. From these, you can determine the program executing on each processor.

   If the current PSW is 070E0000 00000000 and the GPRs are all 0, you are in the no-work wait condition, which indicates no ready work is available for this processor to execute.

   If there is or should be work remaining, an invalid wait condition results. (Refer to the chapter on "Waits" in Section 4.)

   If the registers are not equal to zero and the PSW does not contain the wait bit (X'0002'), there is an active program. If the wait task is dispatched, the system is in the no-work wait condition.

3. **ILC/CC** - location X'84' for external interrupts; location X'88' for SVC interrupts; location X'8C' for program interrupts. These fields indicate the last type of interrupt associated with each interrupt class for each processor. The work active when each interrupt occurs is represented by the old PSWs at locations: X'18' (external); X'20' (SVC); X'28' (program). Common contents of these fields are:

X'84' 00001004 clock comparator
00001005 CPU timer
000x1201 SIGP-emergency signal
000x1202 SIGP-external call
Where x indicates the processor issuing the SIGP instruction.

X'88' 000200xx where xx is the SVC number. This field should be inspected for unusual SVCs such as:

1 - WAIT:	can indicate an enabled wait situation
D - ABEND:	can indicate program error processing
F - ERREXCP:	can indicate a problem in I/O error processing
10 - PURGE:	can indicate a problem in the swap process
38 - ENQ:	can indicate a resource contention problem
4F - STATUS:	can indicate a non-dispatchability problem

X'8C' 000X0011 indicates a page fault interrupt. Anything other than a code of 11 is highly suspect and must be inspected further. Also with a code of 11, the program check old PSW (location X'28') must be enabled (mask = X'07') because disabled page faults are not allowed in MVS and it is an error if one occurs.

4. **PSA+X'204' (CPU ID)**

5. **PSA+X'208' (address of PCCA - 1 per processor)** - The PCCA contains information about the physical facilities of each processor.

6. **PSA+X'210' (address of LCCA - 1 per processor)** - The LCCA contains many of the status-saving areas that were located in low storage in previous systems. It is used for software environment saving and indications. The registers associated with each of the interrupts you find in the PSA are saved in this area. In addition, the system mode indicators for each processor are maintained in the LCCA.

7. **PSA+X'224' (PSAAOLD)** - This is the address of the ASCB of the work last dispatched on each processor. This field indicates the address space that is currently executing.

8. **PSA+X'21C' (PSATOLD)** - This is the address of the TCB of the work last dispatched on each processor. This field in conjunction with PSAAOLD isolates to a task within an address space. **Note:** PSATOLD=0 when SRBs are dispatched.

9. **PSA+X'228' (PSASUPER)** - This is a field of bits that represent various supervisory functions in the system. If a loop is suspected, these bits should be checked in an attempt to isolate the looping process.

   Note: Because of SRM timer processing in MVS, the external first level interrupt handler bit (X'20') or the dispatcher bit (X'04') may be set in this field even in the enabled wait situation.

10. **PSA+X'2F8' (PSACLHS)** - This field indicates the current locks held on each processor. Knowing which locks are held helps isolate the problem, especially in a loop situation. By determining the lock holders you can isolate the current process. (See the chapter on "Locking" later in this section.)

11. **PSA+X'380' (PSACSTK)** - This is the address of the active recovery stack which contains the address of the recovery routines to be routed control in case of an error. If the address is other than X'C00' (normal stack), the type of stack (for example, program check FLIH or restart FLIH) is meaningful, especially in the loop situation.

    By searching the normal stack (X'C00') and associating the recovery routine to active mainline routines you may get an idea of the current process. This is true only if the pointer to the current entry is not X'CE0,' which would indicate an empty recovery stack.

    Note: If a loop is suspected, the first word following each routine address in the current stack should be scanned. A X'80' indicates that routine is in control. A X'40' indicates that routine is in control and that it is a nested recovery routine.

    If X'28' into the stack is non-zero, also check for an SDWA address at X'5C' into the active stack. This block is mapped by the SDWA DSECT and is described in the *Debugging Handbook* (RTCA and SDWA are different names for the same control block). If an SDWA address is present, an error has occurred and it can be related to the problem you are analyzing. If trapping via RTM's SLIP facility, the registers at entry to RTM are contained in this area.

12. **PSA+X'414' (PSACSID)** - This is the ID of the channel set that is currently connected to this processor.

Work Queues, TCBs and Address Space Analysis

Examine the following areas to help determine the current state of work in the system.

TCB Summary

The TCB summary report, produced by AMDPRDMP (print dump program), contains a summary of the address spaces and their associated tasks. A quick scan of the completion (CMP) field for each task reveals any abnormal terminations that have occurred. Discovery of an error completion code warrants further investigation as to the cause. Remember, however, that these codes are residual and the job or task might have recovered from the problem.

Also investigate multiple abnormal completion codes which all relate to the same area of the system, or many tasks that all have the same completion code. These completion codes can all relate to one area of the system and perhaps to the problem you are investigating. Again, LOGREC should provide further documentation in an error situation such as this.

SRB Dispatching Queues

The print dump program formats the SRB dispatching queues. Elements on any of these queues should be investigated, especially in cases where no work appears to be progressing through the system.

Elements on the global, SVT, or address space local services management queues (SVTGSMQ, SVTLSMQ, or ASCBLSMQ) can indicate that the dispatcher has not received control since these SRBs were scheduled. This is an unusual condition that should be investigated to determine why the SRBs have not been dispatched.

Elements on the global/local service priority lists (GSPLs/LSPLs) should be explained. It is possible the dump was taken before the SRB routines were able to execute. But it more likely indicates some other system problem such as an enabled wait or disabled loop. If there are SRBs on an LSMQ/LSPL, you should determine if the associated address space is swapped into storage and if it is not, why not. (Possible causes are real frame shortage or a problem in the paging/swapping mechanism.) Again this is an indication of a potential system problem. The chapter on "Waits" in Section 4 and the chapter on "Dispatcher" in Section 5 contain additional information on the dispatching queues.

If, at this point, you can isolate the problem to a component, refer to the "Component Analysis" for that component in Section 5. The chapter on "Waits" in Section 4 should prove helpful if you have isolated to a problem in the system.

Address Space Analysis

If you have isolated the error to a given address space or want to determine the state of a given address space, analyze the ASCB.

Important indicators in the ASCB are:

- ASCBLOCK (ASCB+X'80') - to determine the specific state of the local lock. If it contains 7FFFFFFF, FFFFFFFF, or 4FFFFFFF (the lock suspend/interrupt/ready-to-run IDs), refer to the chapter on "Locking" later in this section for an explanation.

  Note: When holding a suspend lock, a routine can only be suspended because it attempts to obtain an unavailable cross memory services lock or because of a page fault, synchronous page fix, or if the SMF buffers are full when SMF is entered, or the routine specifies SUSPEND=YES on the SDUMP macro. To find the reason for the suspension, refer to the discussion of Task Analysis later in this chapter and to the chapter on "Locking" later in this section.

- ASCBEWST (ASCB+X'48') - to determine the TOD clock value when the address space last executed. This field helps you determine how long an address space has been swapped-out. By subtracting this field (bytes 3-6 which repeat approximately every 15 minutes) from the last timer value in the MVS trace table and converting to seconds, you can discover the approximate swap-out time. (See the chapter "MVS Trace Analysis" later in this section.)

- ASCBTNEW (ASCB+X'1C') - identifies highest-priority TCB that is dispatchable. Explicit wait sets a non-dispatchability flag (TCBFLGS4=X'04').

- ASCBCPUS (ASCB+X'20') - number of processors running tasks in this address space.

- ASCBSEQN (ASCB+X'26') - indicates the address space's position on the dispatching queue. If its value is X'7FFF' the ASCB is not on the dispatching queue.

- ASCBRCTF (ASCB+X'66'), ASCBFLG1 (ASCB+X'67') - current status of the address space.

- ASCBASXB (ASCB+X'6C') - pointer to the ASXB that anchors the TCBs.

- ASCBDSP1 (ASCB+X'72') - address space non-dispatching flags.

- ASCBSRBS (ASCB+X'76') - number of SRBs currently suspended in the address space.

- ASCBOUCB (ASCB+X'90') - pointer to the OUCB, which is helpful when determining why an address space is swapped-out.

- ASCBFMCT (ASCB+X'98') - number of real frames currently occupied by the address space.

- ASCBTCBS (ASCB+X'D8') - number of ready TCBs not requiring the local lock.

- ASCBTCBL (ASCB+X'DC') - number of ready TCBs requiring the local lock.

- ASCBLOCI (ASCB+X'E8') - contains either the ASCB address of the address space holding this ASCB's local lock as a CML lock, or zero.

- ASCBCMLH (ASCB+X'EC') - contains either the address of the suspended TCB or SSRB holding this ASCB's local lock, or zero. The high-order bit on in field ASCBCMLH indicates an SSRB.

Task Analysis

Once you understand the ASCB you should analyze the associated task structure. Once again, scan the TCBs associated with your address space and look for an abnormal completion field. While doing so, check the RB structure for each task. Remember that the region control task, dump task, and STC/LOGON are represented by the first three TCBs. "Normally" they will be waiting during task execution. If one of them is not, you should determine why.

Assuming the first three TCBs are not obvious problem areas, continue inspecting the remaining TCBs. You are trying to explain each RB. Starting with the last RB created (the first RB, pointed to by the TCB+0), determine what work is represented. If work is waiting, find out why.

Note: The master scheduler address space has system task TCBs that differ from other address spaces. Refer to the diagrams for Master Scheduler Initialization, Start Initiator, and Job Execution in the topic "General System Flow" in the *Debugging Handbook,* Volume 1 for details of the TCB structures.

The RBOPSW indicates the issuer of an explicit WAIT. TCBFLGS4 indicates an explicit WAIT. If it is not an explicit WAIT, consider the following suspension possibilities and their associated key indicators:

1. If ASCBLOCK=X'7FFFFFFF', X'FFFFFFFF', or X'4FFFFFFF', the status (registers and PSW) of the suspended, interrupted, or ready-to-run task is saved in the IHSA of the locally locked address space (ASCB+X'6C' points to ASXB; ASXB+X'20' points to IHSA). The IHSA is serialized by the address space's local lock. The reason for suspension is important. If it is for a lock, find out what address space or task owns that lock and what the owners' state is. (The chapter on "Locking" later in this section shows how to determine lock owners.) If it is for a page fault or synchronous page fix, determine the state of that page fault or synchronous page fix. If it is for an SMF suspension, the field at SMCA+X'8C' points to an SMF suspend block (SSB). (For additional information on SMF suspension, see the topic "SMF Suspension" later in this section.) Note also that while the RBTRANS field points to the page fault causing address, the RBWCF is 0.

   Note: If a task owned the local lock at the time of the suspension or interruption, TCBACTIV is left on. If no TCB in the task structure has an active indicator set, you can assume an SRB owned the lock. If no SRBs are on either of the cross memory services lock suspend queue, the suspension is probably the result of a page fault or a synchronous page fix.

An SRB can be suspended requesting an unavailable suspend lock (local or cross memory services), or because of a page fault or a page fix. Once an executing SRB is suspended for any of the above reasons, an SSRB (see the *Debugging Handbook*) is constructed. Also, an SRB can be delayed by the dispatcher if the local lock is unavailable and the SRB is to receive control with the local lock held. No status is saved for a delayed SRB; instead the SRB is placed on the local lock suspend queue. If suspended for page fault processing, the SSRB is pointed to by the corresponding PCB+X'1C' (PCBSRB). PCBs are generally chained together and anchored in two locations: (1) the RSMHDR for local address space page faults; (2) the PVT for page faults caused by referencing commonly addressable storage. Note that if real frames were not available when the page fault occurred, even local page faults are queued from the PVT on the defer queue (PVTGFADF, PVT+X'754').

For a cross memory services lock request, the SSRB is on the requested cross memory services lock's suspend queue. See the chapter on "Waits" in Section 4 for details on how to locate the SSRB. For Local lock suspensions, the SRBs and SSRBs are chained together on a queue anchored in the ASCB field ASCBLSQH (ASCB+X'84').

A locked TCB can be suspended for the same reasons as an SRB. The save area is the IHSA of the locally locked address space (described in the *Debugging Handbook*). The IHSA is valid during a page fault if the corresponding PCB+X'08' flag is on, indicating the lock was held at the time of the page fault. Also, the TCBLLH (TCB+X'114') is set to X'01' if the task was locally locked at the time of the page fault.

The IHSA is valid for a cross memory services lock suspension if the ASCB is on the cross memory services lock's suspend queue. The CMSSMF lock suspend queue header is at label CMSFRSQH, the ENQ/DEQ cross memory services lock suspend queue is at label CMSEDLK+X'4', and the general cross memory services lock suspend queue is at label CMSSQH in CSECT IEAVESLA. If there is a page fault, the TCB could be suspended while holding both the local lock and at least one cross memory services lock. An indication of this is that the flag for cross memory services locks (ASCB+X'2A') is turned on, and the ASCB address is in at least one of the cross memory services locks. The cross memory services lockword contains the ASCB address of the locally locked address space. The requester of the cross memory services lock can own a cross memory local (CML) lock.

Note: The local and cross memory services lock bits in ASCBHLHI are set at suspend and are never reset.

2. If ASCBLOCK=X'00000000' and the memory/task is waiting, the status is saved in the RB/TCB. (See the chapter on "System Execution Modes and Status Saving" later in this section.)

   A task can issue the SUSPEND macro to cause the wait count of an RB to be non-zero. The RBOPSW indicates the issuer of a SUSPEND RB=CURRENT request. However, an RB can also issue SUSPEND

RB=PREVIOUS. In this case, the only clue to the issuer is the interrupt code in the RB. If the interrupt code indicates a type 2, 3, or 4 SVC, then this SVC routine could have issued the suspend for this RB; but it is also possible that some IRB had executed on behalf of the task and issued a suspend for its previous RB.

If the RBOPSW does not indicate the issuer of a WAIT or SUSPEND, and the RB is not in either page fault wait or page fix wait, then a SUSPEND RB=PREVIOUS might have been issued.

Note: The explicit wait flag in the TCB (in TCBFLGS4) will not be on for a suspended RB.

3. Suspended SRBs can cause bottlenecks. The chapter on "System Execution Modes and Status Saving" can aid in locating any suspended SRBs that relate to the address space. **Note:** Do not spend time looking for them unless other facts about the problem indicate a potential problem in this area.

By far the most important consideration in task analysis is the RB structure of each task. Generally if you have isolated the problem to an address space, RB analysis shows a potential problem in the way of:

- Long RB chains
- Contention caused by an ENQ (SVC 38) request
- SMF suspension
- Page fault or synchronous page fix waits
- I/O waits
- Abnormal termination processing, that is, SVC D RB

Once you have analyzed the RB structure you might want to go back and further analyze the TCBs. Following are additional important fields in the TCB:

1. TCBFLGS (TCB+X'1D') - indicators of how the system currently considers this task.

2. TCBGRS (TCB+X'30') - general purpose registers (0-15) saved when a TYPE 1 SVC is issued or for an interruption for a non-locked task.

3. TCBSCNDY (TCB+X'AC') - additional system indicators for this task that help to determine why this task is not executing.

4. TCBRTWA (TCB+X'E0') - pointer to the RTM2 work area (mapped in the *Debugging Handbook*) which contains information similar to the SDWA but also data for RTM processing.

Summary

This chapter contains major considerations you must be aware of when analyzing a stand-alone dump in MVS. A disciplined approach is important; resist the tendency to go off on tangents upon finding the first unexplainable condition. After gathering all the facts, try to resolve the "cause and effect" situations you are bound to uncover. Generally, at this point you will have isolated the error and can start a detailed component/process analysis.

System Execution Modes and Status Saving

MVS differs significantly from previous operating systems by having multiple execution modes. Status is saved and restored from many different locations depending upon the execution mode at the time control was lost. This chapter explains those modes and how they affect problem analysis.

System Execution Modes

MVS has four execution modes:

- Task mode
- SRB mode
- Physically disabled mode
- Locked mode

Code always executes in one of these modes or, in certain cases, in a combination of modes. For instance, code running in task or SRB mode can also be either locally locked or physically disabled.

In general, the supervisor dispatches units of work according to the following priority: SRB, locked, and task mode. Because a unit of work that is disabled is already executing, disabled mode work is not dispatched as such.

When a unit of work is running, the locally locked ASCB is found through PSALOCAL or PSAAOLD. If PSALOCAL=0 and PSACLHS indicates that a local lock is held, then PSAAOLD points to the locked ASCB. If PSALOCAL"0 then PSALOCAL points to the CML locked address space.

In conjunction with the four execution modes, a unit of work can execute in cross memory mode. Cross memory mode is defined by control registers 3 and 4 and the PSW S-bit. The S-bit (bit 16 of the PSW) indicates whether current addressability is to the primary address space (S-bit=0) or the secondary address space (S-bit=1). The primary and secondary address spaces are defined by the ASIDs in control registers 3 and 4. The home address space is the address space in which the unit of work resides (indicated by PSAAOLD) when that unit of work is executing. When primary and secondary addressability is to the home address space and the S-bit=0, then the unit of work is not in cross memory mode.

Task Mode

Task mode describes code that is executing in the system because the dispatcher selected work from the task control block (TCB) chain or from the interrupt handler save area (if the interrupted TCB held a local lock). To start execution, the dispatcher sets up the environment (registers, PSW, cross memory state, PCLINK stack, and FRR stack) and then passes control to the code to be executed.

1. Information for an unlocked task dispatch environment is found as follows:

TCB+X'30' (TCBGRS)	- General purpose registers.
TCB+X'0' (TCBRBP)	- Address of the RB.
RB+X'10' (RBOPSW)	- Old PSW.
RB-X'20' (RBXSB)	- Address of the XSB.
XSB+X'8' (XSBXMCRS)	- Cross memory status.
XSB+X'18' (XSBSTKE)	- PCLINK stack header.
TCB+X'E4' (TCBNSSP)	- Address of the NSSA.
NSSA+X'C' (NSSAFRRS)	- FRR stack for an enabled unlocked task mode FRR.

2. Information for a locally locked or a CML locked task dispatch environment is found in the locally locked address space as follows:

   - From the ASCB of the locally locked address space:

ASCB+X'E8' (ASCBLOCI)	Contains either the address of the ASCB holding this ASCB's local lock as a CML lock, or zero if this ASCB's local lock is held as a LOCAL lock.
ASCB+X'EC' (ASCBCMLH)	Address of the TCB holding this ASCB's local lock.

   - From the task holding a local lock:

TCB+X'E8' (TCBXLAS)	Contains either the address of the ASCB of the locally locked address space, or zero if holding the LOCAL lock.
ASCB+X'6C' (ASCBASXB)	Address of the ASXB.
ASXB+X'20' (ASXBIHSA)	Address of the IHSA.
IHSA+X'38' (IHSAGPRS)	General purpose registers.
IHSA+X'10' (IHSACPSW)	PSW for the redispatched task.
IHSA+X'80' (IHSAXSB)	Address of the XSB.
XSB+X'8' (XSBXMCRS)	Cross memory status.
XSB+X'18' (XSBSTKE)	PCLINK stack header.
IHSA+X'8C' (IHSAFRRS)	FRR stack.

Task mode is probably the most common execution mode. All programs given control via ATTACH, LINK, and XCTL operate in this mode.

SRB Mode

SRB (service request block) mode describes code that is executing in the system because the dispatcher found an SRB on one of the SRB queues. SRB set-up is started by the SCHEDULE macro. SCHEDULE is a macro that places the requestor-furnished SRB directly on the queue or, alternatively, calls a routine to do so. SRBs are generally placed on the service management queue (SMQ), unless both the SMQ and the service priority list (SPL) are empty, in which case the SRB is placed on the SPL. The global services management queue (GSMQ) is located at SVTGSMQ (SVT+X'20'). It is also pointed to by CVTGSMQ (CVT+X'264'). The global service priority list (GSPL) is located at SVTGSPL (SVT+X'24') and can also be found from CVTGSPL (CVT+X'26C'). The SVT local service management queue (LSMQ) is located at SVTLSMQ (SVT+X'28'), and can be found from CVTLSMQ (CVT+X'268'). Finally, there is one local

SMQ and one local SPL per address space. ASCBLSMQ is located at ASCB+X'D0', and ASCBLSPL is located at ASCB+X'D4'. An SRB scheduled globally for a swapped-out address space is moved to one of the local queues.

SRBs are selected from the SPLs by the dispatcher in order to start execution. The dispatcher loads registers 0, 1, 14, and 15 from information in the SRB and builds the PSW. The PSW key and address are the responsibility of the scheduler of the SRB and are specified in the SRB. SRB mode has the characteristics of being enabled, supervisor state, key requested and non-preemptable. Non-preemptable means that the interrupt handler should return control to the interrupted service routine (code running under SRB mode). However, service routines can be suspended because of a page fault or because a lock (cross memory services or local) is unavailable.

SRB is interrupted. SRBs are non-preemptable. The registers, PSW, and cross memory status are saved in the PSA during interrupt processing. When the system has handled the interrupt, the SLIHs return to the FLIHs, the status is restored from the PSA, and control is returned to the interrupted SRB routine.

SRB is suspended. SRBs that are suspended must have their status saved in a unique area. The process that suspends an SRB is responsible for obtaining an SSRB (suspended SRB) and XSB (extended status block), which will contain the interrupted status used to reschedule the service routine once the reason for suspension has been resolved. See "Locating Status Information in a Storage Dump" later in this chapter for a detailed description of how to find these SSRBs and XSBs.

Physically Disabled Mode

Disabled mode is reserved for high-priority system code whose function is the manipulation of critical system queues and data areas. It is usually combined with supervisor state and key 0 in the PSW, and assures that the routine running disabled is able to complete its function before losing control. It is restricted to just a few modules in MVS (for example, interrupt handlers, the dispatcher, and programs holding a global spin lock).

Physically disabled mode is used for one of two reasons:

1. To assure that data remains static while the code is referencing or updating the data.

2. To assure that non-reentrant code does not lose control while performing critical system functions. For example, IOS must run disabled while enqueueing and dequeueing requests to UCBs and while updating UCBs at the start and end of I/O operations.

In the MVS system, physical disablement on a system basis because of MP must be accompanied by locking in order to guarantee serialization. MVS disabled code is usually accompanied by either a global spin lock or code executing under a "super bit." The "super bits" are located in each processor's PSA (X'228'). They are used primarily for recovery reasons - they allow RTM to recognize that a disabled supervisory function was in control at the time of error even though global locks were not held. This indicates that FRR recovery processing should be initiated by RTM.

Note that type 1 SVCs do not execute disabled in MVS. Instead they are entered with the local lock. Thus they are considered to be task mode physically enabled, holding the local lock.

Type 6 SVCs execute disabled. They are considered to be logical extensions of the SVC FLIH and execute with all the restrictions (that is, cannot page fault, etc.) of a disabled function.

Locked Mode

Locked mode describes code executing in the system while owning a lock. (See the chapter on "Locking" later in this section.) A lock can be requested during any execution mode (SRB, TCB, physically disabled).

Status saving while in a locked mode requires unique considerations from the system. An example is a program that invokes a type 1 SVC, such as EXCP or WAIT, that executes in locked mode. When a type 1 SVC is enabled, it can be interrupted. However, if the SVC is interrupted, the registers cannot be saved in the TCB because it is being used to save registers active at the time of the SVC request for return to the requestor. Therefore, status must be saved elsewhere.

Status saving while in locked mode is described under the previous topics "Task Mode" and "SRB Mode."

Determining Execution Mode From a Stand-alone Dump

Knowing the system's execution mode at the time a stand-alone dump was taken is important in analyzing a disabled coded wait state or a loop. The following areas may help determine the mode of execution:

LCCA Indicators

There is an important dispatcher flag byte at LCCA+X'21D'. For a global SRB, the LCCAGSRB and LCCASRBM flags are set on. For a local SRB, only the LCCASRBM flag is set on.

PSA Indicators

- Super Bits - Flags in the supervisor control field located at PSA+X'228' (PSASUPER) indicate whether the dump was taken while in one of the interrupt handlers or dispatcher. The dispatcher's super bit is left on when the wait task is dispatched.

- PSAMODE - PSA+X'49F' indicates the mode of the system:

X'00'	- Task mode
X'04'	- SRB mode
X'08'	- Wait mode
X'0C'	- I/O recursion mode
X'10'	- Dispatcher mode
X'20'	- Non-preemptive bit (can be on with any of the above bits)

- Recovery Stack - If the first two words of the RTM stack vector table (PSA+X'380') are *not* equal, then control is in one of the interrupt handlers or the dispatcher. The dispatcher stack is current when the wait task is dispatched. Compare the address at PSA + X‘380’ with each entry in the FRR stack vector table starting at PSA + X‘384’ to determine the owner of the active stack. (See the chapter on “Use of Recovery Work Areas for Problem Analysis” later in this section for stack vector table analysis.)

- Current Work - PSA + X‘218’ contains the addresses of the new TCB, old TCB, new ASCB and old ASCB consecutively in a four-word area. If the system is in SRB mode, the address of the old TCB equals 0. If the addresses of the new and old ASCBs are *not* equal, then the stand-alone dump was taken between the time that an address space switch was requested and the time that the dispatcher dispatched an address space, a global SRB, or the wait task. In all cases, the old TCB and ASCB indicate the current work.

- Locks - The PSA also contains the lock indicators. (See the chapter on “Locking” later in this section for a description of how to determine the lock mode.)

ASCB Indicators

The following ASCB locations help determine execution mode:

X‘26’ Set to X‘7FFF’ indicates that the address space is not on the dispatching queue.

X‘66-67’ RCT flags.

X‘72-73’ Non-dispatchability flags.

X‘76’ Count of SRBs suspended in this address space.

X‘80’ Local lock (see “Locking” later in this section for how to interpret this field when ”0).

X‘84’ Address of the SRB suspend queue for local lock requestors.

X‘D0’ Local service management queue (contains SRBs that have not been staged). When the high-order bit of this field is 1, it indicates that a “user-ready” SYSEVENT is required to swap in the address space.

X‘D4’ Local service priority list (contains SRBs that have been staged).

X‘D8’ Number of ready TCBs that do not require the local lock.

X‘DC’ Number of ready TCBs that require the local lock.

X‘E8’ Contains either the ASCB address of the address space holding this ASCB′s local lock as a CML lock, or zero. If nonzero, then the lock is owned by a unit of work in the address space pointed to by this field. If zero, then the lock is not owned or is owned by a unit of work within this address space.

X‘EC’ Contains either the address of the suspended TCB or SSRB that is holding this ASCB′s local lock, or zero. The high-order bit (bit 0) on indicates an SSRB.

Keep in mind that mixed modes frequently occur. For example, a local SRB can obtain a lock, be interrupted, and the stand-alone dump taken while disabled in the I/O supervisor. Depending on the system mode at the time of the interrupt, a task′s status (registers, PSW, etc.) can be saved in one of several places.

Locating Status Information in a Storage Dump

Status information is located in a storage dump depending on the conditions under which it was saved.

Task and SRB Mode Interruptions

Status saving is required whenever the code gives up control, whether voluntarily or involuntarily. Initial status is saved by the first level interrupt handler (FLIH) as follows:

SVC FLIH - Initially:

- Registers 7-9 saved at PSA+X'22C' (PSAGPREG)
- If an error condition is found, registers saved at LCCA+X'380' (LCCASGPR).

Then for all SVCs, status is saved in the TCB and the requestor's RB and XSB:

- Registers 0-15 saved at TCB+X'30' (TCBGRS)
- PSW saved at requestor's RB+X'10' (RBOPSW)
- Cross memory status saved at XSB+X'8' (XSBXMCRS)
- PCLINK stack header saved at XSB+X'18' (XSBSTKE).

Then for Type 2, 3, and 4 SVCs:

- Registers 0-15 saved at SVRB+X'20' (RBGRSAVE).

I/O FLIH - Initially:

- Register 1 saved at PSA+X'22C' (PSAGPREG).

Then for unlocked tasks, status is saved in the TCB, RB, and XSB:

- Registers 0-15 saved in TCB+X'30' (TCBGRS)
- PSW saved at RB+X'10' (RBOPSW)
- Cross memory status saved at XSB+X'8' (XSBXMCRS).

For locally locked tasks, status is saved in the IHSA and XSB of the locked address space:

- Registers 0-15 saved at IHSA+X'38' (IHSAGPRS)
- PSW saved at IHSA+X'10' (IHSACPSW)
- Cross memory status saved at XSB+X'8' (XSBXMCRS).

For SRBs and non-preemptive TCBs:

- Register 0-15 saved at PSA+X'678' (PSAGGRSV)
- PSW saved at PSA+X'300' (PSASVPSW)
- Cross memory status saved at PSA+X'5A8' (PGSAGXMSV).

External FLIH - Initially:

- Registers 14 and 15 saved at PSA+X'230' (PSAGPREG).

Then for locally locked tasks, status is saved in the IHSA and XSB of the locked address space:

- Registers 0-15 saved at IHSA+X'38' (IHSAGPRS)
- PSW saved at IHSA+X'10' (IHSACPSW).
- Cross memory status saved at XSB+X'8' (XSBXMCRS).

For unlocked tasks, status is saved in the TCB, RB, and XSB:

- Registers 0-15 saved at TCB+X'30' (TCBGRS)
- PSW saved at RB+X'10' (RBOPSW)
- Cross memory status saved at XSB+X'8' (XSBXMCRS).

For SRBs and non-preemptive TCBs:

- Registers 0-15 saved at PSA+X'678' (PSAGGRSV)
- PSW saved at PSA+X'240' (PSAEXPS1)
- Cross memory status saved at PSA+X'5A8' (PSAGXMSV).

If first recursion:

- Registers 0-15 saved at LCCA+X'E0' (LCCAXGR2)
- PSW saved at PSA+X'248' (PSAEXPS2).

If second recursion:

- Registers 0-15 saved at LCCA+X'120' (LCCAXGR3)
- PSW remains at PSA+X'18' (FLCEOPSW).

Program check - Initially:

- Registers saved at LCCA+X'08' (LCCAPGR1) for recursive program interruptions.

- Registers saved at LCCA+X'48' (LCCAPGR2) for nonrecursive program interruptions.

- Registers saved at LCCA+X'A0' (LCCAPGR3) for monitor call interrupts that occur while processing a page fault

- PSW saved at PSA+X'400' (PSAPCPSW).

For page faults that require I/O the following occurs:

- Unlocked tasks:

  - Registers moved to TCB
  - PSW moved to RB

- Locked tasks:
  - Registers moved to IHSA
  - PSW moved to IHSA
- SRBs:
  - Are suspended: see "SRB Suspension" later in this chapter.

Locally Locked Task Suspension

Status saving is the same as for locked task interruptions (described earlier under "I/O FLIH") except that IHSA of the locally locked address space also contains the floating point registers, the FRR stacks, and the PSW. The ASCBLOCK field is updated to contain X'7FFFFFFF'. The XSB contains cross memory status, which includes control registers 3 and 4.

SRB Suspension

An SRB can be suspended in four cases. If a service routine encounters a page fault and a page-in is required, then the SRB routine must give up control. In that event, an SSRB (suspended SRB) must be obtained and the status saved in that control block. Then the SSRB is queued from the page control block (PCB) in the real storage manager. When the paging I/O completes, the SSRB is scheduled to the local service priority list (LSPL) where it is found later by the dispatcher. The SSRB must be obtained because the original SRB was not retained after the dispatch. Status saved in an SSRB must include the current FRR stack.

In the second case, a service routine requests a page fix and a page-in is required. This suspension is handled as in case one, except that the SSRB is queued from the page control block root (PCRB).

The third case of SRB suspension is an unconditional request for an unavailable lock. Status saving for SRB suspension for a lock differs from the page fault where the SSRB is queued and where control returns after the redispatch of the SSRB. For a request for the LOCAL lock when it is unavailable, the SSRB is queued from the ASCB. For a request for an unavailable CML lock, the SSRB is queued from the ASCB whose lock is requested. For a request for an unavailable cross memory services lock, the SSRB is queued on that cross memory service lock's suspend queue. (For more detail see the chapter on "Locking" later in this section.) In the cross memory services case of SRB suspension, resumption is at the appropriate entry in the lock manager to try to acquire the lock. Upon release of the cross memory services lock by the holder, any SSRBs are rescheduled. Upón release of the local lock by the holder, and if all suspended elements are for LOCAL lock requests rather than CML lock requests, then the first SSRB that was suspended is given the local lock and rescheduled. The SRB is given control at the next sequential instruction following the lock manager call. If any one of the elements on the local lock suspend queue is suspended as a result of a CML lock request, then all queue elements are dequeued and rescheduled to retry the lock request.

The fourth case of SRB suspension is SMF suspension. See the topic "SMF Suspension" later in this chapter for details of SMF suspension.

Suspend SRB queues can be summarized:

Page Faults

- PCB is chained from PVTCIOQF (at PVT+X'75C') for a common area page and from RSMLIOQ (at RSMHD+X'1C') for a private area page.

- PCB+X'1C' points to SSRB.

Page Fix

- PCB is chained as for page fault.
- PCB+X'09' points to PCBR.
- PCBR+X'18' points to SSRB.

Local Lock Requests

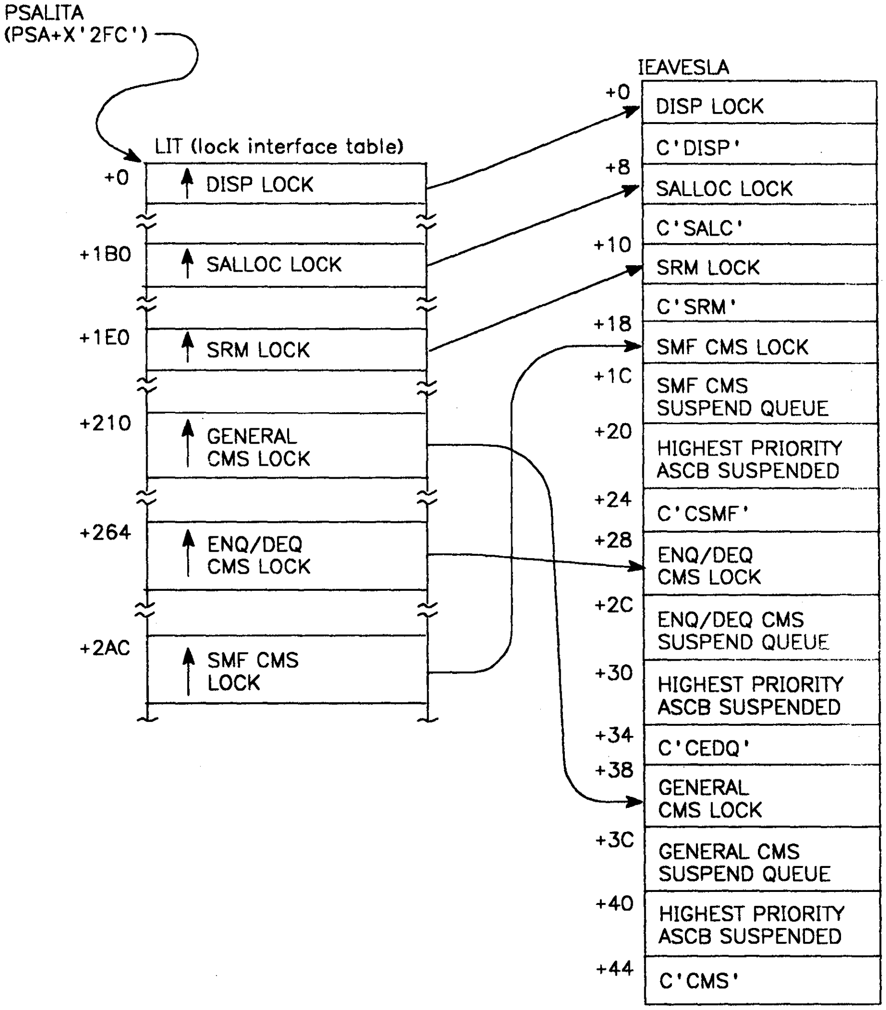
- SSRB is queued from ASCBLSQH(ASCB+X'84').

CML Lock Requests

- SSRB is queued from ASCBLSQH of the ASCB whose lock is requested.

Cross Memory Services Lock Requests

- The SSRB is queued from a cross memory services lock's suspend queue in IEAVESLA as shown:



SMF Suspension

A task or SRB can be suspended if it tries to write a record to SMF (via SVC 83, or branch entry to the SVC routine) and there are no SMF buffers available. Normally, when tasks and SRBs pass records to SMF, SMF places them into buffers located in CSA. When a buffer is full, SMF schedules an SRB to the master scheduler address space which writes the buffers to the SMF data set.

If records are passed to SMF faster than they can be written to the SMF data set, the buffers will fill up. When this happens, the next unit of work which tries to write a record is suspended by SMF until a buffer is available. Other units of work which attempt to write SMF records will be suspended when they attempt to obtain the CMSSMF lock. Note that the master scheduler address space is not suspended by SMF. If the master scheduler address space attempts to write a record when the buffers are full, SMF queues the records in a different chain.

When SMF suspends a unit of work, SMF creates an SSB (SMF suspend block). The SSB contains information needed to reset the unit of work when buffers are available.

Field CVTSMCA (CVT + X'C4') points to the SMCA. Field SMCASSB (SMCA + X'BC') points to the SSB. The field at SSB + X'8C' points to the SSRB created for the suspended task or SRB. The field at SSB + X'90' points to the associated RB if a TCB was suspended. (This field is zero if an SRB was suspended.) The field at SSB + X'98' points to the ASCB for the suspended work.

Normally, work is suspended only for a short time. If the address space holding the CMSSMF lock is suspended (indicated by ASCBLOCK = X'7FFFFFFF', and SMCASSB is not X'00000000'), other address spaces might get backed up on the CMSSMF lock.

Locking

Serialization of resources to provide data integrity and protection is a necessary function of operating systems. In pre-MVS systems, resource serialization was accomplished by physical disablement and by the ENQ/DEQ component. Physical disablement controls only one processor and thus, in MP systems, does not guarantee serialization.

To achieve these requirements the locking facility provides:

- Serialization in a tightly-coupled MP system
- Serialization across address spaces for common resources
- Serialization within address spaces

A lock manager function acquires and maintains all locks. Use of the lock manager is restricted to key 0 programs running in supervisor state, which prevents unauthorized problem programs from interfering with the serialization process. The lock manager is located in the nucleus in CSECT IEAVELK.

Categories of Locks

MVS locks are divided into two categories:

- *Global Locks*, which protect serially reusable resources related to more than one address space. These resources provide system-wide services or use control information in the common area. Examples of resources protected by global locks are UCBs and RSM control blocks.

- *Local Locks*, which protect serially reusable resources assigned to a particular address space. When a task or SRB holds a local lock, the queues and control blocks serialized by that lock can be used only by the task or SRB holding the lock.

Figure 2-1 defines the MVS locks. All MVS locks, except the LOCAL and CML locks, are global locks.

Name	Description
DISP	Global dispatcher lock - serializes functions on a global level.
ASM	Auxiliary storage management lock - serializes the auxiliary storage resources.
SALLOC	Space allocation lock - serializes real storage management (RSM) resources, virtual storage management (VSM) global resources, and some auxiliary storage management (ASM) resources.
IOSYNCH	I/O supervisor synchronization lock - serializes the IOS purge function and other IOS resources.
IOSCAT	IOS channel availability table lock - serializes the IOS processor-related save area.
IOSUCB	IOS unit control block lock - serializes access and updates to the unit control blocks. There is one lock per UCB.
IOSLCH	IOS logical channel queue lock - serializes access and updates to the IOS logical channel queues. There is one lock per channel queue.
SRM	System resources manager lock - serializes use of the SRM control blocks and associated data.
CMSSMF	SMF cross memory services lock - serialize SMF functions and use of SMF control blocks.
CMSEQDQ	ENQ/DEQ cross memory services lock - serializes ENQ/DEQ functions and use of ENQ/DEQ control blocks.
CMS	Cross memory services lock - serializes on more than one address space where this serialization is not provided by one or more of the other global locks. Provides global serialization when enablement is required.
CML	Local storage lock - serializes functions and storage within an address space other than the home address space. There is one cross memory local (CML) lock per address space.
LOCAL	Local storage lock - serializes functions and storage within a local address space. There is one LOCAL lock per address space.

Note: Locks are listed in hierarchical order, with DISP being the highest lock in the hierarchy. An exception is the cross-memory services locks (CMSSMF, CMSEQDQ, and CMS) which are equal to each other in the hierarchy. Also, the LOCAL and CML locks are equal to each other in the hierarchy.

Figure 2-1. Definition and Hierarchy of MVS Locks

Types of Locks

Two types of locks exist. The type determines what happens when a processor makes an unconditional request for a lock that is unavailable. The types are:

- Spin locks - prevent the requesting processor from doing any work until the lock is released by the owning processor. The requesting processor enters a loop in the lock manager (IEAVELK) that keeps testing the lock until the owning processor releases it. As soon as the resource is free, the spinning processor can obtain the resource and continue processing.

- Suspend locks - prevent the requesting unit of work from doing work until the lock is available, but allow the processor to continue doing other work. The request is queued by suspending the requesting task or SRB, and the requesting processor is dispatched to do other work. Upon release of the lock, all of the queued requesters are made dispatchable to retry the lock request, except in the case of the local lock. Upon release of the local lock, the first SSRB will be given the lock and rescheduled; unless there are CML lock requesters on the suspend queue, in which case, all requesters are rescheduled to retry the lock request.

Combining categories and types of locks provide the following:

Global Spin Lock, which is used primarily to provide serialization in MP systems. While code is executing under a global spin lock, it is physically disabled for I/O and external interruptions. An unconditional request for an unavailable lock will cause the processor to spin in the lock manager. Upon release of the global spin lock, the looping processor acquires ownership and returns control to the requestor.

The global spin locks supported by MVS are: DISP, SALLOC, ASM, IOSYNCH, IOSCAT, IOSUCB, IOSLCH, and SRM.

Local Suspend Lock, which is used to serialize resources within an address space. There is one local suspend lock per address space and it is located in the ASCB. An unconditional request for the LOCAL or CML lock when it is not available causes the suspension of the requesting task or SRB until the lock is released.

Global Suspend Lock, which is used to serialize resources that are commonly addressable from any address space. The requestor remains physically enabled while owning the lock. The general cross memory services lock (CMS), the ENQ/DEQ cross memory services lock (CMSEQDQ), and the SMF cross memory services lock (CMSSMF) are the only supported global suspend locks. A local lock must be held in order to obtain a cross memory services lock. An unconditional request for any cross memory services lock when it is unavailable causes suspension of the requesting task or SRB.

Locking Hierarchy

To prevent a deadlock between processors, MVS locks are arranged in a hierarchy, and a processor may unconditionally request only locks higher in the hierarchy than locks that it currently holds. The locking hierarchy is the order in which the locks are listed in Figure 2-1 with DISP being the highest lock in the hierarchy.

Some locks are single system locks (for example, DISP), and some locks are multiple locks in which there is more than one lock within the lock level (for example, IOSUCB). For those global lock levels that have more than one lock, a processor may only hold one lock of each level. For example, if a processor holds an IOSUCB lock, it may not request a different IOSUCB lock.

A unit of work can hold only one local lock at a time. A unit of work cannot hold both its own LOCAL lock and CML lock of another address space. Note that the CML lock of an address space, obtained from another address space, is the same as the LOCAL lock of the address space.

A local lock must be held by the caller when requesting any cross memory services lock. Also, a local lock cannot be released while holding any cross memory services lock.

It is not necessary to obtain all locks in the hierarchy up to the highest lock needed. Only the needed locks have to be obtained, but in hierarchical sequence.

The caller may obtain the three cross memory services locks (CMSSMF, CMSEQDQ, and CMS) only by requesting all of them in a single lock manager request. If a caller holds any one and requests another, an abend will result.

Determining Which Locks Are Held On a Processor

To diagnose certain MVS problems, such as wait states and performance degradation, it is necessary to determine the lock status of the system as well as the back-up of work caused by lock contention.

Locks held by a particular processor are indicated in the processors PSA (prefixed save area). There is a bit map in the PSA which the lock manager checks when a request is made for a lock. This map is called PSACLHS (PSA current locks held string). Each bit corresponds to a particular lock in the hierarchy. The bits are in the same order as the hierarchy so that the low-order bit corresponds to the lowest lock in the lock hierarchy. When a bit is on, it means that lock is held by the current unit of work executing on the corresponding processor. Figure 2-2 shows the bit assignments.

When the local lock bit is on in the PSACLHS, either the LOCAL lock or a CML lock is held. To determine which lock is held by the current unit of work, check the contents of PSALOCAL. If PSALOCAL is zero, then the LOCAL lock of the home address space (pointed to by PSAAOLD) is held. If PSALOCAL is nonzero, the local lock of the address space pointed to by PSALOCAL is held as a CML lock.

(**Note:** When a holder of the local lock or a cross memory services lock is suspended, the corresponding bit in the PSACLHS field is copied to the ASCBHLHI and the PSACLHS is set to 0 even though the lock is still held.)

PSACLHS (location X'2F8' in PSA)

2F8	2F9	2FA	2FB	
00	00	10	00	DISP
00	00	08	00	ASM
00	00	04	00	SALLOC
00	00	02	00	IOSYNCH
00	00	01	00	IOSCAT
00	00	00	80	IOSUCB
00	00	00	40	IOSLCH
00	00	00	20	Reserved
00	00	00	10	Reserved
00	00	00	08	Reserved
00	00	00	04	SRM
00	00	00	02	CMS/CMSEQDQ/ CMSSMF
00	00	00	01	LOCAL/CML

Figure 2-2. Bit Map to Show Locks Held on a Processor

Content of Lockwords

Each lock is represented by a lockword that defines the availability and status of the lock. The contents of lockwords differ according to the category and type of lock they describe:

Global Spin Lockword

- X'00000000' - Lock is available.
- X'0000004n' - Lock is held on processor n.

Global Suspend Lockword (Cross Memory Services Locks)

- X'00000000' - Lock is available.

- X'00xxxxxx' - ASCB address of the locally locked address space. If an address space holds a cross memory services lock but is interrupted or suspended, ASCBHLHI of the locally locked address space will be set and the cross memory services lock-held bit in PSACLHS is turned off until the address space is redispatched. The ASCB address remains in the cross memory services lock until the lock is released.

Local Suspend Lockword (Local Lock)

- X'00000000' - Lock is available.

- X'0000004n' - Lock is held on processor n.

- X'4FFFFFFF' - Task holding a CML lock is now dispatchable or an SSRB holding either the LOCAL or a CML lock is now dispatchable.

- X'7FFFFFFF' - Task or SRB suspended while holding the lock. The reason for suspension is:

  - A page fault.

  - Waiting for a synchronous page fix to complete.

  - An unconditional request for a cross memory services lock while it was unavailable.

  - SMF suspension.

  - SUSPEND=YES was specified on the SDUMP macro.

- X'FFFFFFFF' - Task holding the LOCAL lock was suspended or interrupted but is now dispatchable. The reasons for this state are:

  - A page fault or page fix has been resolved for a locked task.
  - The cross memory services lock, at one time unavailable, is now available.
  - A task holding the LOCAL lock has been preempted.

How To Find Lockwords

Lockwords for single system locks are located in a system lock area called IEAVESLA. PSA+X‘2FC’, PSALITA, points to the lock interface table (LIT); LIT+0 points to IEAVESLA. Lockwords for single system locks can also be located at the label IEAVESLA in a NUCMAP.

Lockwords for multiple system locks are supplied by the requestor of the lock. The addresses of these are placed in the PSA for each processor at locations X‘284’ to X‘298’.

The locations of lockwords are shown in Figure 2-3. Note that all lockwords must reside in fixed common storage.

Lock Name	Category	Type	Number of Locks	Location of Lock	Location of Address of Lock (when actually held)
DISP	Global	Spin	1	IEAVESLA+0	
ASM	Global	Spin	1 per ASID	ASMHD+X‘14’	PSA+X‘284’
SALLOC	Global	Spin	1	IEAVESLA+8	
IOSYNCH	Global	Spin	1	IOCOM+X‘38’	PSA+X‘28C’
IOSCAT	Global	Spin	1	IOCOM+X‘30’	PSA+X‘290’
IOSUCB	Global	Spin	1 per UCB	UCB-8	PSA+X‘294’
IOSLCH	Global	Spin	1 per LCH	LCH+8	PSA+X‘298’
SRM	Global	Spin	1	IEAVESLA+X‘10’	
CMSSMF	Global	Suspend	1	IEAVESLA+X‘18’	
CMSEQDQ	Global	Suspend	1	IEAVESLA+X‘28’	
CMS	Global	Suspend	1	IEAVESLA+X‘38’	
CML	Local	Suspend	1 per address space	ASCB+X‘80’	PSA+‘2EC’
LOCAL	Local	Suspend	1 per address space	ASCB+X‘80’	

*PSA+X‘2FC’ points to the lock interface table; the lock interfacetable+0 points to IEAVESLA.

Figure 2-3. Classification and Location of Locks

Results of Requests For Unavailable Locks

The results of requests for unavailable locks are described in the following topics.

Global Spin Locks

An unconditional request for an unavailable global spin lock results in a disabled loop in the lock manager (IEAVELK). While in the disabled spin loop, the lock manager will periodically enable for EMS or MFA interrupts. The lock manager spins until the global lock is released by the owning processor. In this case, register 11 contains the address of the requested lock and PSALKR14 contains the address of the requestor.

If the lock manager spins for an excessive period of time, then message IEE331A is issued to the operator when the lock manager invokes the excessive spin notification routine, IEEVEXSN. If the operator does not initiate an ACR condition, the lock manager continues to spin until the lock becomes available.

Local Locks

Tasks requesting an unavailable LOCAL lock are suspended. In each case, the request block old PSW (RBOPSW) is set to re-enter the lock manager, and the registers are saved in the TCB. A flag is set in the TCB (TCBLLREQ) to indicate to the dispatcher that the task should not be dispatched until the LOCAL lock is available.

SRBs requesting an unavailable LOCAL lock are suspended. In each case, the lock manager calls the STOP/RESET service (IEAVESRT) to have an SSRB obtained and status saved. The lock manager then queues the SSRB on the LOCAL lock suspend queue.

If the SRB was scheduled with the LOCAL lock option, the LOCAL lock will be obtained for the SRB by the dispatcher. The SRB will get control with the LOCAL lock held. If the LOCAL lock is unavailable, the dispatcher will delay the SRB, that is, the SRB will be queued to the LOCAL lock suspend queue and will not be dispatched until the LOCAL lock is available.

Tasks that request an unavailable CML lock are stopped via a call by the lock manager to the STOP/RESET service (IEAVESRT). The registers, PSW, and cross memory status are saved by IEAVESRT in the TCB, RB, and XSB. The lock manager obtains an SRB, initializes it to run in the requester's address space, and queues it to the local lock suspend queue header (ASCBLSQH) of the address space whose CML lock was requested. This SRB has the SRBCMLRQ bit set on to indicate that a CML request was made for this ASCB's local lock.

SRBs that request an unavailable CML lock are suspended. The lock manager calls the STOP/RESET service (IEAVESRT) to have an SSRB obtained and status saved. The registers, PSW, and cross memory status are saved in this SSRB and its XSB. The lock manager then queues the SSRB to the CML address space's local lock suspend queue header (ASCBLSQH). This SSRB has the SRBCMLRQ bit set on to indicate that this address space's lock was requested as a CML lock.

Notes:

1. *The FRR stack can be used to help recreate the process leading up to the point of suspension by interpreting the recovery routines that are currently active. SSRBs for local lock suspensions can be found by inspecting the local lock suspend queue anchored in the ASCB from field ASCBLSQH (ASCB+X'84'). SSRBs are obtained from SQA (SP 245). SSRBs and delayed SRBs on the local lock suspend queue are chained together at SRB+X'04'.*

2. *When interrogating a given address space, if the ASCBLOCK field is not X'00000000', check the ASCBLSQH to determine the SRB work being delayed in this address space because of lock contention.*

3. *If the ASCBLOCK field is not X'00000000', check the ASCBLOCI field (ASCB+X'E8'). If ASCBLOCI is X'00000000', then the address space's lock is held as a LOCAL lock and not a CML lock.*

4. *If the ASCBLOCK field is not X'00000000' and not a processor ID, then the ASCBCMLH field (ASCB+X'EC') contains the address of the unit of work that was suspended while holding the address space's local lock.*

   For a unit of work that is suspended and holds the address space's local lock as a CML lock, the ASCBLOCI field (ASCB+X'E8') points to the ASCB where the lock owner resides, and the ASCBCMLH field (ASCB+X'EC') points to the owning unit of work. When the high-order bit of ASCBCMLH is on, the unit of work is an SSRB.

   When the local lock is released, the suspend queue is scanned until the first suspended (oldest) element is found or until an element representing a CML lock request is found. If no requesters for a CML lock exist on the suspend queue, the first suspended (oldest) element is dequeued, the ready-to-run ID is placed in the lockword (ASCBLOCK=X'4FFFFFFF'), the SRB/SSRB is given the lock by turning on the SRB local lock held flag (SRBLLHLD), and the SRB is scheduled locally. If a requester for the CML lock exists on the suspend queue, then all suspended elements (SRBs and SSRBs) are dequeued and rescheduled so that the obtain request is retried. The SRBs on the suspend queue that represent the CML lock requester cause the task to be resumed so that the lock request is retried.

Cross Memory Services Locks

Tasks unconditionally requesting a cross memory services lock when it is unavailable are suspended. For each task:

- GPRs are saved in the IHSA which is pointed to from ASXB+X'20' of the locally locked address space.

- The resume PSW in the IHSA is set to re-enter the lock manager.

- The cross memory status is saved in the XSB pointed to by IHSA+X'80'.

- The locally locked ASCB is queued on that cross memory services lock's suspend queue. The suspend queue header for the SMF cross memory services lock follows the lockword in CSECT IEAVESLA at offset X'1C', the suspend queue header for the ENQ/DEQ cross memory services lock follows the lockword at offset X'2C', and the suspend queue header for the general cross memory services lock follows the lockword at offset X'3C'. **Note**: When a NUCMAP is not available, locate the IEAVESLA through PSA+X'2FC' which contains the address of the lock interface table, the lock interface table+X'0' contains the address of IEAVESLA.

The tasks suspended on a cross memory services lock suspend queue are represented by the ASCBs whose local locks they own. For example, if task A in address space A owned the CML lock of address space B and was suspended on the cross memory services lock suspend queue, then the cross memory services suspend queue header would contain the ASCB address of address space B. The ASCBLOCI field of address space B would point to address space A and the ASCBCMLH field of address space B would point to task A. The ASCBs are chained together at field ASCBCMSF (forward pointer).

Note: When an ASCB is on the cross memory services lock suspend queue, the local lock (ASCBLOCK) contains X'7FFFFFFF'.

When the cross memory services lock is released, the ASCBLOCK field of the locally locked address spaces on the suspend queue is changed to one of the following values:

- X'FFFFFFFF' - the LOCAL lock was held by a task that is now ready to run.

- X'4FFFFFFF' - the lock was held by either (1) a task holding the lock as a CML lock and the task is now ready to run, or (2) an SSRB holding a CML or LOCAL lock and the SSRB is now ready to run.

SRBs unconditionally requesting a cross memory services lock when it is unavailable, are suspended. For each SRB, the lock manager calls the STOP/RESET service (IEAVESRT) which:

- Obtains an SSRB from SQA
- Saves GPRs and the FRR stack in the SSRB
- Sets the local lock (ASCBLOCK) to X'7FFFFFFF'
- Saves cross memory status in the XSB of the SSRB.

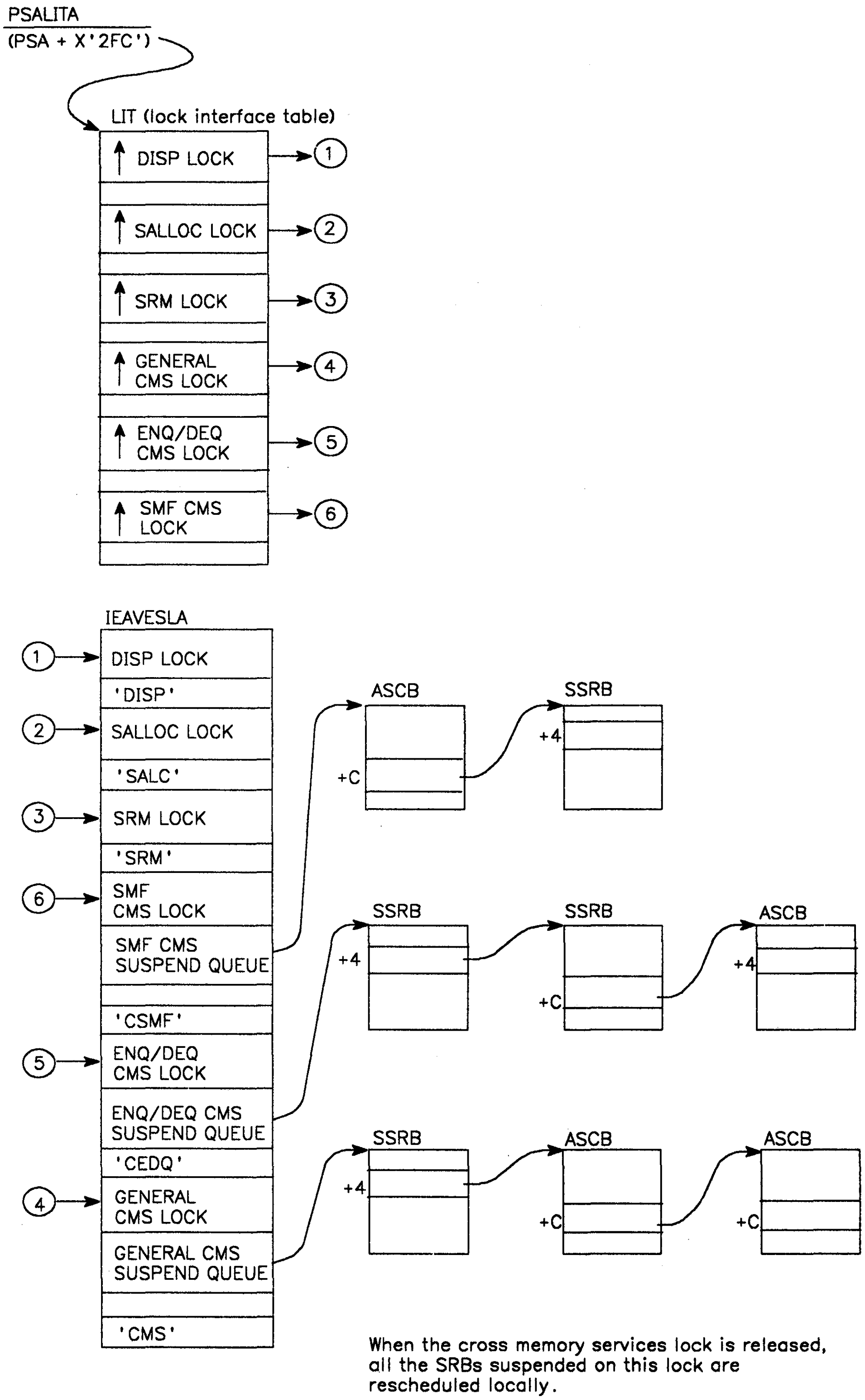
Then the lock manager chains the SSRB on that cross memory services lock-suspend queue located in IEAVESLA.

The offsets for the cross memory services lock suspend queues are:

- CMSSMF - IEAVESLA+X'1C'
- CMSEQDQ - IEAVESLA+X'2C'
- general - IEAVESLA+X'3C'

There is one suspend queue per cross memory services lock and the requestor is chained on the queue associated with the unavailable lock.

The SSRBs and ASCBs are chained on the respective suspend queues using either ASCBCMSF (ASCB+X'C') or SRBFLNK (SSRB+X'4'). There are no backward pointers. Thus the cross memory services lock suspend queues could appear as shown in Figure 2-4.



When the cross memory services lock is released, all the SRBs suspended on this lock are rescheduled locally.

Figure 2-4. Cross Memory Services Lock Suspend Queues

Intersect

Locking serializes resources between routines. The intersect function is used in conjunction with locking to serialize dispatcher control blocks between specific routines and the dispatcher.

There are two levels of intersection:

1. Global intersect, which serializes the ASCB dispatching queue and address space dispatchability flags, first requires that the dispatcher lock be held (the lock serializes between routines, the intersect serializes only with the dispatcher).

2. Local intersect, which serializes the TCB dispatching queue and TCB dispatchability flags, first requires that the local lock of the target address space be held.

Note: The lock associated with each intersect must be obtained before the intersect is requested and the intersect must be reset before releasing the lock to ensure proper serialization.

Determining if Intersects are Held on a Processor

Intersect contention can exist just like locking contention. To check for this condition it is necessary to know where the intersect words are and what they should look like. The global intersect word is in the SVT. PSA+X'B4C' (PSASVT) contains the address of the SVT. If any bits are on in the SVTDSREQ word (SVT+X'1C'), then the global intersect is held. Refer to the SVT mapping in the *Debugging Handbook* to determine who holds the intersect.

A local intersect word exists at X'B4' into each ASCB. Bits in this word are defined in the same way as the global intersect. Refer to the ASCB mapping in the *Debugging Handbook* to determine who holds the local intersect.

The dispatcher has a four-word field of its own which it sets when processing. The dispatcher active field (SVTDACTV) is at offset X'6C' into the SVT. The first byte of the field corresponds to CPU0, the second to CPU1, and so forth. Each byte should have one of the following settings:

- X'00' - Dispatcher not active on the corresponding processor

- Logical CPUID with high order bit off - Dispatcher is executing on this processor

- Logical CPUID with high order bit on - Dispatcher is in recursion mode on this processor.

Requesting the Intersect

A routine requests the intersect via the INTSECT macro. The macro turns on the requestor's intersect bit, then, if the dispatcher active field is not zero, the macro passes control to the intersect service routine (IEAVEINT) which spins, waiting for all dispatcher active bytes to go to zero.

This intersect spin can be detected by examining the registers:

- R15 - address of IEAVEINT
- R1 - address of SVT
- R2 - physical CPU ID of processor on which dispatcher is active
- R3 - contents of dispatcher active byte
- R14 - return address of caller

Ensure that the contents of R3 are valid (that is, that SVTDACTV has not been overlaid). If the caller was disabled, then intersect would set LCCASPN1 with its spin bit (X'02').

Use of Recovery Work Areas For Problem Analysis

Recovery processing enhances the reliability of the MVS operating system. When an error occurs, "active recovery" is given control, one routine at a time, in an attempt to isolate the error to a unit of work. Recovery terminates that work instead of the entire operating system and then continues normal system operation. This process occurs whether the error is in the system or an application.

Because system operation is not halted at the point of error, the resulting storage dumps represent system status sometime after the original error(s). Often the system can encounter numerous errors, fully recover, and continue. At other times it can be a recovery failure that causes the system to cease operations. In either case, the obvious problem and its associated tracks have been covered over. This makes the back-tracking process extremely difficult.

However, experience has shown that although recovery causes this difficulty, it can very often provide valuable clues for the problem analyst. This chapter points out important recovery areas and explains how they can be used in the debugging process.

CAUTION: Recovery is *not* designed to aid the problem solver; it is designed as a means by which the system can prevent total loss. Because recovery maintains system status information, its work areas often provide the same information to the analyst. However, once recovery is invoked, the system is in a tenuous position; it is attempting to maintain operation despite an error. It is possible that the recovery process itself can encounter the same error or bad data. Most often this is not the case; the system does recover and continues normal operation. But the possibility of recursive errors in the recovery process does exist, in which case the new error becomes of prime consideration. If you are dependent on internal recovery control blocks and queues, be aware of this possibility. Don't get caught following a chain of blocks for some subsequent or unrelated problem that will hinder your own error-finding efforts. This danger is most prevalent when you use recovery work areas without following the normal work-related debugging techniques. Do not immediately use the RTM2 work area without analyzing the Task/RB structure and associated indicators.

The following work areas should be used carefully and only after traditional techniques have failed. The exceptions to this rule are:

- When the dump is taken as a result of a trap (for example, SLIP) and the analyst understands that the current status at the time of error can only be found by using the recovery save areas.

- When there are problems in the recovery process itself.

In other instances, be aware of the total environment so that what you discover in these areas bears some relationship to the problem you are analyzing. These areas are of great importance if used with understanding.

SYS1.LOGREC Analysis

For effective problem analysis, use the information in SYS1.LOGREC to understand the error history of the system. Because of recovery processing, MVS does not halt operation when an error occurs. Dump analysis must be performed using a snapshot of storage as it appears sometime after the error and recovery have occurred; therefore, some type of recording mechanism is needed in order to trace the error.

The entries in SYS1.LOGREC provide information about a potential problem. This is the most informative data about the error that you receive. The SYS1.LOGREC entries serve as a diagnostic trace of the problem encountered by the operating system; they usually provide a history of events leading up to a system incident. Use this information to understand system problems, the recovery actions that are taken as a result of these problems, and the outcome of the recovery attempt. The entries in SYS1.LOGREC are described in *SPL: SYS1.LOGREC Error Recording.*

Often more than one record exists for the same software incident. You must be able to relate these records in the proper sequence and understand the progress of recovery the various records indicate. Knowing the errors that have occurred since the last IPL helps you understand the system behavior and explains your findings at dump analysis time.

In stand-alone dump analysis you should always inspect the in-storage LOGREC buffer for entries that recovery routines have made but which were not written to the SYS1.LOGREC data set because of a system problem. Very often it is these records that are the key to the problem solution. (There is a discussion of LOGREC buffer analysis later in this chapter.)

Information that is written by recovery routines to the SYS1.LOGREC data set is used primarily to monitor incidents both when retry is attempted and when percolation to the next recovery routine takes place.

Generally, functional recovery routines (FRRs) will write a SYS1.LOGREC record (via RECORD=YES on the SETRP macro) when they are entered as the first recovery routine for the abend. The default for ESTAE routines, however, is to *not* write a record. This means that unless the ESTAE routine specifically requests recording, no SYS1.LOGREC record will be built.

Listing the SYS1.LOGREC Data Set

To get a listing of the SYS1.LOGREC data set, use EREP as described in *Environment Record Editing and Printing (EREP) User's Guide*. (The JCL required to print the SYS1.LOGREC data set is contained in the chapter "Additional Data Gathering" later in this section. It is important to obtain both an event history and a full report. The event history (EVENT=Y parameter on the EXEC statement) prints an abstract for all records in chronological order. This allows the analyst to recreate the sequence of events.) EREP formats the standard area, the first X'194' bytes of each SDWA, into a series of titles, each followed by pertinent data found in the standard area. EREP will put the variable area, the last X'FF' bytes of each SDWA, after the standard area. This variable recording area (SDWAVRA) is used by the recovery routines to construct messages and to provide data that often contains valuable debugging information.

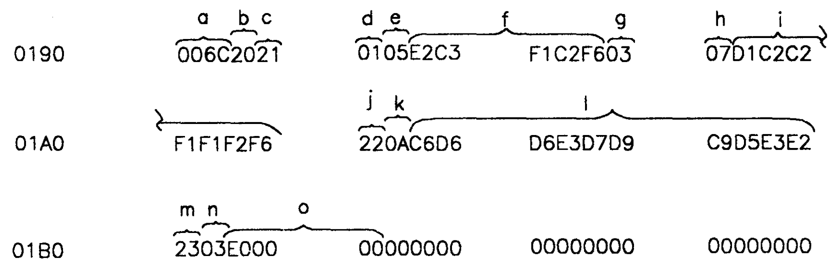
SDWAVRA Key-Length-Data Format

The SDWA variable recording area (SDWAVRA) can optionally be mapped in a key-length-data format. Some MVS recovery routines use this format to provide standardized diagnostic information for software incidents. This formatted information allows you to more easily screen duplicate errors.

Constants for the key field have been defined to describe data such as: component ID, subcomponent name, module ID, assembly date, return and/or reason codes, parameter lists, registers, and control block information. For example, a key of X‘10’ indicates a recovery routine parameter area. The SDWAVRAM bit (in the fixed portion of the SDWA) indicates that the SDWAVRA has been mapped in the key-length-data format as described by the IHAVRA mapping macro. (SDWAVRAM is the third bit in field SDWADPVA at X‘192’.) Refer to the *Debugging Handbook*, for the format of the SDWA, which includes the SDWAVRA, and the format of the VRAMAP, which includes a list of key values.

Some MVS components have assigned specific meanings to key fields for use by their recovery routines. Refer to the module listings of the individual recovery routines that use the key-length-data format for detailed information. “Section 5: Component Analysis” also has details about key fields for some components.

Figure 2-5 shows an example of the SDWAVRA formatted in the key-length-data format. The example starts at offset X‘190’ in a hexadecimal dump of the SDWA.



```
0190     006C2021     0105E2C3     F1C2F603     07D1C2C2

01A0     F1F1F2F6     220AC6D6     D6E3D7D9     C9D5E3E2

01B0     2303E000     00000000     00000000     00000000
```

a - length of variable recording area (108 bytes)
b - X'20' indicates VRA is formatted in key-length-data
c - length of user data in the VRA (33 bytes)
d - key (X'01' - component ID)
e - length (5 bytes)
f - data (SC1B6)
g - key (X'03' - product level)
h - length (7 bytes)
i - data (JBB1126)
j - key (X'22' - header)
k - length (10 bytes)
l - data (FOOTPRINTS)
m - key (X'23' - footprint bytes)
n - length (3 bytes)
o - data (E00000)

SDWAVRA in key-length-data format

Figure 2-5. Example of SDWAVRA in Key-Length-Data Format

Important Considerations About SYS1.LOGREC Records

The LOGREC records are mostly SDWAs the system supplies, plus variable user data areas the individual recovery routines supply.

Following are some special considerations pertaining to specific portions of LOGREC entries:

- Compare the time stamp at the top of the incident records with those in adjacent records. If the system is percolating through FRRs, these times are either identical or just a fraction of a second apart.

- Abend Reason Code - If this field is zero, check *System Codes* to see if a register contains a reason code for the system abend code.

- Jobname - If the jobname is "NONE-FRR," this indicates that the record is generated by an SRB's FRR (Functional Recovery Routine) or the current ASCB was invalid.

- Comp ID Involved - If the component ID is not formatted, you can determine the ID of the failing component by using the name of the module involved in the error and checking the "Module Summary" topic in the *Debugging Handbook.*

- "EC PSW from ESTAE RB (0 for ESTAI)" - This field has the following possible meanings:

  - If the ESTAE is associated with an RB level other than the one encountering the error, this is the PSW at the time that the RB level associated with the ESTAE last gave up control. **Note:** If this is the case, the "RB of ESTAE Not in Control" flag should also be set.

    If the ESTAE is associated with the RB level in error, the PSW is equal to the "EC PSW at Time of ABEND" because the last time the RB level gave up control was when the error occurred.

    If the error occurred locked, disabled, or in SRB mode and is covered by an ESTAE, the two PSWs might not match even for the top RB. "At Abend" is the locked PSW, and "ESTAE RB" is the PSW for the last unlocked interruption.

  - If the record was generated by an FRR, this is the PSW used to pass control to the FRR and is therefore the address of the FRR.

  - If the record was generated by an FRR (that is, a locked/disabled routine is in control, or the system is in SRB mode), and the "EC PSW at Time of ABEND" is equal to the EC PSW from ESTAE RB, this is a system-generated record.

- "Regs of RB Level of ESTAE Exit or Zero for ESTAI":

  - If the ESTAE exit is associated with the RB level that encountered the error, these registers are the same as "Regs at Time of Error."

  - If the ESTAE is associated with an RB level other than the one encountering the error, then these are the registers at the time that RB last gave up control.

  - If this is an FRR-generated record, the two sets of registers are identical. However, if the FRR or ESTAE has updated the registers for retry, these registers are the new, updated registers.

- "SVC by Locked or SRB Routine" - This indicator can be misleading. A forced SVC 13, which is often the way FRR-protected code passes control to recovery, also causes this flag to be set if the SVC occurred in locked, disabled, or SRB mode. Although the flag is set, this situation is not a key error indication in itself. The analyst must investigate why the issuing routine invoked SVC 13.

- Error Identifier - This field, as described in recovery termination management (Section 5), contains pertinent information regarding the error described by this SYS1.LOGREC entry, and provides a correlation to other SYS1.LOGREC entries. Related software and MCH records have the same sequence (SEQ) number that allows the correlation of records written in a particular recovery path (that is, FRR and/or ESTAE percolation, or MCH and subsequent software entries). For locked, disabled, or SRB routines, the processor identifier (CPU) indicates the processor on which the routine was running when it encountered an error. A zero processor identifier indicates that the record was written by an ESTAE routine (that is, the processor identifier is not uniquely identifiable). ASID indicates the current ASID at the time of the error. TIME indicates the time that the ERROR ID was generated. It is normally very close to the time that the record was written, as indicated in the first line of the record. TIME can be used to chronologically order related SYS1.LOGREC entries that contain the same SEQ number. This ordering is useful in reconstructing the environment as it was at the time of the error. Note that TIME changes only during recursion; percolation does not change TIME.

  If an SVC dump is taken, the ERROR ID as it appears in the SYS1.LOGREC record, will also appear in the SVC dump output and associated IEA911I message. Do not be concerned if the ERROR ID sequence numbers seem to have an increment of more than one. Although the RTM adds one to the sequence number of each unique entry (not percolation or recursion), there may be no associated recording of the error, thus, the sequence number is updated internally but is not always externally written. In particular, SDUMP and SNAP might get many expected program checks (which are not recorded) when determining which storage areas can be dumped.

As shown above, the SYS1.LOGREC data set is a vital tool in debugging. At times, the information in the LOGREC printout can be used to describe the entire problem situation. A search of the APAR data base on Retain for the CSECT, recovery routine, and abend code will often identify the problem as a known one.

SYS1.LOGREC Recording Control Buffer

This is one of the most important areas to be used when analyzing problems in MVS. The previous discussion of LOGREC records analysis generally applies to the in-storage LOGREC buffer as well.

This buffer serves as the intermediate storage location for data that the recovery process uses after it has completed but before the data reaches SYS1.LOGREC. The physical I/O is done from this buffer. Its real significance is in the error history it displays. Also, any records in the buffer that have *not* reached SYS1.LOGREC are almost certainly related to the problem you are trying to solve.

Formatting the LOGREC Buffer

The in-storage LOGREC buffer can be formatted by specifying the LOGDATA verb under AMDPRDMP. This verb causes the entries still in the buffer to be formatted in the same manner as those printed from SYS1.LOGREC. For detailed information on how to invoke the AMDPRDMP service aid, see *OS/VS2 SPL: Service Aids.*

Finding the LOGREC Recording Control Buffer

There are two 4K recording buffers in the SQA - one for LOGREC messages, and one for WTO messages.

The CVT+X'23C' (CVTRTMCT) points to the RTCT (recovery termination control table); and RTCT+X'20' (RTCTRCB) points to the RTMRCB (LOGREC recording control buffer). The LOGREC recording control buffer resides in fetch-protected SQA on a page boundary and is 4K bytes in length. On the next page boundary is the 4K buffer that contains WTO messages. By adding X'1000' to the address in RTCT+X'20', you can obtain the address of the WTO message buffer. The WTO message buffer also uses the RCB mapping format.

Format of the LOGREC Recording Control Buffer

The LOGREC recording control buffer is a "wrap-table" similar to the MVS trace table. The entries are variable in size. The latest entries are the most significant especially if they have not yet been written to SYS1.LOGREC. Knowing the areas of the system that have encountered errors and the actions of their associated recovery routines, information obtained from SYS1.LOGREC and the LOGREC recording control buffer, helps provide an overall understanding of the environment you are about to investigate. Figure 2-6 shows the format of the buffer and Figure 2-7 shows the format of individual records within the buffer.

0	4	8	C	E	10
RCBBUFB ↑ start of record area	RCBBUFE ↑ end of record area	RCBFREE ↑ next available space	RCBFLNG number of bytes available	RCBDUM Dummy Displacement	SRB used to post Recording Task in Master Address Space in order to write record to SYS1.LOGREC

X'40'	X'50'
Missing Record Header - This record shows the number of times space was requested but was not available.	Processor serial number

X'58'	X'59'		X'5E'
LCNT Missing record count	FLGS SRB in use flag		RCBTLNG Total buffer length

If the record contains a counter or is present in SYS1.LOGREC, you have a good indication of a recovery loop.

X'60' = first possible record header

Figure 2-6. Format of the LOGREC Recording Control Buffer

Record Header

0	2	3	4	6	8	C	10
Length of Record	Record Types	Options		ASID for POST	ECB	Reserved	Actual Record

Record Type – X'80' – This record wraps around from the end of the buffer space back through the beginning.

X'40' – This record is to go to SYS1.LOGREC.

X'20' – This record is a WTO.

Options – X'08' – Record not buffered; the address of the record exists at X'10'.

X'04' – The recording requestor is to be posted when the record is written.

X'01' – Record is ready to be written. If not set, the record is still being constructed.

Note: The beginning of the actual record + X'20' is the start of the SDWA for software records. The SDWA contains software diagnostic information at the time of the error and is mapped in the Debugging Handbook.

Figure 2-7. Format of Records Within the LOGREC Recording Control Buffer

FRR Stacks

The FRR (functional recovery routines) stacks are often useful for understanding the latest processes on the processors. Entries are added and deleted dynamically as processing occurs. The PSA+X'380' contains the pointer to the current stack. The format is described in Data Areas section of the *Debugging Handbook* under FRRS. Experience has shown that the normal stack (located at X'C00' in each PSA) is perhaps the most useful, although all stacks have been beneficial on occasion.

The FRR stack +X'C' (FRRSCURR) points to the current recovery stack entry. (Unless the FRRSCURR matches FRR stack +0 (FRRSEMP), in which case no recovery is present on the stack.) This entry +0 (FRRSFRRA pointed to by FRRSCURR) points to the recovery routine that is to gain control in case of error. The entry +4 (FRRSFLGS) contains flags used for RTM processing; a X'80' indicates this FRR is currently in control, a X'40' indicates a nested FRR is currently in control. The next 24 bytes (FRRSPARM) serve as a work area for the mainline function associated with the FRR pointed to by this entry. This parameter area may contain footprints useful to your debugging efforts. The previous entry in the stack (X'20' bytes in front of the current) represents the next most current recovery routine. Only the current and previous entries are valid. The stacks do contain residual information associated with recovery that was previously active but is no longer valid. You should not rely on any information beyond the current entry.

Also consider the case where:

A gains control and establishes recovery;
A passes control to B;
B establishes recovery, performs its function, deletes recovery, and passes control to C;
C establishes recovery and subsequently encounters an error.

The FRR stack will contain entries for module A's and C's recovery routines. There is no indication from the FRR stack that B was ever involved in the process although it might have contributed to or even caused the error. The debugger gains an insight into the process but is not presented with the *exact* flow. Although you can get an idea of the general process or flow, do not make assumptions based solely on the FRR stack contents.

If you have trapped a specific problem, the stacks often contain valuable information. The same is true of a stand-alone dump taken because of a suspected loop. If RT1W+0 (RT1TLPN) at FRR stack +X'28' is not zero, the FRR stack contains current, valid data. Following are some of the more valuable fields in the FRR stacks from a debugging viewpoint:

1. FRR stack+X'28' (FRRSRTMW) - RTM 1 work area (RT1W)

   In the case of an error, the RT1W+2 (RT1TENPT) field indicates the error type as follows:

X'01'	- program check
X'02'	- restart key
X'03'	- SVC error (SVC was issued while in locked, disabled, or SRB mode)
X'04'	- DAT error
X'05'	- machine check
X'0A'	- paging I/O error
X'0B'	- abnormal termination
X'0C'	- branch entry to abnormal termination (compatibility interface)

X‘0D’	- cross memory abnormal termination reentry
X‘0E’	- abnormal termination of current TCB
X‘0F’	- memory termination
X‘10’	- cross memory abnormal termination
X‘14’	- MCH (machine check handler)

2. RT1W+X‘34’ (RT1WRTCA) - address of system diagnostic work area (SDWA)

   If no pointers can be found, the global SDWA for the super stacks is located at the respective super stack + X‘410’. For the normal stack, the global SDWA immediately follows the RESTART super stack SDWA at + X‘3F0’. (PSA+X‘3B8’ points to the restart stack.)

3. RT1W+X‘40’ (RT1WMODE) - mode at entry to RTM1

X‘80’	- supervisor control mode (PSASUPER”0)
X‘40’	- physically disabled mode
X‘20’	- global spin lock held
X‘10’	- global suspend lock held
X‘08’	- local lock held
X‘04’	- Type 1 SVC mode
X‘02’	- SRB mode
X‘01’	- unlocked task mode

   This is the system mode at the time of entry to RTM1. The mode may change as processing continues through recovery; the current mode is at RT1W+X‘41’ (FRR stack+X‘69’).

Extended Error Descriptor (EED)

The extended error descriptor (EED) passes error information between RTM1 and RTM2 and also between successive schedules of RTM1. The EED address is found at RT1W+X‘3C’ (RT1WEED), at TCBRTM12 (TCB+X‘104’), or in the RTM2 SVRB at X‘7C’. The EED, pointed to by RTM's SVRB, is generally not valid because RTM2 releases it early in its processing. The EED is described in the *Debugging Handbook*. Important EED fields are:

EED+0 (EEDFWRDP)	pointer to the next EED on the chain, or zero	
EED+4 (EEDID)	description of contents of the rest of the EED	
	BYTE 0	= 1 - software EED
		= 2 - dump parameters
		= 3 - hardware EED
		= 4 - errorid EED

For a software EED:

EED+X‘C’ (EEDREGS)	registers 0-15 at the time of the error
EED+X‘4C’ (EEDPSW)	PSW/Instruction Length Code (ILC)/Translation Exception Address (TEA) at time of error
EED+X‘5C’ (EEDXM)	control registers 3 and 4 at the time of the error

RTM2 Work Area (RTM2WA)

This is the work area used by RTM2 to control abend processing. Registers, PSW, abend code, etc. at the time of the error are recorded in the RTM2WA. This area is often useful for debugging purposes and is described in the *Debugging Handbook* by RTM2WA. This work area can be found through TCB+X‘E0’ (TCBRTWA), or RTM2 SVRB+X‘80’.

Formatted RTM Control Blocks

RTM control blocks are formatted either by AMDPRDMP as a TCB exit with the FORMAT, PRINT CURRENT, and PRINT JOBNAMES control statements, or with the ERR option under SNAP/ABEND. With the exception of the RTCT, the formatted control blocks are all TCB-related, and are formatted only when they are associated with the TCB. The formatted control blocks are:

- RTCT (recovery termination control table) - formatted with the first TCB of the current address space on the processor on which the dump was initiated. (This control block is formatted only by AMDPRDMP.)

- FRRS (functional recovery routine stack) - has the RT1W embedded within it and is formatted with the current TCB if the local lock is held. (This control block is formatted only by AMDPRDMP and it is mutually exclusive of the IHSA).

- IHSA (interrupt handler save area) - has the normal FRR stack saved within it and is formatted with the TCB pointed to by the IHSA, if the address space was interrupted or suspended while the TCB was holding the local lock. (This control block is formatted only by AMDPRDMP and it is mutually exclusive of the FRRS.)

- RTM2WA (RTM2 work area) - formatted if the TCB pointer to it is not zero.

- ESA (extended save area of the SVRB) bit summary - formatted only if the RTM2WA formatted successfully and the related SVRB could be located.

- SDWA (system diagnostic work area) - formats the registers at the time of error only if the ESA formatted successfully and the SDWA could be located.

- EED (extended error descriptor block) - formatted if the TCB or RT1W pointer to it is not zero.

- SCB (STAE control block) - formatted under AMDPRDMP for abend tasks only. It is formatted under SNAP/ABEND whenever the TCB pointer to it is not zero.

System Diagnostic Work Area (SDWA) Use in RTM2

This work area is used to pass information to ESTAE recovery routines. It is found by: SVRB+X'80' points to RTM2WA; RTM2WA+X'D4' points to SDWA. Also, register 1 contains the address of the SDWA when the recovery routines are entered.

Effects of Multiprocessing On Problem Analysis

The multiprocessing (MP) capability of MVS allows multiple processors to share real storage using one control program. (MP refers to multiprocessing on both multiprocessors and attached processors.) MVS also functions on a uniprocessor configuration, which may be only one processor configured out of what is otherwise an MP system. In MP mode, each processor has addressability to all of main storage and executes under the control of one set of supervisor routines.

Because various queue structures must be processed in a serial fashion, interlocking facilities are implemented in both the hardware and software to allow serialization of portions of the control program where conflicts may arise. Queue structures that don't require serialization are processed in parallel, that is, without regard to other processors.

Features of an MP Environment

The main features of a multiprocessing configuration are:

PSA - Each processor has a unique real storage frame, called a prefixed save area (PSA), referenced with addresses from 0 to 4K. Its location in real storage is in the processor's prefix register.

Inter-Processor Communication - Malfunction alerts (MFA) are automatically generated by failing processors before entering the check-stop state. Other inter-processor signaling is accomplished with the SIGP instruction. (This feature is discussed in detail later in this chapter.)

VARY Command - Performs three functions: (1) dynamically add or remove a processor from the configuration; (2) dynamically increase or decrease the amount of useable real storage; (3) control the availability of channels and devices.

QUIESCE Command - Quiesces the system so that I/O pools or two channel switches or both can be reconfigured.

Locking - Access to various supervisory services is serialized by means of a software locking structure.

Dispatching - Assures that highest-priority ready work is processed by available processors.

PTLB (purge translation lookaside buffer) - When an entry is to be invalidated in a page or segment table, the translation lookaside buffer (TLB) on every processor must be purged before permitting subsequent references to the corresponding virtual address.

Timing - The TOD clocks must be synchronized among the configured processors.

RMS - When components of the hardware operating system fail, it becomes the responsibility of the recovery management support (RMS) to help define the extent of the damage.

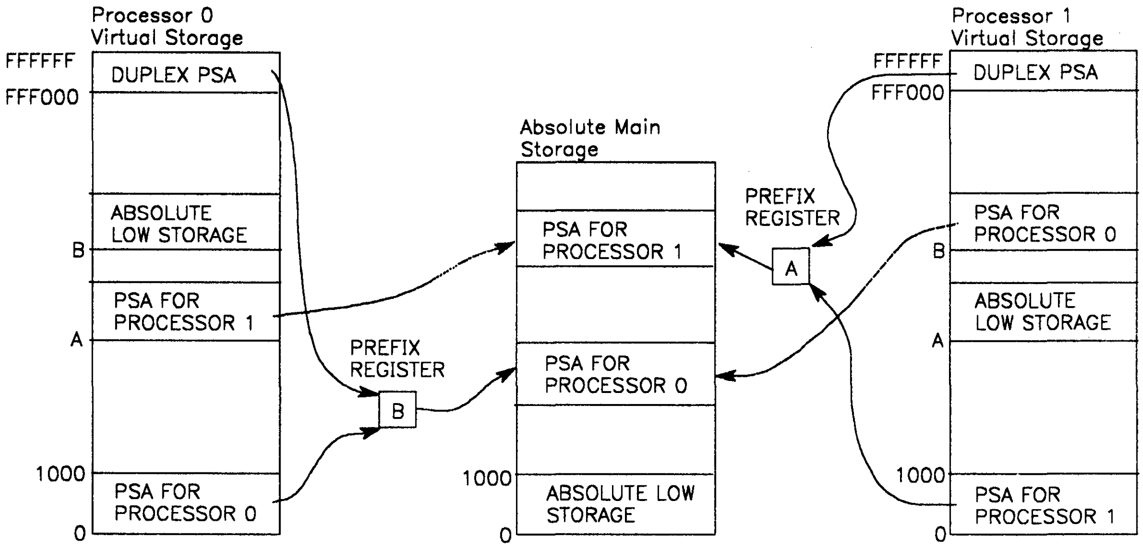
Compare and Swap - Two instructions assure interlocked update operations. They are Compare and Swap (CS) and Compare Double and Swap (CDS). References to storage for these instructions are interlocked the same way as the Test and Set (TS) instruction.

IOS - has the ability to initiate I/O activity to a device from whichever processor has an available path.

ACR - When one processor fails in an MP configuration, the alternate CPU recovery (ACR) function takes the failing processor offline and attempts to release the global system resources held on that processor so that system operation can continue with the remaining processors. (See Miscellaneous Debugging Hints.)

CPU Affinity - The ability to force a job step to execute on a particular processor is a feature of MVS. (For example, because an emulator feature is generally installed on only one of the processors in an MP environment, processor affinity will force the execution of programs that require this feature to the proper processor.)

MP Storage Usage - The following diagram shows storage relationships.



Note that a processor's virtual PSA and the duplex PSA map to the same real address (0). A processor can access the other processor's PSA by using the virtual address of the other processor's PSA. A processor can access absolute low storage by using the virtual address of its own PSA. The duplex PSA is used only by IOS.

MP Dump Analysis

Experience with MVS has shown that there are comparatively few bugs unique to MP. Usually, problems encountered in an MP environment could also be discovered in a UP environment. The increased interaction (parallelism) between software components in an MP environment tends to increase the probability of hitting bugs that are not unique to MP. Thus, the odds are that the dump you are trying to debug could also occur on a UP configuration.

The first step of MP dump analysis is to determine conclusively that it is an MP dump. To do this, you must find the common system data area (CSD). The CSD address is located at offset X'294' in the CVT. The halfword CSDCPUOL, at offset X'A' in the CSD, gives the number of processors currently active. If this number is more than one, you are looking at an MP dump. For the rest of this discussion, we will assume that CSDCPUOL=2.

Several other fields in the CSD are informative. For example, the byte CSDACR at offset X'16', indicates whether or not ACR is in progress. ACR in progress (X'FF' in CSDACR) indicates that one of the processors in the configuration is becoming inactive. If this is the case, the problem may be the result of a failure during ACR processing, and the MP dump will probably present at least two problems:

1. A failure causing ACR to be invoked.

2. A failure during ACR processing. (See the discussion on ACR processing in the "Miscellaneous Debugging Hints" chapter later in this section.)

Data Areas Associated With the MP Environment

There are several processor-related areas with which you should be familiar:

1. The PCCA (physical configuration communication area)
2. The LCCA (logical configuration communication area)
3. The PSA (prefixed save area)

There is a set of these control blocks for each processor located as follows:

CVT+X'2FC' points to the PCCAVT (contains the address of a PCCA for each processor)

CVT+X'300' points to the LCCAVT (contains the address of an LCCA for each processor)

PCCA+X'18' is the virtual address of the PSA for that processor

PCCA+X'1C' is the real address of the PSA for that processor

PSA+X'208' is the virtual address of the PCCA for that processor

PSA+X'20C' is the real address of the PCCA for that processor

PSA+X'210' is the virtual address of the LCCA for that processor

PSA+X'214' is the real address of the LCCA for that processor

The PSA is the "low storage area" (first 4K bytes of storage) and it contains, among other things, the hardware-assigned storage locations. *System/370 Principles of Operation* details the prefixing mechanism the hardware uses to reassign a block of real storage for each processor to a different block in absolute main storage. Prefixing permits processors to share main storage and operate concurrently.

The PCCA contains information about the physical facilities associated with its processor, the LCCA contains save areas for use by the first level interrupt handlers (FLIHs). The need for processor unique areas arises, for example, because external interrupts could occur simultaneously on each processor, and therefore a processor-related area must exist for status saving by the external FLIH. Such areas are in the processor's PSA and LCCA. After locating these control blocks, you can determine several things about the status of each processor.

- The PSWs at the time of the last program, I/O, SVC, external, and machine check interrupts for each processor (PSA)

- The general purpose registers at each program check and machine check interrupt (LCCA)

- The mode (SRB or task) of each processor (LCCA or PSA)

- The address of the device causing the last I/O interrupt on each processor (PSA)

In addition, a work/save area vector table (WSAVTC) pointed to at LCCA+X'218' is associated with each processor. This vector table contains pointers to processor-related work/save areas. For example, there is a large save area for use by ACR, which is pointed to in the processor's WSAVTC. It is important to be aware of the existence of these processor-related areas because GTF, SRM, ACR, IOS, etc., use them; but you must narrow your problem to one of these processes (such as GTF, SRM, etc.) before the information in the associated work/save areas become helpful.

Parallelism

The most important characteristic of the MVS MP capability is parallelism. In looking at MP dumps, you must always remember that several processes might run in parallel and reference the same main storage locations. As a result, queue structures and common data areas are vulnerable. In order to preserve their integrity, the system must insure that they are accessed serially. The resources that must be serialized in order to guarantee their integrity are called serially reusable resources (SRRs). The use of shared resources is the key item to be kept in mind in debugging an MP dump. There are various mechanisms available for serializing SRRs:

- ENQ/DEQ
- WAIT/POST
- Disablement
- Locking
- Compare and Swap (CS) instructions (CS and CDS)
- Nondispatchability
- Test and Set (TS) instruction
- RESERVE/RELEASE
- Intersect

Obviously all users of a particular SRR must use the same serialization mechanism. The integrity of an SRR is reduced if one user uses locking and another uses ENQ/DEQ. You need to understand the processes going on in all processors at the time of the failure. The processor on which the failure occurred might *not* be the one that caused the problem.

Use of the work/save areas pointed to from the ASXB is a good example. These areas are serialized with the local lock. The following diagram shows what could happen if the same address space is running on two processors and one of the processes involved fails to serialize properly.

PROCESSOR 0	**PROCESSOR 1**
●	●
●	●
●	●
●	Gets local lock
Branch enters validity check routine	Branch enters validity check routine
●	
●	Releases local lock
	●
	●

In this example, assume that the process executing on processor 0 fails to get the local lock before it branch enters the system validity check routine. The validity check routine uses the local lock to serialize one of the save areas mentioned above in order to save the caller's registers. The registers saved by the validity check routine on processor 1 can be overlaid by the registers saved by the validity check routine on processor 0. Thus, the failure would be encountered on processor 1, but the processor 0 process would be the one that caused the failure.

OI/NI (OR Immediate and AND Immediate) instructions also illustrate this phenomenon. These instructions take more than one machine cycle to complete (that is, the operand is fetched, altered, and then stored). In previous operating systems, physical disablement and UP environments were enough to insure the completion of one instruction before another was executed. In MVS, with multiple processors, this is no longer true.

For example, suppose processor 0 issues OI and the operand has been fetched. Before processor 0 stores the changed byte, processor 1 executes the fetch cycle of an NI instruction to change a different bit in the same byte. Now, processor 0 stores the original status plus the OI change; subsequently the NI instruction completes, which erases the effect of the OI on the same byte. In MVS, locking is used to solve some of the problems arising from such multi-cycle instructions. When locking is not an appropriate solution, the Compare and Swap instructions could be the appropriate solution. CS serializes the word containing the byte against other processors. CDS serializes a doubleword. The point is that in debugging an MP dump, all processors must be considered because interaction between processes and shared resources is generally the key to solving the problem.

When a program serializes a resource incorrectly, other programs can alter the resource before the first program completes its update. The other programs may be running on other processors, or they may have received control on the same processor because the first program was preempted (for example, SRB suspension because of a page fault) before completing its update. Proving that a problem resulted from incorrect serialization is accomplished by finding both the "other" program and the interval in which a program opens a serialization exposure.

The system trace table can sometimes be used to find potential "other" programs. If the occurrence of the error has not been overlaid in the trace table, it may be possible to reconstruct the series of events leading up to the failure by:

1. Listing all events on that processor, in order, using the logical processor address field in each event's trace entry

2. Making a similar list of all of the events on the other processor(s)

3. Comparing the lists to see if the processes executing in parallel on the processors are altering a common resource

Try to relate these processes that are executing in parallel to the serialization problem that caused the dump.

General Hints For MP Dump Analysis

The following is a list of general hints to help you analyze an MP dump.

1. The use of PRIORITY and DPRTY parameters no longer ensures the order in which tasks are dispatched. First, the SRM, when attempting to handle resources, can allow a task or job with a lower DPRTY to run prior to a job with a higher priority. Second, as the dispatcher dispatches tasks on other processors, tasks of different priority may be executing on multiple processors simultaneously.

2. The CHAP (change priority) SVC does not ensure that tasks are dispatched in the expected order when dispatching on other processors.

3. Attached tasks can execute at the same time as the mother task on different processors. Therefore, if both tasks reference the same data, serialization of the data is required.

4. Any references made to system control blocks that change dynamically after IPL must be serialized to preserve the integrity of the data. The serialization technique for the data item must match that employed by the system.

5. Tasks can be redispatched on a different processor from the one on which they were previously operating. Therefore, do not use the LCCA, PCCA, WSA, or PSA when enabled for interrupts, because redispatch on a different processor results in different data being referenced.

6. If subpools are shared between tasks, users must serialize the use of any data in the subpools common to the two tasks.

7. SRBs can be dispatched on any processor unless they are scheduled with affinity for a particular processor.

8. Asynchronous appendages on one processor can operate simultaneously with the task on another processor.

9. Enabled recovery routines can run on any processor, not necessarily the one on which the error was detected.

10. STATUS STOP SRB request does not prevent SRBs from being added to the local queue; it merely quiesces the address space after any currently executing or suspended SRBs have completed.

11. When access methods allow sharing of data sets between tasks in the same address space, access to the data sets must be serialized between the tasks.

Inter-Processor Communication

MVS uses the inter-processor communication (IPC) function in doing its inter-processor related work. The IPC function uses the SIGP (Signal Processor) instruction to provide the necessary hardware interface between the MP-configured processors. This instruction provides twelve distinct functions. Two of these functions are augmented by the control program to request services of the other processor; external call (XC) and emergency signal (EMS) which are SIGP codes 02 and 03, respectively. Thus, there are two classes of IPC services:

1. *Direct* - These services are defined for those control program functions that require the modification or sensing of the physical state of one of the processors. Ten of the twelve SIGP functions are defined as IPC direct services:

Function	Function Code
sense	01
start	04
stop	05
restart	06
initial program reset	07
program reset	08
stop and store status	09
initial microprogram load	0A
initial processor reset	0B
processor reset	0C

Note: Codes 0A, 0B, and 0C are not valid on a Model 158.

2. *Remote* - These services are defined for those control program functions that require the execution of a software function on one of the processors. The two remaining SIGP functions, external call (XC) and emergency signal (EMS), provide the hardware interface and interruption mechanism to initiate the desired program on the proper processor. The remote service function is provided in two categories:

- Pendable service - use the XC function of SIGP
- Immediate service - use the EMS function of SIGP

When processor A issues a SIGP (XC or EMS) instruction to processor B, a request for an interrupt becomes pending in processor B for the external class. If external interrupts are disabled in the current PSW for processor B, the interrupt is not taken. If the PSW for processor B is enabled, then separate mask bits for XC and EMS are interrogated in control register 0. Interrupts are taken one at a time for those requests enabled in the control register. If processor B is disabled, processor B keeps pending at most one XC and one EMS request. XC requests can pend simultaneously. Each specific XC request is encoded in a physical configuration communication area (PCCA) buffer associated with the receiving processor.

Both the direct and remote services may be used to initiate the desired function on any of the processors physically attached via the MP feature, including the processor the request is initiated on.

Direct Services

The direct service function consists of a macro instruction (DSGNL) and a SIGP issuing routine (IEAVEDR). The DSGNL macro generates an in-line sequence of instructions that:

1. Loads general register 0 with one of the ten SIGP function codes used to perform the desired hardware action

2. Loads general register 1 with the address of the specified processor's physical configuration communication area (PCCA)

3. Loads general register 15 with the address of IEAVEDR

4. BALRs 14, 15

Upon return from IEAVEDR, register 15 contains a return code indicating the status of the request. If the return code is 8, register 0 contains sense information about the receiving processor as shown in Figure 2-8.

Return Code of 8:	**Register 0 Bit**	**Meaning**
	0	Equipment check
	1-23	Reserved
	24	External call pending
	25	Stopped
	26	Operator intervening
	27	Check stop
	28	Not ready
	29	Reserved
	30	Invalid order
	31	Receiver check

The other return codes are:

0 - SIGP instruction successfully initiated. The function is not necessarily completed upon return to the caller.

4 - SIGP function not completed because path to the addressed processor was busy or the addressed processor was in a state where it could not accept and respond to the function code.

12 - Not operational, that is, the specified processor is either not installed or is not configured into the system or is powered off.

16 - SIGP unsuccessful. Processor is a uniprocessor and does not have SIGP sending and receiving capabilities.

Figure 2-8. SIGP Return Codes

Remote Pendable Services

The remote pendable services function (external call) consists of a macro instruction (RPSGNL) and a routine (IEAVERP) which are used to invoke the execution of a specified program on a specific processor. This service is used by supervisor state, zero protection key functions that are *not* dependent upon the completion of the specified service in order to continue their processing. The RPSGNL macro generates an in-line instruction sequence that:

1. Loads register 0 with a code identifying one of the services to be initiated

2. Loads register 1 with the address of the PCCA of the processor on which the service is to be initiated

3. Loads register 15 with the address of IEAVERP

4. BALRs 14, 15

Upon return, register 15 contains a return code. If the return code is 8, register 0 contains sense information (see Figure 2-8). There are currently six functions that can be initiated via external call:

1. **Switch** specifies that the task execution on the other processor is to be preempted.

2. **SIO** - specifies that the IOS start I/O routine (IECIPC) is to be executed on the specified processor.

3. **RQCHECK** - specifies that the timer supervisor TQE check service routine (IEAPRQCK) is to be executed. This routine ensures that the top TQE on the real-time queue is being timed.

4. **GTFCRM** - specifies the GTF service routine (AHLSTCLS) that modifies the Monitor Call (MC) control registers is to be executed.

5. **MODE** - specifies the recovery management services (RMS) service routine (IGFPEXI2) that modifies the RMS oriented control registers is to be executed.

6. **MEMSWT** - specifies that the memory switch service routine (IEAVEMS3) is to be executed, either to force the signaled processor to master's address space, or to initiate work on a waiting processor.

The remote pendable services routine (IEAVERP) sets the appropriate code in the external call buffer of the *receiving* processor's PCCA (offset X'84') as follows:

SWITCH	X'80'
SIO	X'40'
RQCHECK	X'20'
GTFCRM	X'10'
MODE	X'04'
MEMSWT	X'01'

Then IEAVERP sets the external call (XC) function code (X'02') in register 0 and uses the DSGNL service routine instruction to cause the SIGP instruction to be issued. If a busy condition is indicated by the DSGNL service routine, IEAVERP calls the excessive spin notification routine (IEEVEXSN) which issues message IEE331A. The receiving processor will take an external interrupt when it becomes enabled for such interrupts. The external FLIH determines that the interrupt was an XC and passes control to the XC SLIH. The XC SLIH locates the XC buffer (X'84') in his PCCA, determines the function requested, and branches (BAL) to the appropriate routine. Refer to Figure 2-9 for the XC process flow.

Remote Immediate Services

The remote immediate services function consists of a macro instruction, RISGNL, and a routine, IEAVERI, which are used, like the remote pendable services, to cause the execution of a specified program on any of the online MP-configured processors. However, the immediate service differs from the pendable service in two important ways:

- The processors in an MP configuration are enabled for the emergency signal (EMS) interrupt at times when the processors are not enabled for the external call interrupt. In particular, EMS interrupts are enabled when the processor is in the "window spin" state in which all other asynchronous interrupts (except machine check and malfunction alerts) are disabled. This "window spin" state is entered by a routine, such as the lock manager, when a point is reached in its processing that requires an action on the other processor in order for processing to continue. The "window spin" state specifically allows either the malfunction alert or EMS interrupts that are used to trigger the alternate CPU recovery (ACR) function to be accepted and processed.

- An immediate service routine can be requested to execute serially or in parallel with the function requesting the service. That is, IEAVERI will spin while waiting for the designated processor to signal either that the receiving routine has completed execution (serial) or that the receiving routine has been given control (parallel).

Some of the functions that can be initiated via EMS are:

- **HIO** - A Halt I/O command is issued to the designated device by the receiving processor.

- **ACR Function** - The receiving processor helps the sending processor from a failure by alternate CPU recovery procedures.

- **Clock Synchronization** - TOD clocks are adjusted so the same value is in each clock.

- **PTLB** - The receiving processor purges its translation-lookaside buffer (TLB).

The remote immediate services macro, RISGNL, generates an in-line sequence of instructions that:

1. Loads register 0 with the PARALLEL/SERIAL indication

2. Loads register 1 with the address of the PCCA of the processor on which the service is to be executed

3. Loads register 11 with the address of a parameter list to be passed to the service routine

4. Loads register 12 with the entry point address of the service routine to be executed

5. Loads register 15 with the address of IEAVERI

6. BALRs 14, 15

As for direct and remote pendable services, upon return register 15 contains a return code. Register 0 contains sense information in case the return code was eight. (See Figure 2-8.)

IEAVERI builds the emergency signal buffer in the sending processor's own PCCA at offset X'88', sets the EMS function code X'03' in register 0, and uses the DSGNL service routine to cause the SIGP to be issued. The receiving processor will take an external interrupt when it becomes enabled. The external FLIH determines that the interrupt is an EMS and routes control to the EMS SLIH. The SLIH locates the EMS buffer of the sender and, for a parallel request, the SLIH turns off the parallel bit and calls the receiving routine. For a serial request, the receiving routine is given control, and, upon completion, the serial bit is turned off. During this interrupt handling process, the sending processor was in the window spin state until the serial or parallel bit was turned off. Figure 2-10 shows the EMS process flow.

If the receiving routine does not acknowledge the serial request within a certain period of time, the EMS SIGP is reissued. If the spinning processor does not receive acknowledgement of the serial or parallel request after a certain time period, the excessive spin notification routine (IEEVEXSN) is called to issue message IEE331A.

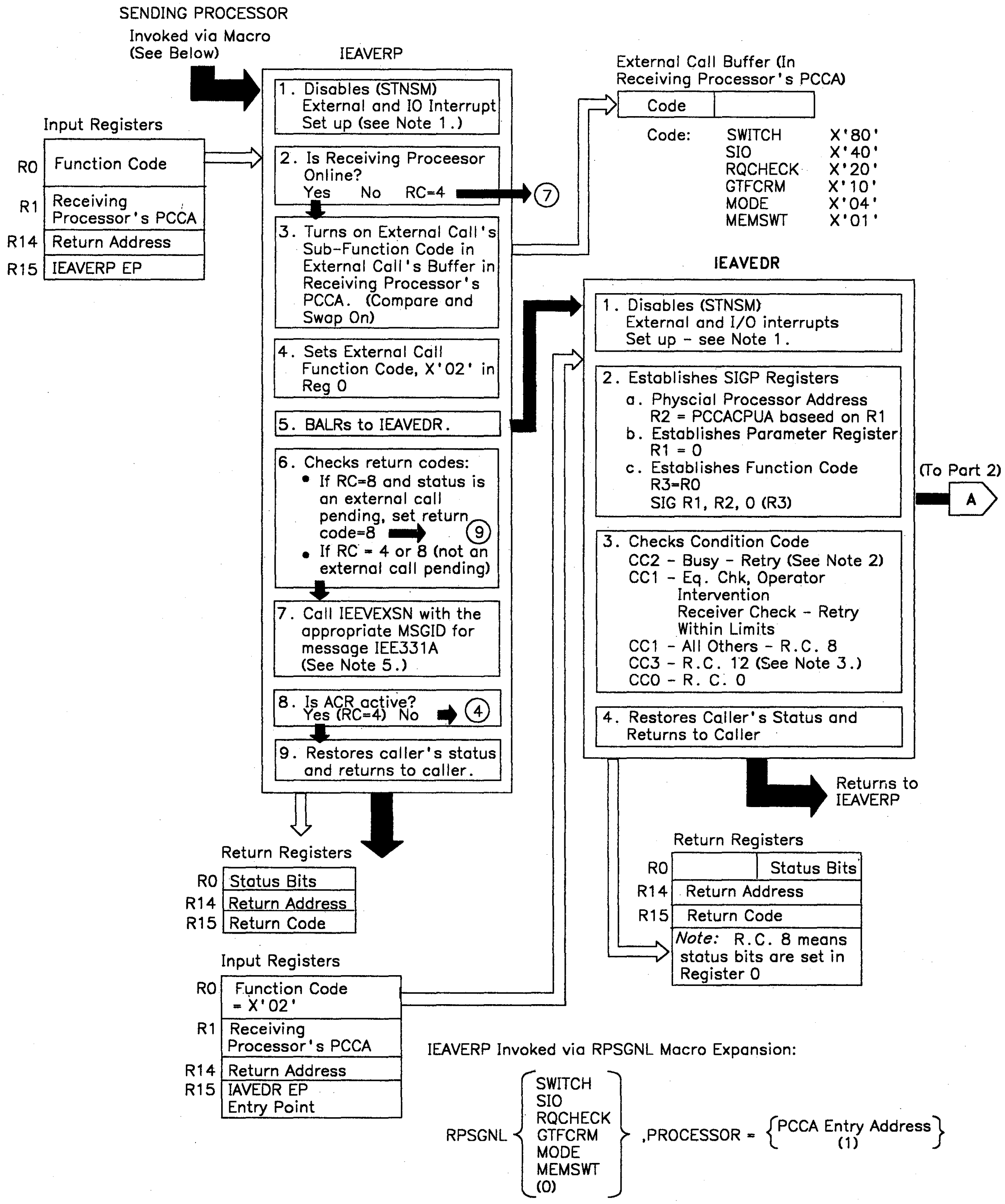
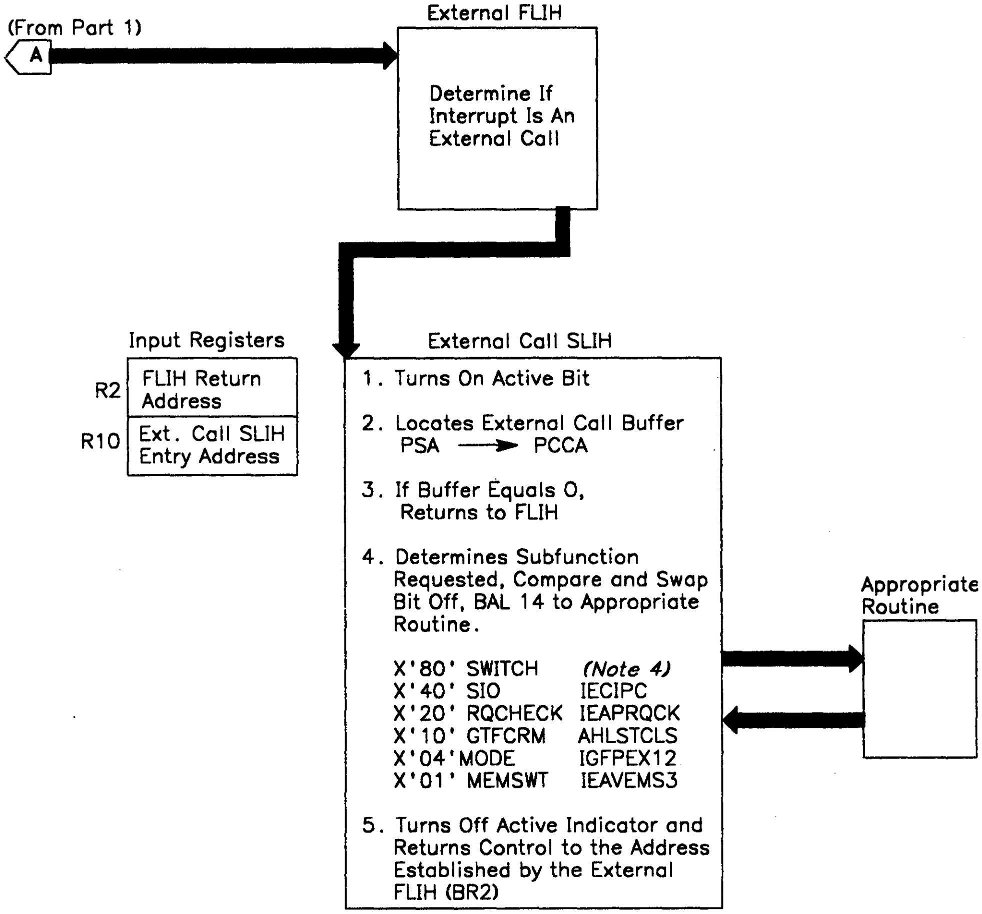

SENDING PROCESSOR
Invoked via Macro
(See Below)
IEAVERP
Input Registers
R0 Function Code
R1 Receiving Processor's PCCA
R14 Return Address
R15 IEAVERP EP
1. Disables (STNSM) External and IO Interrupt Set up (see Note 1.)
2. Is Receiving Proceesor Online?
Yes No RC=4
7
3. Turns on External Call's Sub-Function Code in External Call's Buffer in Receiving Processor's PCCA. (Compare and Swap On)
4. Sets External Call Function Code, X'02' in Reg 0
5. BALRs to IEAVEDR.
6. Checks return codes:
• If RC=8 and status is an external call pending, set return code=8
9
• If RC = 4 or 8 (not an external call pending)
7. Call IEEVEXSN with the appropriate MSGID for message IEE331A (See Note 5.)
8. Is ACR active?
Yes (RC=4) No
4
9. Restores caller's status and returns to caller.
External Call Buffer (In Receiving Processor's PCCA)
Code
Code: SWITCH X'80'
SIO X'40'
RQCHECK X'20'
GTFCRM X'10'
MODE X'04'
MEMSWT X'01'
IEAVEDR
1. Disables (STNSM) External and I/O interrupts Set up - see Note 1.
2. Establishes SIGP Registers
a. Physcial Processor Address R2 = PCCACPUA baseed on R1
b. Establishes Parameter Register R1 = 0
c. Establishes Function Code R3=R0
SIG R1, R2, 0 (R3)
(To Part 2)
A
3. Checks Condition Code
CC2 - Busy - Retry (See Note 2)
CC1 - Eq. Chk, Operator Intervention
Receiver Check - Retry Within Limits
CC1 - All Others - R.C. 8
CC3 - R.C. 12 (See Note 3.)
CC0 - R. C. 0
4. Restores Caller's Status and Returns to Caller
Returns to IEAVERP
Return Registers
R0 Status Bits
R14 Return Address
R15 Return Code
Note: R.C. 8 means status bits are set in Register 0
Return Registers
R0 Status Bits
R14 Return Address
R15 Return Code
Input Registers
R0 Function Code = X'02'
R1 Receiving Processor's PCCA
R14 Return Address
R15 IAVEDR EP Entry Point
IEAVERP Invoked via RPSGNL Macro Expansion:
RPSGNL {SWITCH SIO RQCHECK GTFCRM MODE MEMSWT (0)} ,PROCESSOR = {PCCA Entry Address (1)}

Figure 2-9 (Part 1 of 2). External Call (XC) Process Flow

RECEIVING PROCESSOR



1. Turns On Active Bit
2. Locates External Call Buffer
   PSA ——► PCCA
3. If Buffer Equals 0,
   Returns to FLIH
4. Determines Subfunction Requested, Compare and Swap Bit Off, BAL 14 to Appropriate Routine.

X'80'	SWITCH	*(Note 4)*
X'40'	SIO	IECIPC
X'20'	RQCHECK	IEAPRQCK
X'10'	GTFCRM	AHLSTCLS
X'04'	MODE	IGFPEX12
X'01'	MEMSWT	IEAVEMS3

5. Turns Off Active Indicator and Returns Control to the Address Established by the External FLIH (BR2)

Notes:

1. Turns on active indicator
   Saves callers registers
   Establishes addressability
2. Retry for a period of time because the processor is temporarily busy.
3. If CC = 3 and yet the processor is logically online, a SIGP hardware failure may exist. A "Soft ACR" option is available to the system operator to reconfigure to a UP system.
4. Returns to the dispatcher forcing the interrupted task to be preempted.
5. MSGIDs are determined by the status bits in register 0 if RC = 9 is indicated.

Figure 2-9 (Part 2 of 2). External Call (XC) Process Flow

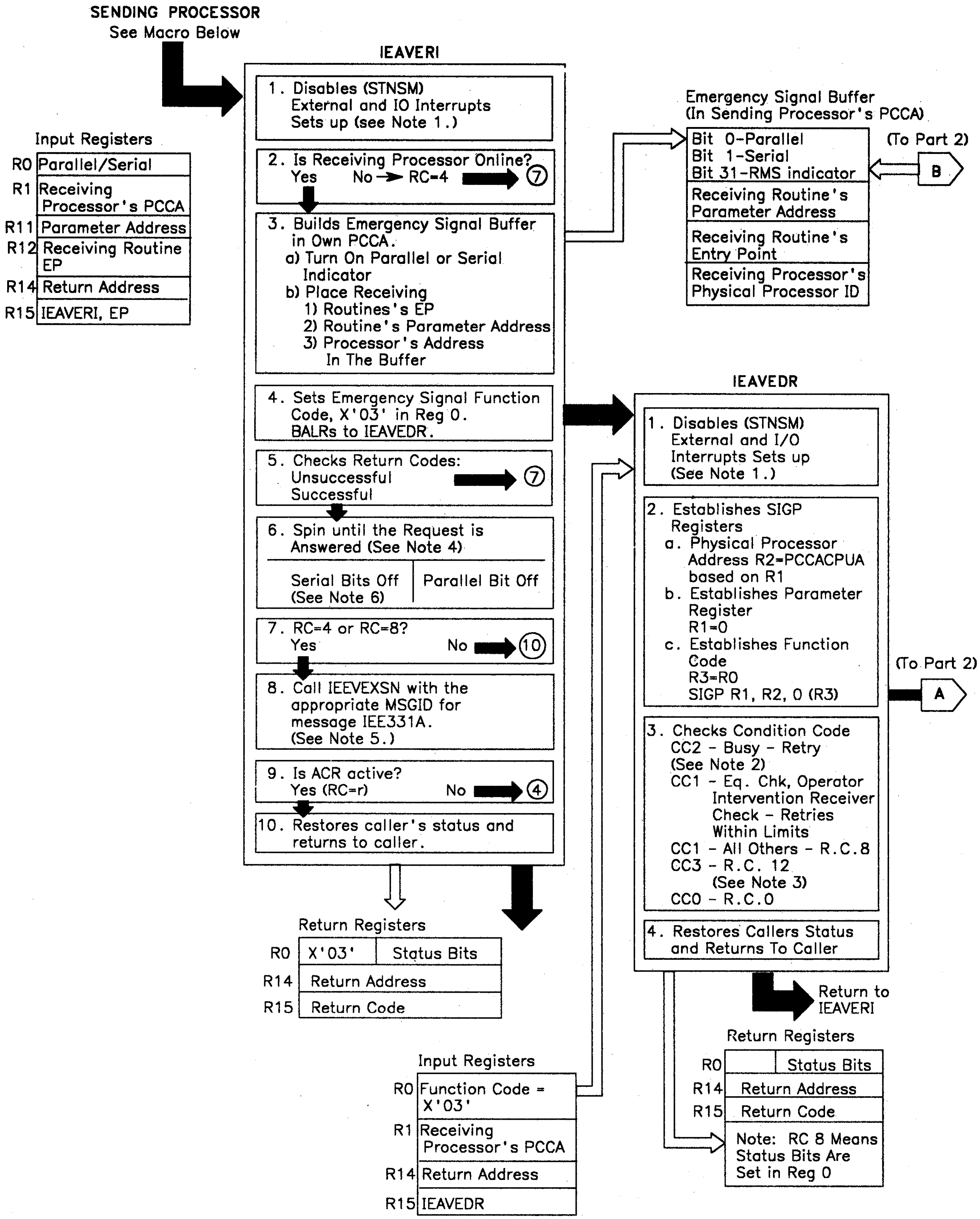
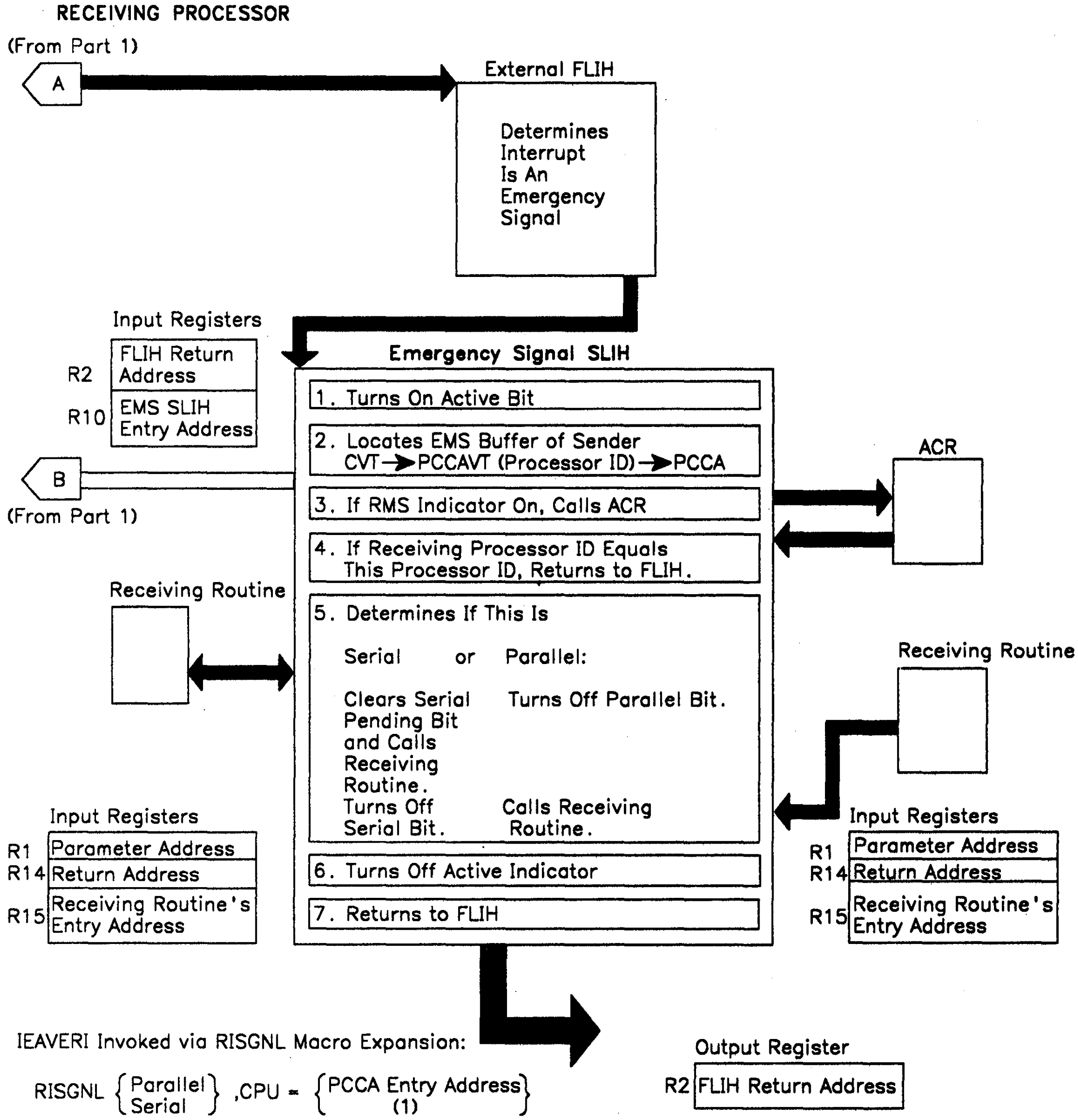

SENDING PROCESSOR
See Macro Below
Input Registers
R0 Parallel/Serial
R1 Receiving Processor's PCCA
R11 Parameter Address
R12 Receiving Routine EP
R14 Return Address
R15 IEAVERI, EP
IEAVERI
1. Disables (STNSM) External and IO Interrupts Sets up (see Note 1.)
2. Is Receiving Processor Online? Yes No → RC-4 → 7
3. Builds Emergency Signal Buffer in Own PCCA.
a) Turn On Parallel or Serial Indicator
b) Place Receiving
1) Routines's EP
2) Routine's Parameter Address
3) Processor's Address In The Buffer
4. Sets Emergency Signal Function Code, X'03' in Reg 0. BALRs to IEAVEDR.
5. Checks Return Codes: Unsuccessful → 7 Successful
6. Spin until the Request is Answered (See Note 4)
Serial Bits Off (See Note 6)
Parallel Bit Off
7. RC=4 or RC=8? Yes No → 10
8. Call IEEVEXSN with the appropriate MSGID for message IEE331A. (See Note 5.)
9. Is ACR active? Yes (RC=r) No → 4
10. Restores caller's status and returns to caller.
Return Registers
R0 X'03' Status Bits
R14 Return Address
R15 Return Code
Input Registers
R0 Function Code = X'03'
R1 Receiving Processor's PCCA
R14 Return Address
R15 IEAVEDR
Emergency Signal Buffer (In Sending Processor's PCCA)
Bit 0-Parallel
Bit 1-Serial
Bit 31-RMS indicator
Receiving Routine's Parameter Address
Receiving Routine's Entry Point
Receiving Processor's Physical Processor ID
(To Part 2)
B
IEAVEDR
1. Disables (STNSM) External and I/O Interrupts Sets up (See Note 1.)
2. Establishes SIGP Registers
a. Physical Processor Address R2=PCCACPUA based on R1
b. Establishes Parameter Register R1=0
c. Establishes Function Code R3=R0
SIGP R1, R2, 0 (R3)
(To Part 2)
A
3. Checks Condition Code
CC2 - Busy - Retry (See Note 2)
CC1 - Eq. Chk, Operator Intervention Receiver Check - Retries Within Limits
CC1 - All Others - R.C.8
CC3 - R.C. 12 (See Note 3)
CC0 - R.C.0
4. Restores Callers Status and Returns To Caller
Return to IEAVERI
Return Registers
R0 Status Bits
R14 Return Address
R15 Return Code
Note: RC 8 Means Status Bits Are Set in Reg 0

Figure 2-10 (Part 1 of 2). Emergency Signal (EMS) Process Flow



IEAVERI Invoked via RISGNL Macro Expansion:

$$\text{RISGNL} \left\{\begin{matrix}\text{Parallel}\\ \text{Serial}\end{matrix}\right\} \text{,CPU} = \left\{\begin{matrix}\text{PCCA Entry Address}\\ \text{(1)}\end{matrix}\right\}$$

$$\text{EP} = \left\{\begin{matrix}\text{Address}\\ \text{(12)}\end{matrix}\right\} \left[\text{,PARM} = \left\{\begin{matrix}\text{Address}\\ \text{(11)}\end{matrix}\right\}\right]$$

Notes:

1. Turns on active indicator
   Saves Callers registers
   Establishes addressability
2. Retry for a period of time because the processor is temporarily busy.
3. If CC=3 and yet the processor is logically online, a SIGP hardware failure may exist. A "Soft ACF" option is available to the system operator to reconfigure to a UP system.
4. Disables/Enables Spin
   1. Turns on SPIN indiator
   2. Enables for MFA and emergency signal interrupts
   3. Disables
   4. Turns off SPIN indicator
5. MSGIDs are determined by the status bits in register 0 if RC=8 is indicated.
6. If the serial request is not acknowledged within a certain time period, reissue the SIGP.

Figure 2-10 (Part 2 of 2). Emergency Signal (EMS) Process Flow

MP Debugging Hints

1. Apparent disabled loop in IEAVERI on processor A

   This is probably caused when processor A sends an EMS to processor B, but the receiving routine on processor B has not yet turned off the serial or parallel bit in processor A's PCCA. Thus, processor A is in the "window spin" state in IEAVERI.

   To find what processor A wanted processor B to do, locate processor A's PCCA using one of the following:

   - Field PSAPCCAV (PSA+X'208') in processor A's PSA contains the virtual address of the PCCA for processor A.

   - Field CVTPCCAT (CVT+X'2FC') points to the PCCAVT. The virtual address of the PCCA for processor A is found by indexing into the PCCAVT using four times the CPUID in PSACPUPA (PSA+X'204').

   PROCESSOR A's PCCA

X'88'	RISP	EMS2	
X'8C'	Receiving Routine PARM address		
X'90'	Receiving Routine EP address		
X'94'	Receiving Processor's PCCA address		

   RISP field

   X'80' - Parallel Request
   X'40' - Serial Request

   EMS2 field

   X'80' - Serial Pending

   By locating the proper PCCA (in this case processor A's), you can determine whether the EMS request was parallel or serial, the entry point, and therefore, the name of the receiving routine. The serial pending bit indicates whether or not the receiving processor has taken the external interrupt. If the serial pending bit is on (PCCAEMS2=X'80'), the interrupt has not been received. Although this information tells quite a bit about the current process on processor A, the real problem, however, is most likely on processor B.

   The following are possible situations that can cause a disabled loop:

   - Processor B, if disabled for EMS interrupts, would never take the EMS interrupt; therefore the receiving routine would never get control and the parallel or serial bit would never get turned off.

   - There could be a hardware problem with the SIGP circuitry. For example, if IEAVERI got condition code 0 as a result of issuing the SIGP instruction on processor A, but the SIGP was never received on processor B, there would be a loop in IEAVERI.

- If the receiving routine loops or hangs for a serial request, IEAVERI will also loop with the serial bit on and the serial pending bit off.

2. Locate External Call buffers

The external call buffer is located at offset X‘84’ in the PCCA. Normally, the buffer is clear, but it is worthwhile to check to make sure that there is no external call work to process, as indicated by the request codes below:

Request Code (PCCA + X‘84’):

X‘80’	- SWITCH
X‘40’	- SIO
X‘20’	- RQCHECK
X‘10’	- GTFCRM
X‘04’	- MODE
X‘01’	- MEMSWT

The code is set in the *receiving* processor′s PCCA so that a bit on in processor B′s PCCA, for example, means that another processor initiated the request. (Note that multiple bits can be on at the same time.)

Note: If the external call or EMS events are available in the trace table, those table entries contain event-related information (for example, the external call buffer). For details see TTE in the *Debugging Handbook.*

3. Determining Which Processor Has I/O Capability

The processor attribute bits, PCCAATTR, are located at offset X‘178’ in the PCCA. If bit 1 (PCCAIO) is 1, then this processor has I/O capability, which means that this processor has at least one channel logically online.

Bit 1 is set to 0 by:

IEAVNIP0: For each processor that has no channels physically online. (**Note**: For Model 158 and Model 168 AP systems, PCCAIO = 0 for the attached processing unit.)

IEEVCPRL: When the last channel of a processor is varied offline.

Bit 1 is set to 1 by:

IEAVNIP0: For each processor that has channels physically online.

IEEVWKUP: When a processor is varied online and it has channels physically online, or when the first channel of a processor is varied online.

Bit 1 is referenced by:

IGFPTERM: When searching for a live processor, if that processor has I/O capability (PCCAIO = 1), a SIGP EMS is issued to that processor.

IGFPTSIG: When processing an EMS received from a failing processor. When invoked during system termination, if executing on a processor with I/O capability, IGFPTSIG writes to LOGREC and the console.

IGFPXMFA: When processing an MFA received from a failing processor. If executing on a processor that has I/O capability, IGFPXMFA invokes ACR.

IEAVTACR: If PCCAIO = 1 for the failing processor, IEAVTACR invokes I/O restart to handle outstanding I/O.

MVS Trace Analysis

This chapter reviews the MVS trace, GTF trace, and master trace functions. The MVS trace (similar to the OS trace) and the GTF trace are available in both system-initiated dumps (SNAP) and in stand-alone dumps. There are formatting routines for most combinations. The trace table entry format can be found in the "Data Area" section (see TTE-Trace Entry) and the "Dump and Trace Formats" section of the *Debugging Handbook*.

The information in this chapter is provided to assist you in reviewing the various formats as you will see them in a storage dump. The page fault path is used as the vehicle for describing the MVS trace and GTF trace formats in the following examples and descriptions.

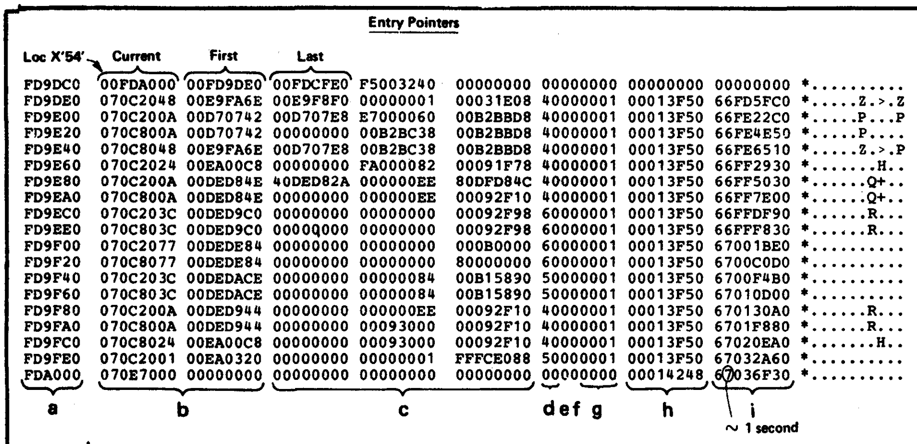
The master trace function and the message processing facility table (MPFT) are described later in this chapter.

Trace Entries

To have these entries formatted in a SYSUDUMP/SYSABEND/SYSMDUMP, the installation must specify SDATA=(TRT) in the SYS1.PARMLIB members or use the CHNGDMP command.

Note: SYSMDUMP produces a machine-readable dump; AMDPRDMP must be used to print it. AMDPRDMP does not format the system trace table.

For unformatted trace table entries, the system queue area (SQA) must have been printed. Use location X'54' as shown in Figure 2-11 to locate the trace table. Remember that 'TRACE ON' was required at IPL time. (Note that if GTF is active, the system trace is turned off.)



Entry Pointers
Loc X'54'
Current
First
Last
a
b
c
d e f g
h
i
~ 1 second

```
FD9DC0  00FDA000 00FD9DE0 00FDCFE0 F5003240  00000000 00000000 00000000 00000000 *..........
FD9DE0  070C2048 00E9FA6E 00E9F8F0 00000001  00031E08 40000001 00013F50 66FD5FC0 *.....Z.>.Z
FD9E00  070C200A 00D70742 00D707E8 E7000060  00B2BBD8 40000001 00013F50 66FE22C0 *.....P...P
FD9E20  070C800A 00D70742 00000000 00B2BC38  00B2BBD8 40000001 00013F50 66FE4E50 *.....P....
FD9E40  070C8048 00E9FA6E 00D707E8 00B2BC38  00B2BBD8 40000001 00013F50 66FE6510 *.....Z.>.P
FD9E60  070C2024 00EA00C8 00000000 FA000082  00091F78 40000001 00013F50 66FF2930 *.......H..
FD9E80  070C200A 00DED84E 40DED82A 000000EE  80DFD84C 40000001 00013F50 66FF5030 *......Q+..
FD9EA0  070C800A 00DED84E 00000000 000000EE  00092F10 40000001 00013F50 66FF7E00 *......Q+..
FD9EC0  070C203C 00DED9C0 00000000 00000000  00092F98 60000001 00013F50 66FFDF90 *......R...
FD9EE0  070C803C 00DED9C0 0000Q000 00000000  00092F98 60000001 00013F50 66FFF830 *......R...
FD9F00  070C2077 00DEDE84 00000000 00000000  000B0000 60000001 00013F50 67001BE0 *..........
FD9F20  070C8077 00DEDE84 00000000 00000000  80000000 60000001 00013F50 6700C0D0 *..........
FD9F40  070C203C 00DEDACE 00000000 00000084  00B15890 50000001 00013F50 6700F4B0 *..........
FD9F60  070C803C 00DEDACE 00000000 00000084  00B15890 50000001 00013F50 67010D00 *..........
FD9F80  070C200A 00DED944 00000000 000000EE  00092F10 40000001 00013F50 670130A0 *......R...
FD9FA0  070C800A 00DED944 00000000 00093000  00092F10 40000001 00013F50 6701F880 *......R...
FD9FC0  070C8024 00EA00C8 00000000 00093000  00092F10 40000001 00013F50 67020EA0 *.......H..
FD9FE0  070C2001 00EA0320 00000000 00000001  FFFCE088 50000001 00013F50 67032A60 *..........
FDA000  070E7000 00000000 00000000 00000000  00000000 00000000 00014248 67036F30 *..........
```

where:

a — address column in SQA

b — PSW or device address/CAW if an SIO operation

c — variable, see TTE in *Debugging Handbook*

d — ILC/CC/PM from the PSW

e — Channel set ID for an SIO or I/O interrupt entry. (This field is zero for I/O interrupt entries when channel set switching is not installed.)

f — CPU ID: 0 for processor 0; 1 for processor 1

g — ASID: 0001 is master scheduler; 0002 is usually JES;
0000 is dummy task or N/A

h — TCB address

i — Timer value

Figure 2-11. How to Locate the Trace Table

If low address storage is overlaid and the trace table pointer (X'54') is lost, you can locate the trace table (which is in the SQA) by searching through the high address range of common storage. Each trace entry is X'20' bytes in length and begins in the extreme left-hand column of a storage dump. Once you locate a pattern of X'07' and X'04' combinations, you have found the trace table.

If location X'54' has not been overlaid, then it will point to the control information for the trace; this information is directly in front of the actual table.

The trace routine places an entry (record) type indicator in the fifth position of the PSW and moves the interrupt code in to make the PSW appear as BC mode. Figure 2-12 explains each of the trace entry types.

```
     Position 5
        ↓
①— 000001D1 00009FD8 00000000 00000000  00FDABD8 00010001 00000000 AB57A140  *   J   Q           Q            *
   078D7000 0009C1A2 00000000 00000001  0017C2BE 60010011 007C40D8 AB57AA30  *      AS          B        Q    *
③— 078D2012 00096CE0 00095288 000955E8  00096F80 40010011 007C40D8 AB58A680  *           H   Y           Q  W *
   070C203C 0001EBD8 00095288 00000000  007FD69C 60010011 007C40D8 AB58AA40  *       Q   H      O        Q    *
⑨— 070C803C 0001EBD8 00000000 00000000  007FD69C 60010011 007C40D8 AB58B190  *       Q          O        Q    *
   078C2078 0001EC94 0000E500 00000178  00000000 40010011 007C40D8 AB58B3D0  *       M  V                Q    *
④┐ 078C8078 0001EC94 00000000 00000178  007A2E88 40010011 007C40D8 AB58C350  *       M           H       Q  C *
 └ 078C3011 0001ECB8 00000000 007A2FF0  007A2E8D C0010011 007C40D8 AB58C610  *                0          Q  F *
⑦— 070C6000 0004AFF0 00000011 00FEC760  00FEC78C 00010011 007C40D8 AB5D8130  *       Q      G   G        Q  A *
②┌ 078C1004 0001EE18 00000000 31000163  40000005 00010000 000115A8 AB58D800  *                           Y  Q *
   078C7078 0001EE18 00000000 31000163  40000005 40010011 007C40D8 AB5A1BE0  *                           Q    *
⑥— 078C51D1 0001EE18 0007D740 0C000001  00000000 00010011 007C40D8 AB5A1E00  *   J      P                Q    *
   060C5582 00018424 8000B1A8 0C000001  00000000 30010011 007C40D8 AB5A3D00  *   B  D    Y               Q    *
   060C5950 00018424 0007E6B8 0C000001  00000000 30010011 007C40D8 AB5A4300  *      D   W                Q    *
⑤— 070C4000 0004AF98 00000012 00FA3F40  00FA3F60 00010012 007CDEB8 AB5A5A30  *       Q                        *
⑧┌ 078D700A 00F5136A 00000003 0000002A  00105F71 4001003B 007FA930 AB5A6220  *     5                    Z     *
   078D3011 00F5136A 00000003 00107000  00106F71 8001003B 007FA930 AB5AC540  *     5                    Z   E *
   070C7003 0001F360 0001F360 00000001  00FA3F40 40010012 007FE080 AB5AFB70  *      3   3                     *
```

where:

① – Fifth digit = 0. This is an SIO entry. The CAW address is '9FD8'. The IOSB address is X'FDABD8.' The channel set id is 0.

② – Fifth digit = 1. This is an external type. The interrupt code is X'1004' generated by a clock comparator interrupt.

③ – Fifth digit = 2. This is an SVC interrupt. An SVC '12' was issued from location X'96CE0' (minus the ILC). Variable fields are registers 15, 0, and 1.

④ – Fifth digit = 3. This is a program interrupt. Interrupt code X'11' is a page exception. Word 4 is the referenced translation exception address (TEA).

⑤ – Fifth digit = 4. This is an SRB dispatch. The address in the PSW (X'4AF98') is the entry point address. Offset X'16' contains the ASID to be dispatched. Word 3 is the purge ASID and word 7 the purge TCB.

⑥ – Fifth digit = 5. This is an I/O interrupt. The device address has been moved into the PSW. Words 3 and 4 are the CSW with channel end/device end.

⑦ – Fifth digit = 6. This is an SRB redispatch. SRBs can be suspended because of lock contention or a page fault. The address in the PSW is the lock manager return address or the instruction that caused the page fault.

⑧ – Fifth digit = 7. This is a task dispatch. The interrupt code is from the last task interrupt. If the interrupt code is 0, it could be the first dispatch of this request block (RB) for the task.

⑨ – Fifth digit = 8. This is an SVC return. The interrupt code is from the last SVC interrupt for the RB.

Note: Additional trace entry types are:

- Fifth digit = 9. This is a Program Call (PC) instruction. Word 3 contains the new PASID (offset X'8') and the new SASID (offset X'A').
- Fifth digit = A. This is a Program Transfer (PT) instruction. Word 3 contains the new PASID (offset X'8'). Word 4 contains the old PASID (offset X'C').
- Fifth digit = B. This is a set secondary ASID (SSAR) instruction. Word 3 contains the new SASID (offset X'A'). Word 4 contains the old SASID (offset X'E').

Figure 2-12. Types of Trace Entries

Note: In previous systems, the program check trace entries had registers 15, 0, 1 in words 3, 4, and 5. Also, the fourth word was the TEA for page fault entries. This is changed in MVS; the fourth word for any type of program check is now the TEA.

Trace Entry for Service Processor Call SVC

The trace entry for the Service Processor Call SVC interruption is represented by a type 2 entry (SVC interruption), with a 122 (X'7A') SVC number, and an ESR code of 6 in register 15.

The trace entry for the MSSFCALL DIAGNOSE or SERVICE CALL instruction external (service signal) interruption (interruption code X'2401') is represented by X'1401' in the trace table.

Both entries contain the following in the register 0 and 1 fields:

- Register 0 field - contains the service processor command word (four bytes).
- Register 1 field - contains the two-byte response that the service processor puts in bytes 6 and 7 of the service processor data block; the one-byte caller flags that the caller put in byte 2 of the data block; and one byte of zeros (reserved).

This information helps you trace Service Processor Call SVC processing. For additional information, refer to the topic "Service Processor Call SVC and MSSFCALL DIAGNOSE Instruction" for processors with an MSSF, or the topic "Service Processor Call SVC and SERVICE CALL Instruction" for processors with the Service Processor Architecture.

Trace Examples

Figure 2-13 through Figure 2-16 illustrate different kinds of MVS and GTF traces, as follows:

Figure 2-13. MVS Trace of a Page Fault Without I/O
Figure 2-14. MVS Trace of a Page Fault With I/O
Figure 2-15. GTF Trace of a Page Fault Without I/O
Figure 2-16. GTF Trace of a Page Fault With I/O

While trace tables do not include all system activity, they can be very helpful in establishing a pattern. Remember that many MVS system services are branch entered and therefore do not appear in any trace type entry.

Figure 2-13 illustrates a page fault that did not require I/O for completion. Note that field IDS contains the information described in notes d, e, f, and g in Figure 2-11.

```
              ⓐ
PGM  OLD PSW  └─071C3011 00E44072   R15/R0 00000000 009F2F50   R1   009F2F50   IDS 90000003   TCB 00A0C318   TME F09679C0
SVC  OLD PSW    070C2013 00BA6B98   R15/R0 00000000 00000198   R1   009F4C78   IDS 60000003   TCB 00A0C318   TME F096D6C0
PGM  OLD PSW    075C3011 00DDA3F6   R15/R0 00000000 009F1F20   R1   000000E0   IDS 70000003   TCB 00A0C318   TME F09728A0
SVC  OLD PSW    075C203C 00DDA972   R15/R0 009F1F20 00000000   R1   809F1FD8   IDS 50000003   TCB 00A0C318   TME F097EE70
RET  NEW PSW    075C803C 00DDA972   R15/R0 00000000 00000000   R1   809F1FD8   IDS 50000003   TCB 00A0C318   TME F0980A70
```

ⓐ – Fifth digit = 3 and the interrupt code is X'11'.. The faulting instruction is at X'E44072' and is referencing X'9F2F50'. Because the next entry for this ASID and TCB is *not* a redispatch of the same location, it can be assumed that the page exception was satisfied by reclamation or the first time reference after a GETMAIN. No I/O was required.

Figure 2-13. MVS Trace of a Page Fault Without I/O (Formatted by SNAP in SYSUDUMP/SYSABEND)

Figure 2-14 illustrates another possible format of a page fault.

```
①──── 078C3011 00F685D8 00F68008 007CD000  007CA000 40010015 007C7958 AB5FB280  *    6EQ 6                    *
  ┌── 000001D2 00009FF0 0007D740 0C000001  00FDABD8 00010001 00000000 AB5FE470  *  K  O  P      Q          U *
②┘    078D7000 0009D3F0 003F52F0 FF17E980  0011E1B4 5E010010 007CFCF0 AB5FEDA0  *    LO   O  Z          O    *
      078D5951 0009D3F0 103BC8F8 0C000000  00000000 1E010010 007CFCF0 AB5FF000  *    LO  H8              O  O *
      070C4000 0004AF98 00000002 00FEC178  00FEC1A4 00010002 007FC588 AB5FF870  *    Q       A   AU    EH  8 *

      078D7000 0009D3F0 003F52F0 FF17E980  0011E184 5E010010 007CFCF0 AB613350  *    LO   O  Z          O    *
③──── 078D51D2 00118F00 0007D740 0C000001  00000000 0E010010 007CFCF0 AB614600  *  K     P               O    *
  ┌── 078C7000 00F685D8 00F68008 007DE7C0  007CA000 40010015 007C7958 AB616840  *    6EQ 6   X                *
④┘
```

where:

① – The page exception.

② – The SIO by IOS after a branch entry from ASM.

③ – The I/O interrupt with channel end/device end.

④ – Redispatch at page faulting location.

Figure 2-14. MVS Trace of a Page Fault With I/O (Unformatted)

Note that the sequence illustrated for the page fault path is not a mandatory one. Frequently ASM finds more than one request for paging on the queue and can satisfy them with one I/O. Also, if RSM queues a request and notes that a request already exists, it does not interface with ASM. The ASM SRB has been scheduled previously.

The next two examples are of GTF traces with the following options in effect:

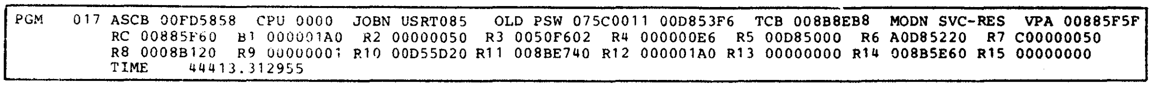
FORMAT=SYS
SVC=ALL
SIO=ALL
PI=ALL
IO=ALL
EXT=YES
RR=YES

USR=YES
GTF=NO
DSP=YES
PCI=YES
RNIO=NO
SRM=YES
USERTIME=YES

Note: The fields in GTF trace records are described in *Debugging Handbook*, Volume 1.

Figure 2-15 illustrates one of two situations:

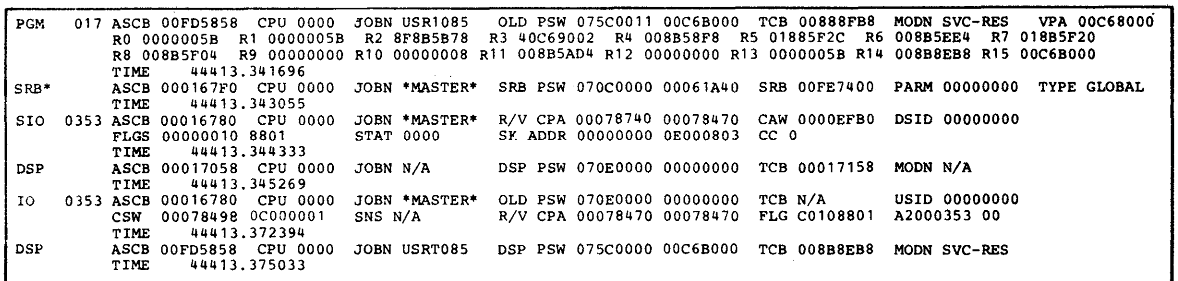
1. A first reference to a page after a GETMAIN was issued for it.
2. A reclaim; that is, a fault on a page which was stolen but whose real frame had not yet been reused.



```
PGM    017 ASCB 00FD5858  CPU 0000  JOBN USRT085   OLD PSW 075C0011 00D853F6  TCB 008B8EB8  MODN SVC-RES  VPA 00885F5F
           RC 00885F60  B1 000001A0  R2 00000050  R3 0050F602  R4 000000E6  R5 00D85000  R6 A0D85220  R7 C00000050
           R8 0008B120  R9 00000001 R10 00D55D20 R11 008BE740 R12 000001A0 R13 00000000 R14 008B5E60 R15 00000000
           TIME    44413.312955
```

Figure 2-15. GTF Trace of a Page Fault Without I/O

Figure 2-16 shows the steps taken to acquire a new page following a page fault.



```
PGM    017 ASCB 00FD5858  CPU 0000  JOBN USR1085   OLD PSW 075C0011 00C6B000  TCB 00888FB8  MODN SVC-RES   VPA 00C68000
           R0 0000005B  R1 0000005B  R2 8F8B5B78  R3 40C69002  R4 008B58F8  R5 01885F2C  R6 008B5EE4  R7 018B5F20
           R8 008B5F04  R9 00000000 R10 00000008 R11 008B5AD4 R12 00000000 R13 0000005B R14 008B8EB8 R15 00C6B000
           TIME    44413.341696
SRB*       ASCB 000167F0  CPU 0000  JOBN *MASTER*  SRB PSW 070C0000 00061A40  SRB 00FE7400  PARM 00000000  TYPE GLOBAL
           TIME    44413.343055
SIO   0353 ASCB 00016780  CPU 0000  JOBN *MASTER*  R/V CPA 00078740 00078470  CAW 0000EFB0  DSID 00000000
           FLGS 00000010 8801       STAT 0000      SK ADDR 00000000 0E000803  CC 0
           TIME    44413.344333
DSP        ASCB 00017058  CPU 0000  JOBN N/A       DSP PSW 070E0000 00000000  TCB 00017158  MODN N/A
           TIME    44413.345269
IO    0353 ASCB 00016780  CPU 0000  JOBN *MASTER*  OLD PSW 070E0000 00000000  TCB N/A       USID 00000000
           CSW  00078498 0C000001   SNS N/A        R/V CPA 00078470 00078470  FLG C0108801  A2000353 00
           TIME    44413.372394
DSP        ASCB 00FD5858  CPU 0000  JOBN USRT085   DSP PSW 075C0000 00C6B000  TCB 008B8EB8  MODN SVC-RES
           TIME    44413.375033
```

PGM 017 – The page fault. VPA=address of fault.

SRB – The dispatch of ASM's part monitor routine in master's address space.

SIO 353 – The Start I/O to page-in the requested page.

DSP – The dispatch of any ready work while the page-in I/O is in progress. In this case, there is no ready work, so the wait task is dispatched.

IO 353 – The I/O interrupt from the paging device. ASM's disable interrupt exit (DIE) routine gets control.

DSP – The faulter resumed where he left off.

* *Note:* This entry appears when ASM is unable to start the I/O in the page faulter's address space because ASM resources are unavailable.

Figure 2-16. GTF Trace of a Page Fault With I/O

Notes For Traces

The trace provides a history of some of the events that lead to a storage dump. Trace interpretation is one of the most important aspects of debugging.

Tracing Procedure

When attempting to recreate the process that was occurring on the processor(s) when the dump was taken, start at the current entry in the trace table (identified either by the trace header or by the highest clock value in the last column) and scan upwards. While scanning, look for unexpected events. These include:

- Unit check, unit exceptions on I/O devices
- Non-CC=0 on SIOs

- Non-type 11 program checks

- SVC D, SVC 33, SVC errors - (see number 6 under "Cautionary Notes" later in this chapter)

- Malfunction alerts (X'1200' external interrupt)

- Entries that show both processors executing the same code as indicated by the ICs (instruction counter) in the entries

- Large time gaps in the TOD clock value

- MP environment and only one processor doing anything

These entries indicate a potential for errors. Do not be distracted if you discover an entry of this type. Record the incident for future use. Then continue scanning back through the trace and try to determine what was happening in the system that might have caused the failure. Remember to conduct the scan by unique processor. Separate the processes that occur on each processor and watch for any obvious interactions in the processes.

You can further subdivide the activity by address space (as depicted by ASID) or by task (TCB address; remember to stay under the same ASID). As you recreate the situation, remember that you are relating individual entries to real events that must occur in order to accomplish work. Do not be distracted. For example, do not look for an I/O interrupt just because you see an SIO. The two events should be associated, but you should also determine the following:

- Why the I/O is occurring;

- If the I/O is related to the process, address space, task, page fault, etc. that you are concerned with;

- If the I/O completion should trigger another event. This is the way work is accomplished in MVS, that is, events triggering more events. As you become familiar with trace coding you learn to expect this "event causing" sequence. Certain sequences occur very frequently; you learn to recognize these and to look for less familiar sequences.

As you are searching trace entries, watch for repeating patterns, which can indicate a loop in the system. These patterns can appear as constantly repeating ICs (generally the case in a tight enabled loop), or as a repeating sequence of entries (often the case in a process loop, such as an ERP constantly retrying an I/O operation). Note that in the latter case, other entries from other processes can intervene periodically in the trace table, especially in an MP environment.

If you reach a point in the trace analysis where you are somewhat comfortable with the processes you are uncovering and recreating, and you feel you have a fair understanding of the activity in the system, pause. Try to understand what you have found. Is there any way you can relate your findings to the reason you have taken the dump in the first place? Do the unexpected events have anything to do with the problem, or are they unrelated to the problem? It can happen that the events you have discovered are unrelated to the problem causing the dump and you have exhausted the scope of the trace. In this case, you probably have to go into the system and study the address space and task structures, queues, and global data areas in order to zero in on the problem.

However, if the events you have discovered are related to the problem causing the dump, you must then attempt to isolate the erroneous process. Try to understand how the unexpected events relate to the process. Look on both sides of the event: did the event trigger the bad process, or is it a result of the bad process?

It is also necessary in trace analysis in MVS to understand whether you are looking at the primary error or at some secondary problem. Is this a mainline failure or a failure because of a problem in the recovery? Also, you must decide if the problem is caused by a previous error from which the system has recovered. Always be sure that it was not something several pages earlier in the trace that caused recovery to be activated and eventually led to the current problem. If this is the case you must now decide which error to pursue. The original error is probably more important; however, much of the required information might be lost because of recovery and the subsequent recovery failure. Also keep in mind that if you must attack the secondary error condition, your search of the dump and the recovery areas can often uncover information about the first error.

The trace is one of the most useful tools available for back-tracking through a problem sequence. You must use it in conjunction with system control blocks and indicators in order to recreate the error sequence. This is still true in MVS despite the fact that the trace contains less information than in previous systems. In MVS, the SVC calls have been greatly reduced because of branch entry logic for both transfer of control and supervisor services. This means that trace entries are not provided as in previous operating systems. Also, many significant events, such as lock acquisition and release, SRB scheduling, and SIGP issuance, are not traced. Because of these MVS considerations, you must be able to understand the processes and interpret the trace table rather than just read it.

Bypassing GTF Lost Events

The following superzap is useful when you need to trace a large number of events (such as identifying a performance problem during teleprocessing operations). It increases the number of GTFBLOKs from 4 to 32.

```
NAME        AHLCWRIT
VER   0194  50AE,CC03
VER   01BC  58AA,CC03
VER   01D2  5899,CC03
VER   0954  509A,CC03
VER   147C  587A,CC03
VER   148A  509A,CC03
VER   1674  0000,0004
REP   0194  50AE,CCF3
REP   01BC  58AA,CCF3
REP   01D2  5899,CCF3
REP   0954  509A,CCF3
REP   147C  587A,CCF3
REP   148A  509A,CCF3
REP   1674  0000,0020
```

Caution: Extreme care must be used when considering a system alteration in order to gather additional data about a problem. No superzaps should be applied before the system programmer has verified the logic being zapped and the trap logic itself. Remember, if any one location or offset within the module or trap changes, all offsets and base registers *must* be verified.

Cautionary Notes

Listed below are some items the problem solver should understand when analyzing an MVS trace table.

1. *I/O Processing:*

   - Much I/O is accomplished in MVS by the branch entry interface to IOS and without the use of SVC 0 (EXCP). Therefore, you often find I/O entries (SIO/I/O interrupt) that are not accompanied by SVC 0.

   - Back-end I/O processing can result in an SRB schedule of IECVPST. This trace entry should appear soon after an I/O interrupt. The register 1 slot will contain the IOSB address. The IOSB is the key to tracking the I/O request.

2. *Timer Value:*

   The last field of each trace entry contains bytes 3-6 of the eight-byte TOD clock at the time the entry was made. The second digit (from the left) represents the value to be increased approximately every second.

   The following steps show how to determine the elapsed time between two trace entries (such as from a SIO to the I/O interruption).

   a. Find the difference between the two hexadecimal timer values.
   b. Convert the difference to a decimal value.
   c. Divide the decimal value by 16 (result is microseconds).
   d. Divide by 1,000,000 (result is seconds).

3. *Enabled Wait State:*

   Because of recovery, the end symptom of many problems is an enabled wait state. For tracing, the wait state presents particular problems in MVS. SRM maintains a timer interval that periodically causes a clock comparator interrupt (code X'1004'). These external interrupts are recorded in the trace table. Also, an SRB is dispatched periodically in the master scheduler's address space to run a section of SRM code which updates the page frame tables unreferenced interval counts (UICs). In addition, in an MP/AP environment there are external calls (code X'1202') issued between the two processors requesting that the receiver look for ready work. These calls will be followed by a re-dispatch of the no-work wait on the receiving processor. In short, the wait state is a combination of dispatches of the no-work wait task, clock comparator interrupts, and SIGP external calls. The IC (instruction counter) will always be 0.

   All this extraneous activity can cause the trace to wrap around and overlay the important trace entries of the events that led up to the enabled wait state.

4. *MP/AP Activity:*

   The communication between the two processors in the MP/AP environment is traced as the external interrupts are accepted by the receiving processor. An external interrupt code of X'1201' is an emergency signal; and an external interrupt code of X'1202' is an external call. (The previous chapter, "Effects of MP on Problem Analysis," explains this communication process.)

5. *Trace Currency:*

   Various processes that occur in MVS turn off the MVS trace. The most prominent of these are GTF and SVC dump. Determine if the trace was running when the dump occurred: if you are unaware that the trace was *not* running when the dump was taken, you might go off on a fruitless chase and lose considerable time. The trace was still active when the dump occurred if the CVT+X'190' value =X'07FA'.

   Note: When SVC dump turns off the MVS trace, it sets bit 0 on in the ASID identifier (offset X'16') in the current trace table entry.

6. *SVC D Entries:*

   SVC D is the means by which termination is invoked. In previous operating systems, SVC D meant abnormal termination. This is not always true in MVS. RTM2 is the mechanism for normal end-of-task processing as well as for abnormal termination; RTM2 is invoked via SVC D. Consequently, SVC D for normal termination is a valid situation and is traced. You can determine whether SVC D implies normal or abnormal termination by inspecting the register 1 slot associated with the SVC D entry. If the first byte contains a X'08', RTM2 is being invoked for normal termination and this is not an error situation.

   MVS does not allow SVC routines to be invoked from code in one of the following states: cross memory mode, non-task mode, any lock held, disabled for I/O or external interruptions, or with any enabled unlocked task mode FRR established. However, SVC D issued in one of these states is a common means to enter RTM1 to invoke recovery. RTM indicators show the SVC error, and the system trace entries for the SVC and SVC error show the issuer's state, but the real problem is why SVC D was issued.

7. *Important events not traced:*

   Since the enabled nowork wait task dispatch entry is now made while enabled, the CS to obtain a slot in the trace table may be executed, but the MVC that moves the entry from PSAWTENT to that slot may never occur. This results in residual trace entries occurring among the "current" trace entries which can be of any type.

8. *Unit exception I/O interrupt on a 3705 communications controller:*

   The presence of unit exception conditions from the 3705 is a common occurrence while running VTAM. This is a normal situation and should not be considered erroneous. The host processor has issued a set of read commands to the 3705, and the channel program has been terminated before all the reads have completed because the NCP did not have enough data to satisfy each read CCW.

9. *GETMAIN, FREEMAIN - SVC X'A', SVC X'78':*

   For SVC X'A', inspect the register 1 slot of the associated trace entry. A value of X'80' in the high-order byte indicates GETMAIN; a value of X'00' indicates FREEMAIN. SVC X'78' uses a code in register 15 (see the *Debugging Handbook.*) If a GETMAIN is indicated, the register 1 slot of the associated re-dispatch of the SVC issuing code can be used to locate the storage allocated by the GETMAIN process.

10. *A GETMAIN for X'4D4' bytes is often seen soon after an SVC D is issued:*

    This is RTM2's request for storage for an RTM2WA and an SDWA for RTM2. By locating the re-dispatch of RTM2 and inspecting the register 1 slot, you can locate the RTM2WA.

Master Trace

Master trace is useful for debugging problems when you need to know the content of recently issued messages. Master trace maintains a wraparound table of the messages that are routed to the hardcopy log. When a message becomes eligible for the hardcopy log, it is entered into the master trace table by the communications task queuing routines. The trace table resides in pageable virtual storage in the master scheduler's private area.

The size of the master trace table is specified by the MT operand of the TRACE command and by the MT= entries in the COMMNDxx member of SYS1.PARMLIB. If a size is not specified, the default size is 24K bytes. The master trace table is created during master scheduler initialization. After system initialization, master trace may be activated and deactivated by using the TRACE command.

The master trace table is included in SVC dumps as a part of the SDATA=TRT option. The default size of 24K bytes accommodates approximately 336 messages (with an average length of 40 characters).

To locate the master trace table: field CVTMSER (at CVT+X'94') points to IEEBASEA (master scheduler resident data area) and field BAMTTBL at offset X'8C' in the IEEBASEA points to the master trace table.

When submitting an APAR, the SVC dump may be submitted for the hardcopy log if the master trace table contains the required messages. For example:

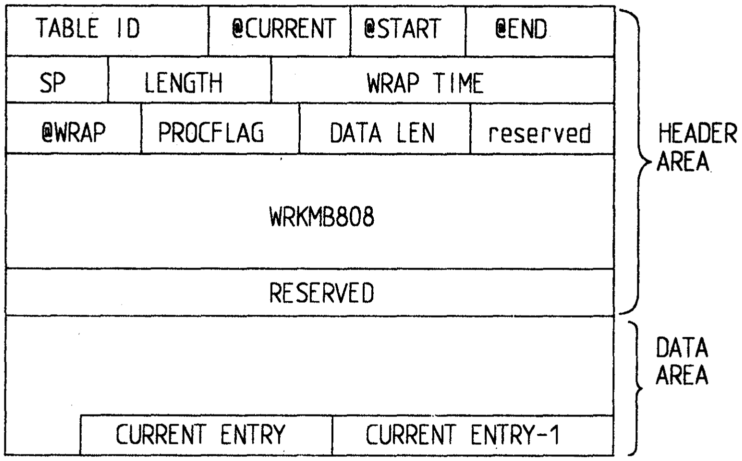
- The master trace table has wrapped at 9:00.
- A message related to a problem was issued at 9:20.
- An SVC dump is taken at 9:30.

In the example, the required messages are in the dump because the problem occurred between the time that the master trace table was wrapped and the dump was taken.

Master Trace Table

Master trace data is maintained in a wraparound table that is anchored in the master scheduler resident data area. The format of the master trace table and entries is described in the *Debugging Handbook*.

The table contains the following header and data areas:



Where:

TABLE ID A fullword field with a constant of 'MTT' to identify the beginning of the master trace table.

@CURRENT Address of the first byte of the current (most recently stored) entry.

@START Address of the first byte of the data area.

@END Address of the first byte beyond the end of the data area.

SP Subpool in which this table resides.

LENGTH Length of the header and data areas combined - the size specified on the TRACE command.

WRAP TIME Either the time that the table was initialized or the time at which the last table wrap occurred. The form is:

xx	- where IT indicates initialize time and WT wrap time
HH	- hours
MM	- minutes
SS	- seconds
T	- tenths of a second

@WRAP Address of the first byte of the last entry stored before the most recent table wrap. It is initialized to zero and remains so until the first table wrap.

PROCFLAG Processing flags used by IEEMB808.

DATA LEN Fullword field containing the size of the data area of the table.

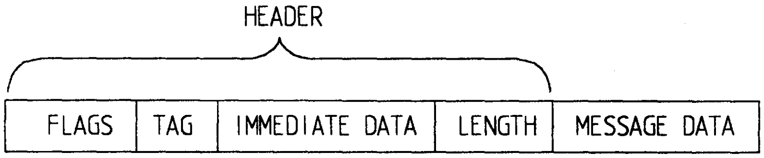
reserved Reserved fullword.

WRKMB808 Sixteen word work area for IEEMB808 and IEEMB809 -- serialized by CMS and local locks.

RESERVED Reserved four word area.

Master Trace Table Entry

The master trace table entries (located in the data area of the master trace table) contain the following fields:



Where:

FLAGS Halfword containing the flags set by the caller in the parameter list passed to IEETRACE.

TAG Halfword value indicating the identity of the caller. Values are defined in macro IEZMTPRM, which maps the parameter list.

IMMEDIATE DATA Fullword containing 32 bits that are defined by the caller. This area is stored in the table without validity checking by the trace service routine.

LENGTH Halfword containing the length of the message data.

MESSAGE DATA Variable length field containing message data provided by the caller. The maximum size is the length of the data area less 10 bytes, or 65,535 bytes, whichever is less.

The table entries are of variable length. If the length of the data is specified as zero by the caller, only the 10 byte header is entered in the table and used to store immediate data. The significance of the immediate data is defined by the caller.

Entries are placed into the table from back to front in the data area. Thus the current entry immediately precedes the entry stored on the previous call to IEETRACE.

The Message Processing Facility Table (MPFT)

By using the SET MPF (message processing facility) system command, you can suppress messages from being displayed on the operator's console.

The message processing facility builds the message processing facility table (MPFT, mapped by macro IEEZB809), which contains an entry for each message ID to be suppressed. The table includes the following:

- Table header:
  - Subpool number (where the table resides)
  - Size of the table (in number of bytes)
  - Number of entries in the table following the header
  - PARMLIB suffix from which the table was built
  - Address of the first entry in the table
  - Length of each entry
- Table entry (one 10-byte entry for each message to be suppressed):
  - Message ID
  - Length of the message
  - Flag byte

The MPFT is located in the common service area (CSA) and is pointed to by field UCMFMPFP in the UCM (from the fixed extension base).

Miscellaneous Debugging Hints

This chapter is a collection of miscellaneous debugging hints to aid the problem solver in specific situations not covered elsewhere in this book. It includes the following topics:

- Alternate CPU Recovery Problem Analysis
- Pattern Recognition
- OPEN/CLOSE/EOV ABENDs
- Debugging Machine Checks
- Debugging Problem Program ABEND Dumps
- Debugging From Summary SVC Dumps
- Started Task Control ABEND and Reason Codes
- SWA Manager Reason Codes

Alternate CPU Recovery (ACR) Problem Analysis

Alternate CPU recovery (ACR) is the process by which MVS dynamically adjusts to the unexpected failure of a processor in a multiprocessing (MP) configuration. ACR is initiated by the failing processor. If the failing processor's hardware detects the failure, it issues a malfunction alert (MFA) external signal to another processor. If the failing processor generates the severe machine check interrupt (recursive or invalid logout) type, the machine check interrupt handler will initiate ACR via the SIGP instruction with the emergency signal (EMS) order code, which generates an external interrupt on the receiving processor.

When a running processor detects that a failing processor is requesting ACR, it places X'FF' in the CSDACR byte (CSD+X'16') in the CSD control block. The byte will be restored to X'00' after ACR is complete.

ACR works in three phases: pre-processing, intermediate, and post-processing phase. Pre-processing is the initialization phase: the running processor copies the PSA and normal functional recovery routine (FRR) stacks of both processor's and places them in the area pointed to from their respective LCCA's WSACACR pointer. The WSACACR pointer is located at X'10' beyond the area pointed to by LCCACPUS. Additionally, LCCAs are marked so that in both processor's LCCAs, LCCADCPU points to the LCCA of the failing processor and LCCARCPU points to the LCCA of the running processor. By means of the LCCACPUA field in the LCCA, you can determine which processor has failed and which are still running.

Note that in a storage dump, the physical PSA of the failed processor is the same as it was when the processor decided that ACR should be initiated. The normal FRR stack, pointers to other FRR stacks, locks, PSASUPER bits etc. all reflect the state of the processor at the time it failed. This will be useful for solving problems in the recovery initiated for the process on the failed processor.

The ACR intermediate phase gets control from the MVS dispatcher, or lock manager global spin lock routine. In this phase, ACR switches from the process on one logical processor to the process on the other logical processor. This switching continues until the RTM1 recovery (routing to FRRs) completes on behalf of the process on the failed processor. At this point, the ACR post-processing phase is entered.

ACR post-processing invokes I/O restart (IECVRSTI) to initialize the channel reconfiguration hardware (CRH) function on a Model 168, or the channel set switching function on a processor that supports this function, or to mark outstanding I/O from the failed processor with a permanent error which then initiates error recovery processing via error recovery procedures (ERPs). Console switch is invoked via POST and the system resources manager (SRM) is notified of the loss of the processor. On a system with an MSSF, post-processing invokes the Service Processor Call SVC (IEAVMSF) to physically vary the processor offline. Finally, ACR issues message IEA858E 'ACR - COMPLETE', and resets the CSDACR flag to X'00'.

Note: The I/O error processing invoked during the ACR process has caused many of the problems discovered to date. Of significant importance is EXCP I/O error processing. The following flow describes the non-CRH situation for an MVS 158 MP system.

1. I/O restart (IECVRSTI) determines all devices that have outstanding requests at the time of a machine check.

2. IECVRSTI simulates an I/O interrupt for each device that was active on a failing channel and sets the channel control check and interface control check (X'00000000 00060000') bits in the CSW and the pseudo interrupt bit in the IRT (IRTPINT bit at X'02' in IRTENVR). This prevents IOS from interfacing with the channel check handler (CCH).

3. IECVRSTI passes control to IOS.

4. IOS sets the IOSCOD field in IOSP to X'74' and schedules IECVPST.

5. IECVPST routes control to the abnormal exit routine.

6. For an EXCP, the EXCP compatibility interface routine receives control.

7. EXCP converts the X'74' to X'7F' in the IOB.

8. EXCP branches to abnormal end appendage.

9. Abnormal end appendage returns to EXCP, which returns to IECVPST.

10. IECVPST invokes normal ERP processing.

11. If no path remains to a device, subsequent I/O requests (either ERP retry or normal new requests) are intercepted by IOS and flagged with IOSCOD=X'51' and IECVPST is scheduled.

12. IECVPST routes control to the abnormal exit routine.

13. For EXCP requests, the abnormal exit is again the EXCP compatibility interface routine.

14. EXCP converts the X'51' to a X'41' (permanent error) in the IOB and enters the abnormal end appendage.

15. The abnormal end appendage returns to EXCP; EXCP returns to IECVPST, which enters the termination routine.

The important point in the preceding discussion is that EXCP changes the ACR completion codes to conventional error post codes.

The most frequent I/O problems have been:

- ERP's abnormal end appendages not coded for a 0 CCW address in CSW.
- ERP's abnormal end appendages not recognizing that the last path to a device has been lost (as with asymmetric I/O) and thus going into an I/O retry loop.

Pattern Recognition

When analyzing a dump you should always be aware of the possibility of a storage overlay. System incidents in MVS are often caused by storage overlays that destroy data, control blocks, or executable code. The results of such an overlay vary. For example:

- The system detects the problem and issues an abnormal completion code, yet the error can be isolated to an address space.
- Referencing the data or instructions can cause an immediate error such as a specification or op-code exception.
- The bad data can be used to reference a second location, which then causes an evident error.

When you recognize that the contents of a storage location are invalid and subsequently recognize the bit pattern as a certain control block or piece of data, you generally can identify the erroneous process/component and start a detailed analysis. This section discusses pattern recognition and potential causes of storage overlays, and points out common patterns that aid the debugger.

Once you recognize an overlay, analyze the bit pattern. If you do not recognize the pattern at all, try to determine the extent of the damaged area. Look at the data on both sides of the obviously bad areas. See if the length is familiar; that is, can you relate the length to a known control block length, data size, MVC length, etc.? If so, check various offsets to determine their contents and, if you recognize some, try to determine the exact control block/data. Even if you do not recognize the pattern, take one more step. Can you determine the offset from some base (X) that would have to be used in order to create the bit pattern? If so, the fact that there is a certain bit pattern at a certain offset (Y) can be helpful. For example, a BALR register value (X'40D21C58') at an offset X'C' can indicate that a program is using this storage for a register save area (perhaps caused by a bad register 13). Another field in the same overlaid area might trigger recognition.

Look at the overlaid area and scan for familiar addresses such as device addresses, UCB addresses, and BAL/BALR register values (fullword with high-order byte containing some "1" bits). If you find any of these, try to determine what components or modules are involved or what control blocks contain these addresses.

Repetition of a pattern can indicate a bad process. If you can recognize the bad data you might be able to relate that data to the component or module that is causing the error. This provides a starting point for further analysis.

Low Storage Overlays

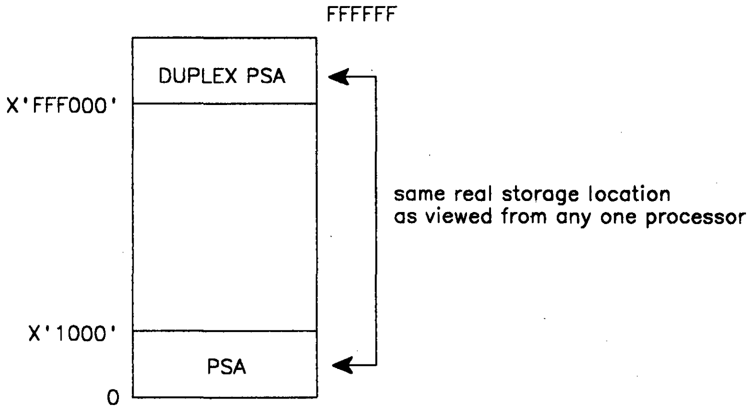
Low storage overlays are generally very difficult problems to solve, primarily because the error is not detected until some time after the error occurs. In order to reduce the number of these incidents, a low address protection feature exists. The feature can be enabled or disabled. When the feature is enabled, any attempt to store into virtual locations 0 through X'1FF' (even with PSW key=0) results in a protection exception. Any routine using a zero pointer as a control block address will be caught when it attempts to store into the control block (assuming the control clock is less than X'200' in size).

Several fields are used to control low address protection. Examine the following fields.

control register 0, bit 3
CVTPRON
PSACR0CB
PSACR0SV bit 3

Hardware uses control register 0 bit 3 to determine if the feature is enabled or disabled. At IPL time, if CVTPRON=B'1' (default value), control register 0 bit 3 is set to B'1' and PSACR0CB is set to X'01' The control register is then saved in PSACR0SV. If CVTPRON=B'0', the corresponding fields are set to 0. PSACR0CB is used as a mask byte by the PROTSPA macro when enabling low address protection. The macro enables low address protection (when the byte=X'01') by altering the value in PSACR0SV and loading control register 0 from that field.

The low address protection feature operates on a logical address (that is: prior to translation and prefixing being performed). Therefore, if a program uses a virtual address that translates to an address between 0 and X'1FF', a protection exception may not occur. However, the PSW key must be zero in this case. This method is used by IOS module IECIOSAM to store the channel address word (CAW) prior to issuing the SIO instruction. IECIOSAM is the only module that uses the duplex PSA. It is always at location X'FFF000'.



Note: The page at X'FFF000' is not backed by real storage but translates to real location 0.

The following is critical data in low storage and may be overlaid only when protection is disabled:

- Location X'10' (CVT pointer) should contain a nucleus address. This location is refreshed by the program check first level interrupt handler and so is often valid when adjacent locations are bad.
- Locations X'18' through X'3F' (old PSWs) should always contain valid PSWs.
- Location X'4C' should be equal to location X'10'.
- Locations X'58' through X'7F' (new PSWs) should contain valid PSWs.

If any of the above statements is not true, consider a low storage overlay. Further analysis is required to determine what the cause may be. Also consider that, on a non-prefixed machine, the low storage locations described above can be overlaid by CCWs for the stand-alone dump program, starting at location X'10'. Do not consider this an error situation.

Two common low storage problems are:

- A register save area starting at location X'30'. This can happen when an area of the system saves register status in a TCB at location 0. Or it can be caused by a routine using PSATOLD for a TCB address when the system is in SRB mode; this is indicated by PSATOLD=0.
- An SRB/IOSB combination starting at location X')'. This can be caused by a problem in the IOS storage manager. The contents vary depending upon how many control blocks the code has initialized. Points to consider are:

  1. The two blocks might point to each other (X'1C' into each).
  2. An ASCB address might be at location 8.
  3. Addresses of IECVEXCP routines might be at X'68' and/or X'6C'.

Common Bad Addresses

Common bad addresses are:

- X'C0000', and this address plus some offset. These are generally the result of some code using 0 as the base register for a control block and subsequently loading a pointer from 0 plus an offset, thereby picking up the first half of a PSW in the PSA.

  Look for storage overlays in code pointed to by an old PSW. These overlays result when 0 plus an offset cause the second half (IC) of a PSW to be used as a pointer.

- X'C00', X'CE0', X'D00', X'D08', X'D20', and other pointers to fields in the normal FRR stack. Routines often lose the contents of a register during a SETFRR macro expansion and illegally use the address of the 24-byte work area returned from the expansion.

- Register save areas. Storage might be overlaid by code doing an STM (Store Multiple) instruction with a bad register save area address. In this case, the registers saved are often useful in determining the component or module at fault.

OPEN/CLOSE/EOV ABENDs

When a dump shows an abend issued from O/C/EOV, the key area to start your diagnosis in is the RTM2 work area. The failing TCB has a pointer (at TCB+X'E0') to this area. This work area contains information current at the time of the abend, the most important being the register contents. Register 4 points to the current O/C/EOV work area. This work area is built by IFG0RR0A during problem determination and contains key information about the problem: the JFCB, IOB, DEB and other pertinent fields are all saved in the work area for use later by the recovery routines. The O/C/EOV work area is documented on microfiche in each O/C/EOV module.

The module in control at the time of the abend can be determined from the "Where To Go" (WTG) table, which is pointed to by register 6 in the RTM2 work area. The WTG table is contained within another work area called the O.C. work area. IFG0RR0A saves a copy of the current DCB in this work area. If multiple DCBs are involved, the prefix to the DCB work area points to another DCB work area. These DCB areas are laid out precisely like a DCB. All these work areas and their prefixes are documented at the end of every O/C/EOV module in the microfiche.

In an MVS environment, O/C/EOV must build these work areas rather than rely on what is in real storage at the time of the dump. The main task is to find these areas and interpret their fields using microfiche. A quick way to find these work areas is to find subpool 230 in the dump. All O/C/EOV data is in this subpool.

Assuming you have all the pertinent information about the failure, the problem becomes the same as an O/C/EOV problem in OS. One more point: built into the code is message IEC999I. This message indicates that there is a problem in the

O/C/EOV code that cannot be determined. While you may be able to circumvent this problem, you should also submit an APAR for it.

Debugging Machine Checks

The machine check interruption is the hardware's method of informing the MVS control program that it has detected a hardware malfunction. Machine checks vary considerably in their impact on software processing. Some machine checks notify software that the processor detected and corrected a hardware problem that required no software recovery action (software calls these errors soft errors). Hard errors are hardware problems detected by a processor but that require software-initiated action for damage repair. Hard errors also require software recovery to verify the integrity of the process that experienced the failure. Obviously, if there are software problems after a machine check, it is more likely that the machine check was a hard error. It is important to get a feeling for which software components are affected by particular hardware failures.

The machine check interrupt code (MCIC), located in the PSA (at X'E8'), describes the error causing the interrupt. (Refer to *Principles of Operation* for a complete description of the MCIC.) The following discussion shows how to find MCICs and how to interpret them for subsequent software processing. Machine checks can be found in a LOGREC buffer (LRB), the SYS1.LOGREC data set, or in the storage area used as a buffer prior to writing records to SYS1.LOGREC (see the discussion of SYS1.LOGREC analysis in the "Recovery Work Areas" chapter earlier in this section). Also, a pointer to the LRB that describes the last machine check that occurred on a processor can be found in that processor's PCCA at
PCCALRBV (PCCA+X'A0'). The LRB contains the machine check interrupt code (MCIC), except when:

- The machine check old PSW is zero. The MCIC is also zero. The LRBMTCKS bit (field LRBTERM at LRB+X'20') is turned on by software.

- MCIC is zero and the machine check old PSW is non-zero. The LRBMTINV bit (field LRBTERM at LRB+X'20') is turned on by software.

The MCIC is the principal driver of software processing after a machine check. It must be examined to determine the actions that MVS should take. The MCIC contains bits describing the conditions that caused the interrupt. Note that more than one failing condition can be described by a machine check at one time. Software performs repair processing for each condition found; software recovery processing is initiated if any hard error conditions are found (except in the cases described on the following pages).

Because hard errors require FRR and ESTAE processing, identifying a hard error is important. Important MCIC bits follow with a description of their hardware significance and impact on software. A handy MCIC reference matrix, containing additional machine check and ensuing action-taken information appears at the back of this section.

Bit 0 (System damage) - The processor is still useable, but damage occurred while the processor was in the process of changing PSWs or otherwise changing system control, and thus has lost the associated process or interrupt. Software recovery routines (FRRs) are entered for this hard error.

Bit 1 (Instruction processing damage) - The processor is still useable, but an instruction has failed to operate as intended. Software recovery is initiated for this hard error, unless the backed-up bit is on with storage error or key error uncorrected on refreshable storage (see Bit 16 description).

Bit 2 (System recovery) - The processor detected and corrected a potential hardware problem. The interrupted process is completely restored by software for this soft error; no repair is performed and no recovery routines are entered.

Bit 3 (Timer damage) - The interval timer at PSA location X'50' has failed. Because MVS does not use this timer, this failure is ignored (indicated as a soft error).

Bit 4 (Timing facility damage) - Damage has occurred to the CPU timer, clock comparator, or time-of-day clock. The particular clock facility that is damaged is described by MCIC bits 46 and 47. A first failure to a facility results in an attempt to reuse it. Subsequent failures result in taking the facility offline (described in the PCCA fields PCCATODE, PCCACCE, or PCCAINTE). If no clock of a particular type remains in the system, any task which requests timing using that type of clock is sent through software recovery. This is treated as a soft error for the process current on the processor at the time of the interrupt.

Bit 5 (External damage) - Damage has occurred to a unit external to the processor. MVS expects more information in a channel check I/O interrupt. This is treated as a soft error.

Bit 7 (Degradation) - The system has detected that elements of the high-speed buffer (cache) or translation look-aside buffer have had bit (parity) errors. The bad elements are automatically reconfigured out of the buffer. Once a predefined threshold of degradation machine checks is reached, the buffer and the translation look-aside buffer are reset, thus making the entire buffer available again. This threshold has a default value of 3 which can be changed by the operator via the MODE command. Until then, the system might perform at a reduced rate because of increased storage access time (cache element deletion) or increased time to translate virtual addresses (because of translation look-aside buffer element deletion). However, because no damage has been done to any software process or data, this soft error is merely recorded in SYS1.LOGREC. The system state at the time of the error is re-established, ignoring the occurrence of the buffer bit error. It is treated as a soft error and no software recovery is initiated.

Bit 8 (Warning) - Damage is imminent; there is a cooling loss or a power drop, etc. Software determines if the error is transient or permanent. If it is transient, the warning interrupt is treated as a soft error. If permanent, an attempt is made to invoke the power warning feature software, to record the system state at the time of this hard error.

Bit 16 (Storage error uncorrected) - There is a block in storage with a double bit error that is located at the real, prefixed address stored in PSA location X'F8'. If the frame's page is refreshable, that is, unchanged, pageable, and in the current address space, it is marked invalid so a future reference will cause a fresh copy to be paged into a new frame. (*Note:* More than one error can occur before the page goes offline.) In all cases, an attempt is made to take the damaged frame offline (unless the frame is in the nucleus). For unchanged nucleus frames, the page is refreshed from a copy paged-out at NIP time. When a storage error uncorrected condition occurs in conjunction with a system recovery or external damage error, it is treated as a soft error and no recovery routines are entered. If the storage error occurs in conjunction with instruction processing damage when the backed-up bit (bit 14) and storage logical validity bit (bit 31) are on, and the frame's page is refreshable, the error is treated as soft and no recovery routines are entered.

Any other occurrences of storage error uncorrected are treated as hard errors and software recovery is initiated for the error.

Bit 17 (Storage error corrected) - A single-bit storage error was detected and successfully corrected by hardware. Software treats this error as a soft error. This error sometimes appears in conjunction with system recovery (bit 2). For a double bit storage error, see bits 16 and 19.

Bit 18 (Storage key error uncorrected) - Hardware has detected a bit error in a storage key. Software attempts to reset the storage key to its original value. If the key is successfully reset, and the storage key error occurs in conjunction with instruction processing damage when the backed-up bit (bit 14) and the storage logical validity bit (bit 31) are on, the error is treated as soft and no recovery routines are entered. When the storage key error occurs in conjunction with a system recovery or external damage error, it is also treated as a soft error and no recovery routines are entered. Change bits are set to one in case the frames have been altered. Any other occurrences of storage key error are treated as hard errors and software recovery is initiated for the error.

Bit 19 (Double bit storage error) - If the storage error corrected bit (bit 17) is also on, bit 19 indicates that a double bit storage error was detected and successfully corrected by hardware. If the page containing the data can be paged, software obtains a new frame, moves the data from the frame that has the indicated double bit error correction into the new frame, and then marks the frame that had the double bit error offline. If the page containing the data cannot be paged, software marks the associated frame as intercepted to go offline, which causes the frame to be taken offline when the page is freed.

In addition to these error description bits there are other MCIC fields that describe the time-of-occurrence of the machine check interrupt, or the validity of the registers, PSW, and other data logged out during the machine check interruption process.

The two time-of-occurrence bits are bits 14 and 15. The backed-up bit (bit 14), when set to 1, indicates that the machine check occurred before actual damage occurred. The delayed bit (bit 15) is set to 1 when the processor has been disabled for one or more of the interrupt conditions described in the MCIC. The processor had been processing after damage was detected.

Validity bits describe the validity of the associated field logged out during the machine check interrupt. If a validity bit is 0, the associated data logged out is incorrect. Validity bits are:

- Bit 20 (PSW EMWP mask validity)
- Bit 21 (masks and key validity)
- Bit 22 (program mask and condition code validity)
- Bit 23 (instruction address of machine check old PSW validity)

- Bit 24 (failing storage address validity)
- Bit 25 (region code validity)
- Bit 27 (floating point register validity)
- Bit 28 (general purpose register validity)
- Bit 29 (control register validity)
- Bit 30 (processor model-dependent logout validity)
- Bit 46 (CPU-timer validity)
- Bit 47 (clock comparator validity)

Additionally, the storage logical validity bit (bit 31 set to 1) indicates that all store operations (that were to occur before the machine check interrupt) have completed.

The following chart attempts to show the action taken for each error condition. For example: In column 6 the condition involves recursive machine checks, *or,* a check stop, *or,* invalid logout. The condition originated on either a Model 158 *or* a Model 168 attached processor system, and did *not* involve the APU. The action taken resulted in a disabled wait. Where multiple errors do exist, appropriate repair action is taken for all errors, and recovery action is taken for the most severe error.

With the exception of I/O reserve outstanding, the status of each of the conditions can be determined from examination of MCH SYS1.LOGREC records.

CONDITION		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
Recursion		X	X	X	X	X	X																								
Check Stop			X	X	X	X	X																								
Invalid Logout		X	X	X	X	X	X																								
Subclass (MCIC)	System Damage							X																							
	Inst. Proc'g. Damage								X	X	X	X																			
	System Recovery												X																		
	Timer Damage												X																		
	Clock Damage													X	X	X	X	X	X												
	External Damage												X																		
	Degradation												X																		
	Warning																			X											
Time	Backed Up								X	X		O																			
	Delayed								O	O																					
Type	Stor Err Uncorr								X			X									X	X	X	X	X						
	Stor Err Corr																									X					
	Key Error									X																	X	X			
	Key Err Unresetable																												X	X	
Validity	PSW (WP, MS, PM, IA)								X	X																					O
	Failing Stor Addr								X	X															O						
	Registers (FP, GR, CR)								X	X																					O
	Logout																														
	Storage Logical								X	X																					
	CPU Timer													O	O	X	X	X	X												
	Clock Comparator													X	X	O	O	X	X												
Location	Pageable								X	X		X									X	X					X			X	
	Nucleus									X													X				X		X		
	LSQA, SQA									X														X			X			X	
	Fixed									X														X			X				
	V=R																							X				X			
	Outside Curr. Memory								O	O		O									X										
Storage State	Changed																					X									
	Unchanged								X			X									X		X								
System	UP	X																													
	MP		X	X	X																										
	AP					X	X																								
Processor	158			X	X	X	X																								
	168		X			X	X																								
	APU					X	O																								
I/O	Reserve Outstanding				X																										
Occurrence	1st													X		X		X													
	2nd														X		X		X												
ACTION TAKEN																															
Reset timing component														X		X		X													
Mark CPU Timer perm. damaged															X																
Mark Clock Comp perm. damaged																	X		X												
Mark TOD Clock perm. damaged																			X												
Invoke PWF if available																				X											
Activate CRH			X																												
Take frame offline immed.									X			X																			
Take frame offline when avail.																					X	X		X				X		X	
Invalidate Page Table Entry									X			X																			
Repair SPF Key										X																	X				
Disabled Wait		X					X													X											
Restartable Wait					X																										
Enter RTM for Recov.*			X	X	X	X		X			X	X									X	X	X	X	X		X	X	X	X	X
Record		X	X	X	X	X		X	X	X	X	X	X	X	X	X	X	X	X		X	X	X	X	X	X	X	X	X	X	X
Take Processor offline			X	X	X	X																									
Resume at MCOPSW									X	X			X	X	X	X	X	X	X							X					
Refresh the nucleus page																							X								

*Possible loss of Job.

Notes:

- *Key.* X = Condition must be present

  O = Condition must *not* be present

  X X = The action is the same no matter which condition represents the situation

Debugging Problem Program Abend Dumps

The following steps may provide some initial assistance in this debugging process:

1. Locate the RTM2 work area (RTM2WA), which is pointed to by the TCBRTWA field in the TCB and the ESART2WA field in the abend SVRB. It provides a summary of the abend as follows:

Name	Offset	Explanation
RTM2CC	1D	Abend completion code.
RTM2ABNM	8C	Abending program name. This is the name of a load module or an external entry point (ALIAS) in the load module.
RTM2ABEP	94	Abending program address (the beginning of the load module or an ALIAS in the load module).
RTM2EREG	3C	Registers at time of error.
RTM2APSW	7C	EC PSW at time of error.
RTM2ILC1	85	Instruction length code for PSW at time of error.
RTM2ERAS	36C	Error ASID.
RTM2TRCU	37C	Address of current trace entry for saved system trace table.
RTM2TRFS	380	Address of first trace entry for saved system trace table.
RTM2TRLS	384	Address of last trace entry for saved system trace table.
RTM2ERRA	B4	Error type.

Notes:

a. *The RTM2ABNM and RTM2ABEP fields do not contain information about the abending program if an SVC has abended.*

b. *In a recursive abend (an abend occurring while the original abend is being processed by an ESTAE or other recovery routine), more than one RTM2WA may be created, and the RTM2PREV or RTM2PRWA field points to other RTM2WAs associated with the problem. The system diagnostic work area (SDWA) is pointed to by the RTM2RTCA field during recovery routine processing, and has register contents at time of error stored in the SDWAGRSV field. These register contents may differ from those in the RTM2WA after a recursive abend.*

2. To find the abend code and its explanation, look at the completion code at the top of the abend dump. A user completion code is printed as a 4-digit decimal number and a system completion code is printed as a 3-digit hexadecimal number.

   If the user code is non-zero, a user program has specified the completion code in an abend macro instruction. Looking up the name of the abending program in the RTM2WA, and investigating why the program would issue this completion code, should lead directly to the cause of the error in the user program.

Usually the *system* code is non-zero. This indicates that a system routine issued the abend but a problem program might indirectly have caused the abnormal termination. For example, a problem program might have branched to an invalid storage address, specified an invalid parameter on a macro instruction, or requested too much storage space.

Often the explanation of the system code gives enough information to determine the cause of the termination. The explanations of system completion codes, along with a short description of the action for the programmer to take to correct the error, are contained in *Message Library: System Codes.*

3. To find the name of the abending program look in the RTM2 work area. System routines usually start with the letters A or I; and module prefixes for system routines are listed in the *Debugging Handbook* Volume 1.

   Note: If the RTM2 work area is not available, or if the name of the abending program is not given in the RTM2 work area, the routine name can be obtained from the contents directory entry (CDE) queued to the program request block (PRB). If the ABEND dump was taken to a data set (or to SYSOUT) specified with a SYSABEND, SYSMDUMP, or SYSUDUMP DD statement, the last two RBs are SVRBs for the SNAP and SYNCH SVCs used to take the dump. The SVC numbers can be checked by obtaining the hexadecimal SVC number from the interruption code of the WC-L-IC field in the RB. The *Debugging Handbook* contains a list of SVC numbers. The SNAP SVC is hexadecimal '33', and the SYNCH SVC is hexadecimal '0C'. The RB for the program that caused the abend is immediately before these two RBs.

   CSECTs within load modules in the private area of an address space can be located using a linkedit map produced by the AMBLIST service aid. CSECTs in load modules in the nucleus, FLPA, or PLPA can be located using a nucleus or link pack area map, also produced by AMBLIST.

4. To find the instruction that caused a program interrupt (program check) completion code (0Cx) in a problem program, examine the PSW at the time of error. It is at the top of the abend dump, in the RTM2 work area, and in the RB for the program that caused the abend. The instruction address field in the PSW contains the address of the next instruction to be executed.

   The length of the abend-causing instruction is printed following the instruction length code's title 'ILC' at the top of some abend dumps. It is also located in the RTM2ILC1 field (see the RTM2 work area), and is formatted in the third and fourth digits (00xx0000) of the WC-L-IC field in the PRB. The address of the instruction that caused the termination can be found by subtracting the instruction length from the address in the PSW.

   Subtract the program address found in the RTM2WA (and in the last PRB) from the instruction address. The resulting offset can be used to find the matching instruction in the abending program's assembler listing for this CSECT.

5. To find the cause of a program interrupt, check the explanation of the system completion code and the instruction that caused the interrupt. Also check the registers from the time of error which are saved in the RTM2WA and in the SVRB following the RB for the program that caused the abend. The formatted save area trace can be used to check the input to the failing CSECT.

6. To find the cause of an abend code from an SVC or from a system I/O routine, check the explanation of the system completion code, then find the last instruction executed in the failing program and examine the related SVC and I/O entries in the trace table or GTF trace records.

   The last PRB in the formatted RBs has a PSW field containing the address of the instruction following the instruction that issued the SVC. For I/O requests, check the entry point address ('EPA') field in the last PRB. The formatted save area trace gives the address of the I/O routine branched to, and the return address in that save area is the address of the last instruction executed in the failing program.

   The trace information can be checked for SVC entries that match the formatted SVRBs, or for I/O entries issued from addresses in the failing program. The trace information is formatted in the dump if the installation has specified it as a dump option. If the system trace table is not formatted, look in the RTM2 work area for pointers to the copy of the system trace table that was saved from the time of the error. Location X'54', which is the FLCTRACE field in the prefixed save area (PSA), points to the system trace table header. The system trace table is frequently overlaid with entries for other system activity by the time the dump is produced.

   If the dump contains trace records, begin at the most recent entry and proceed backwards to locate the most recent SVC entry indicating the problem state. From this entry, proceed forward in the table. Examine each entry for an error that could have terminated the SVC or I/O system routine. The format of system trace table entries is described in the *Debugging Handbook* under the heading 'TTE Trace Table Entry.' The format of GTF trace records is also described in the *Debugging Handbook*.

7. In a cross memory environment, many services are requested by use of the Program Call (PC) instruction rather than by SVCs or SRBs. When an abend is issued by the PC routine, it can be confusing trying to identify the caller and exactly where the PC instruction was issued. This is because the RB old PSW contains the instruction address of the PC routine issuing the abend and the abend SVRB contains the registers of the PC routine.

   To determine if a program is in cross memory mode, examine the SASID and PASID fields in the XSB control block. If they are not equal, the program is executing in cross memory mode. To locate the XSB:

   - In a formatted dump, the XSB is printed following the RB with which it is associated.

   - In storage, field RBXSB (RB-X'20') points to the XSB.

In cross memory mode, you can determine the caller of a PC routine by examining the PCLINK stack. To locate the PCLINK stack element (STKE):

- In a formatted dump, the STKEs are printed following all of the RBs. If there is more than one STKE, the pointer to the one you want is contained in field XSBSTKE (XSB+X'18') of the XSB associated with your RB.

- In storage, field RBXSB (RB-X'20') points to the XSB and field XSBSEL (XSB+X'1C') points to the current STKE.

Important fields in the STKE are:

- STKERET (STKE+X'18') - contains the return address of the caller of the PCLINK service.

- STKEPR15 (STKE+X'1C') - contains parameter register 15 passed to the PC routine.

- STKEPRM0 (STKE+X'20') - contains parameter register 0 passed to the PC routine.

- STKEPRM1 (STKE+X'24') - contains parameter register 1 passed to the PC routine.

- STKESA (STKE+X'14') - contains the address of the previous save area passed by the caller of the PC service.

Debugging From Summary SVC Dumps

The summary dump area formatted by the SUMDUMP option of SDUMP should contain the most current data relevant to the problem present in the dump. It is strongly recommended that the SUMDUMP output be reviewed prior to investigating the usual portions of the dump. The SUMDUMP option provides different output for SVC and branch entries. For example, branch entries generally dump PSA, LCCA, and PCCA control blocks, and SVC entries generally dump RTM2WA control blocks. Each output type is indicated by the header "----tttt---- RECORD ID X'nnnn'," where *tttt* is the title for the type of SUMDUMP output, and *nnnn* is the hexadecimal record identifier assigned to the type. The record id values are described in the table below. They are also described by the IHASMDLR mapping macro in the *Debugging Handbook*.

SUMDUMP Output For SVC-Entry SDUMP

The following table summarizes the SUMDUMP output types for an SVC entry to SDUMP:

SVC-ENTRY TABLE

Record ID			Mapping	Fields used to Dump
Dec	Hex	Title	Macro	PSW or Register Areas
4	4	TRACE TABLE	TTE	-
46	2E	SUMLIST RANGE	-	-
48	30	REGISTER AREA	-	-
49	31	PSW AREA	-	-
53	35	NORMAL DATA END	-	-
57	39	RTM 2 WORK AREA	IHARTM2A	RTM2NXT1 RTM2EREG
58	3A	RTM2WA TRACE TAB	TTE	-
60	3C	ASID INFO	-	-

For an SVC entry to SDUMP, the SUMDUMP output can contain information that is not available in the remainder of the SVC dump if options such as region, LSQA, nucleus, and LPA were not specified in the dump parameters.

For each address space that is dumped, the SUMDUMP output is preceded by a header with the ASID, plus the jobname and stepname for the last task created in the address space. The SUMDUMP output contains RTM2 work areas for tasks in address spaces that are dumped. Many of the fields in the RTM2WA provide valuable debugging information. (See "Debugging Problem Program ABEND Dumps" for more details.)

Each RTM2WA is followed by 'RTM2WA TRACE TAB' output (record id X'3A'), if there is a copy of the system trace table associated with the RTM2WA (RTM2TRCU, RTM2TRFS, and RTM2TRLS fields are non-zero). The current entry in the trace table copy is pointed to by RTM2TCRU (offset 37C) in the associated RTM2 work area. System trace table entries are mapped by the TTE (Trace Table Entry) section in the *Debugging Handbook.*

Each RTM2WA is also followed by 'PSW AREA' output (record id X'31'). A PSW area, consisting of the instruction pointed to by the RTM2NXT1 field in the EC PSW saved in the RTM2WA, and the preceding instruction with length from the RTM2ILC1 field, is dumped if the instructions can be accessed.

After information for all RTM2WAs associated with a task is dumped, 'PSW AREA' (record id X'31') and 'REGISTER AREA' (record id X'30') output appears. This consists of 2K of storage before and after each valid unique address pointed to by the PSW and the registers from the time of the error (RTM2NXT1 and RTM2EREG fields) from all the RTM2 work areas. Up to 32 unique addresses can be dumped for each task. Register addresses less than 2K are not dumped because they are considered to be counters. If the storage that is 2K before and after an address cannot be accessed, a length of 300 bytes is tried. If that amount of storage cannot be accessed, the address′ record entry appears with a zero length.

'TRACE TABLE' output (record id X'04') appears if the first address space dumped has no trace table saved in an RTM2 work area and the system trace was active. The output includes the header (pointers to the current, first, and last entries) and the entries in the system trace table. System trace table entries are mapped by the trace table entry (TTE) described in the *Debugging Handbook.*

'SUMLIST RANGE' output (record id X'2E') appears at the beginning of the SUMDUMP output if the SUMLIST keyword was specified in the SDUMP macro instruction.

SUMDUMP Output For Branch-Entry SDUMP

The following table summarizes the SUMDUMP output types from a branch entry to SDUMP:

BRANCH-ENTRY TABLE

Record ID			Mapping	Fields used to Dump
Dec	Hex	Title	Macro	PSW or Register Areas:
1	1	PCCA	IHAPCCA	-
2	2	LCCA	IHALCCA	-
3	3	PSA	IHAPSA	FLCIOPSW, FLCPOPSW FLCEOPSW, FLCROPSW
4	4	TRACE TABLE	TTE	-
5	5	FRR STACK	IHAYSTAK	-
6	6	GWSA PAGE IO ERR	-	-
7	7	GWSA GET/FREEMAIN	-	-
8	8	GWSA RSM	-	-
9	9	GWSA RSM SUSPEND	-	-
10	A	GWSA MEM SWITCH	-	-
11	B	GWSA STATUS	-	-
12	C	GWSA SRM	-	-
13	D	GWSA MEM TERM	-	-
14	E	GWSA ENQ/DEQ	-	-
15	F	GWSA STOP/RESTRT	-	-
16	10	GWSA IEAVESC0	-	-
17	11	CWSA LOW-LVL CMN	-	-
18	12	CWSA GTF	-	-
19	13	CWSA SRM	-	-
20	14	CWSA TIMER	-	-
21	15	CWSA ACR	-	-
22	16	CWSA RTM/MACHK	-	-
23	17	CWSA IOS FLIH	-	-
24	18	CWSA DISPATCHER	-	-
25	19	CWSA MF1	-	-
26	1A	CWSA ABTERM	-	-
27	1B	CWSA I/O RESTART	-	-
28	1C	CWSA STATUS	-	-
29	1D	CWSA SUPR REPAIR	-	-
30	1E	CWSA RTM-CCH	-	-
31	1F	LWSA LOW@LVL CMN	-	-
32	20	LWSA VALID'Y CHK	-	-
33	21	LWSA RTM	-	-
34	22	LWSA SDUMP	-	-
35	23	LWSA ABTERM	-	-
36	24	LWSA CIRB	-	-
37	25	LWSA STG2 EXT EF	-	-
38	26	LWSA EXIT (SVC3)	-	-
39	27	LWSA POST	-	-
40	28	LWSA WAIT	-	-
41	29	LWSA STATUS	-	-
42	2A	LWSA STAE	-	-
43	2B	LWSA EVENTS	-	-
44	2C	LWSA RSM	-	-
45	2D	LWSA ASCB CHAP	-	-
46	2E	SUMLIST RANGE	-	-

BRANCH-ENTRY TABLE (Continued)

Record ID			Mapping	Fields used to Dump
Dec	Hex	Title	Macro	PSW or Register Areas:
47	2F	INT HANDLER SA	IHAIHSA	IHSAGPRS
48	30	REGISTER AREA	-	-
49	31	PSW AREA	-	-
50	32	GBL WSA VEC TABL	IHAWSAVT (WSAVTG)	-
51	33	CPU WSA VEC TABL	IHAWSAVT (WSAVTC)	-
52	34	LCL WSA VEC TABL	IHAWSAVT (WSAVTL)	-
53	35	NORMAL DATA END	-	-
54	36	CWSA ASM DIE	-	-
55	37	CWSA ASM SRB@I/O	-	-
56	38	SDWA	IHASDWA	SDWAGRSV
60	3C	ASID INFO	-	-

The SUMDUMP output for a branch entry to SDUMP might not match the data that is at the same addresses in the remainder of the dump. The reason for this is that the SUMDUMP is taken at the entry to SDUMP, and while the processor is disabled for interrupts. The system data in the remainder of the dump is often changed because other system activity occurs before the dump is complete. The SUMDUMP output is preceded by a header with the ASID for the failing address space.

From a branch entry into SDUMP, the SUMLIST range and trace table output is handled similarly to that from an SVC entry. However, SUMLIST addresses must point to areas that are paged-in or they cannot be dumped.

The PSA, LCCA, and PCCA are dumped for each alive processor (record ids X‘03’, X‘02’, and X‘01’ respectively).

The interrupt handler save area (IHSA - record id X‘2F’) is dumped for the current address space. This save area includes the current FRR stack for suspended address spaces.

The system diagnostic work area (SDWA - record id X‘38’) is dumped for the current error if the RTM1 work area is currently valid and being used.

Unique register contents are obtained from the IHSA and the current SDWA. Each unique register value is used as an address and storage is dumped from 2K plus and minus this address for a total of 4K each. These ‘Register Areas’ are printed with record id X‘30’.

The Super FRR Stack (record id X‘05’), including RTM1 work areas are dumped.

The global, local, and processor work save area vector tables (record id X‘32’, X‘34’, and X‘33’ respectively) are dumped. The save areas pointed to by these save area vector tables are also dumped. The branch-entry table at the beginning of this description lists the record ids for each work save area.

2K of storage on either side of the address portion of the I/O old PSW, the program check old PSW, the external old PSW, and the restart old PSW saved in the PSA for all processors, is dumped. These 'PSW Areas' are printed with record id X'31'.

Note: The SUMDUMP output from a branch entry to SDUMP only contains areas that were already paged-in when the SUMDUMP was taken.

Started Task Control ABEND and Reason Codes

In case of an irreparable error, the started task control (STC) routines issue these ABEND codes:

0B8 - An error occurred while STC routines were processing a START, MOUNT, or LOGON command.

In each case, the command task is terminated; for a START or MOUNT command, the STC routines issue message IEE824I.

The following error codes can appear in register 15 at the time of the ABEND:

04 - Module IEEPRWI2 or IEFJSWT detected an invalid command code in the CSCB; the command code was incorrect for a START, MOUNT, or LOGON command.

08 - Module IEESB605 invoked IEFAB4FC (an Allocation routine) to build a TIOT for the START, MOUNT, or LOGON task; IEFAB4FC returned control to IEESB605 with a return code indicating failure.

12 - Module IEESB605 invoked IEFJSWT (an STC routine) to write the internal JCL text for the START, MOUNT, or LOGON command into system data set; IEFJSWT returned control to IEESB605 with a return code indicating that it failed in its attempt to open the data set.

16 - Module IEEPRW12 received an undefined return code from the system address space initialization routine. The defined codes are 0 and 4.

20 - Module IEEPRW12 requested a SYSEVENT TRANSWAP (via the POST macro) and received a nonzero completion code in the ECB. This indicates that the address space cannot be made nonswappable.

0B9 - Module IEESB605 invoked the master subsystem via the subsystem interface to determine whether a START command was issued to start a subsystem; an error occurred during master subsystem processing.

The command task is terminated; for a START or MOUNT command, IEESB605 issued message IEE824I.

OBA - Module IEESB605 invoked the master subsystem via the subsystem interface to determine whether a START command was issued to start a subsystem; an error occurred during subsystem interface processing.

The command task is terminated; for a START or MOUNT command, IEESB605 issues message IEE824I.

SWA Manager Reason Codes

In case of an irreparable error, the SWA manager routines issue a 0B0 ABEND. Before abending, both object modules IEFQB550 and IEFQB555 place a code in register 15 indicating the exact cause of the error.

These are the error codes that can appear in register 15.

04 - The routine that called SWA manager requested an invalid function.

08 - The routine that called SWA manager passed an invalid SWA virtual address (SVA). Either the SVA does not point to the beginning of a SWA prefix or the SWA prefix has been destroyed.

0C - A SWA manager routine has attempted to read a record not yet written into SWA.

10 - Either IEFQB550 (move mode module) has attempted to read or write a block which is not 176 bytes or IEFQB555 (locate mode module) has attempted to assign a block with a specified length of 0 or a negative number.

14 - The routine that called SWA manager has specified an invalid count field. For move mode, an invalid count is 0 for a READ, WRITE, or ASSIGN function; an invalid count for WRITE/ASSIGN is 00.

18 - The routine that called SWA manager by issuing the QMNGRIO macro instruction specified both or neither of the READ or WRITE options.

1C - The routine that called SWA manager was attempting to write into a SWA block for the first time; it either passed a nonexistent ID or failed to pass one at all.

20 - IEFQB555 has attempted to write a block using an invalid pointer to the block.

Additional Data Gathering Techniques

This chapter describes additional techniques for gathering data and circumventing certain system problems. The superzaps should be checked out before they are applied to your system. Displacements vary according to release level and PTF activity.

The examples were deliberately kept simple and are designed to illustrate a technique rather than to be practical in themselves.

CAUTION: Extreme care must be used when you are considering a system alternation in order to gather additional data about a problem. *None* of the Superzaps described in this chapter should be applied before the system programmer has verified the logic being zapped and the trap logic itself. Remember if any one location or offset within the module or trap changes, all offsets and base registers *must* be verified.

This chapter contains the following topics:

- Using the CHNGDUMP, DISPLAY DUMP, and DUMP Commands
- How to Print Dumps
- How to Automatically Establish System Options for SVC Dump
- How to Copy PRDMP Tapes
- How to Rebuild SYS1.UADS
- How to Print SYS1.DUMPxx
- How to Clear SYS1.DUMPxx Without Printing
- How to Print the SYS1.COMWRITE Data Set
- How to Print an LMOD Map of a Module
- How to Re-create SYS1.STGINDEX
- Software LOGREC Recording
- Using the PSA as a Patch Area
- Using the SLIP Command
- System Stop Routine
- How to Expand the Trace Table

Using the CHNGDUMP, DISPLAY DUMP and DUMP Operator Commands

A dump obtained from MVS contains those storage areas specified in the dump request and those defined as system defaults in SYS1.PARMLIB for SYSABEND, SYSMDUMP, and SYSUDUMP. Normal system defaults are:

SYSABEND:	CB, ENQ, TRT, ALLPA, SPLS, LSQA, PSW, REGS, SA, DM, IO, and ERR
SYSMDUMP:	LSQA, NUC, RGN, SQA, SWA, and TRT
SYSUDUMP:	CB, ENQ, TRT, ALLPA, SPLS, PSW, REGS, SA, DM, IO, and ERR

For an SVC dump, the normal system defaults are SQA, ALLPSA, and SUMDUMP.

The **CHNGDUMP command** is used to dynamically alter the options specified originally by SYS1.PARMLIB or by previous CHNGDUMP commands. Dump mode may be set to ADD, OVER, or NODUMP. System action for each setting is:

ADD - merges the options specified on the dump request with the options in the system dump options list.

OVER - ignores the options specified in the dump request and uses only the options in the dump options list.

NODUMP - ignores the request and does not dump.

To determine the current system dump options, use the DISPLAY DUMP, OPTIONS command. If an error is made while specifying the CHNGDUMP command, the system rejects the command and issues an error message.

The topic "How to Automatically Establish System Options For SVC Dump," which appears later in this chapter, describes how to issue the CHNGDUMP command during IPL. See *Operator's Library: System Commands* for the format of the CHNGDUMP command.

The **DISPLAY DUMP command** is used for the following:

- To display the effects of the CHNGDUMP command or to determine the current system dump options. (DISPLAY DUMP,OPTIONS)

- To determine which dump data sets are full and which are available. (DISPLAY DUMP,STATUS)

See *Operator's Library: System Commands* for the format of the DISPLAY DUMP command.

The **DUMP command** must be used carefully if the desired dump is to be obtained. For instance, the following typical error can occur when requesting a dump. The operator enters DUMP COMM=(title). The system responds with message IEE094 requesting the dump parameters. If the operator replies 'U' to this message, the system dumps the current address space which is the master scheduler address space. The operator must reply with ASID, Jobname, or TSOname. See *Operator's Library: System Commands* for the format of the DUMP command.

How to Print Dumps

The PRDMP control statements can be used to minimize the size of the output produced from a stand-alone dump and still keep the number of reruns to a minimum. This section discusses the DD statements and selected control statements used in the following example:

```
//ASIDDMP JOB MSGLEVEL=1
// EXEC PGM=AMDPRDMP
//PRINTER DD SYSOUT=A
//SYSPRINT DD SYSOUT=A
//TAPE DD UNIT=TAPE,LABEL=(1,NL),VOL=SER=ABCTPE,DISP=OLD
//SYSUT1 DD UNIT=251,SPACE=(CYL,(20,1)),DISP=NEW
//* PRINT STORAGE=ASID(X)=(X,X,X,X,X,X) IS PROPER FORMAT
   CVTMAP
   CPUDATA
   SUMMARY
   QCBTRACE
   SUMDUMP
   LPAMAP
   FORMAT
   EDIT
   PRINT CURRENT,SQA
   PRINT STORAGE=ASID(X)=(xxxx,xxxx,xxxx,xxxx)
   PRINT JOBNAME=(jobnames)
   PRINT REAL=(xxxx,xxxx)
   ASMDATA
   END
```

See *SPL: Service Aids* for a complete description of PRDMP DD and control statements.

The PRINTER DD statement defines the output data set for the dump itself. It should be directed to a SYSOUT class as shown.

The SYSPRINT DD statement defines the data set for PRDMP messages, etc.

The TAPE DD statement defines the input data set to PRDMP. It can define one of the SYS1.DUMPxx data sets, a stand-alone dump tape, or a GTF output data set on either tape or DASD.

The SYSUT1 DD statement defines work space to PRDMP. It can be used to define the input data set. It is not required if the input data set is defined by the TAPE DD statement. It does, however, significantly enhance the performance of PRDMP when it is used in conjunction with the TAPE DD statement and when the input is a tape data set.

The SPACE parameter is determined by the size of the dump. Generally 5 cylinders or 95 tracks or 285 4104 records should be specified for each megabyte of real storage dumped by SADMP.

Control Statements

The placement of the control statements determines the sequence in which the dump is printed. Refer to the "Dump and Trace Formats" section of the *Debugging Handbook* for examples of how these statements format a dump.

Note: To reduce the volume of output, select only those PRDMP control statements that provide the desired control blocks and output.

The following statements can be specified with PRDMP:

CVTMAP - formats the CVT and can be an aid in finding other significant control blocks in the system.

CPUDATA - formats the CSD, PSA, PCCA and LCCA for each active processor.

SUMMARY - defines and prints the dump ranges of the dump, active processor, active tasks, etc.

QCBTRACE or GRSTRACE - formats the ENQ/DEQ control blocks in use at the time the dump was taken.

SUMDUMP - locates and prints the summary dump data provided by SVC dump. It should be used on all SVC dumps.

LPAMAP - provides a listing of the modules on the link pack area list. It identifies the entry point address of those modules and their length. It does not identify SVC modules since they are found by the SVC table.

The FORMAT statement can produce voluminous data depending on the number of address spaces defined at the time the dump is taken. It produces the formatted TCB summary showing the abend completion codes for each TCB in the system and the global and local SPLs.

The EDIT statement formats and prints the GTF buffers (that is, all internal trace buffers or those external trace buffers that have not been written to the TRACE data set) if GTF is active at the time the dump is taken. If GTF is not active, only an error message is printed.

The PRINT statement can be used several ways:

- PRINT CURRENT,SQA - should be included in the initial run of PRDMP. It formats and prints the address space and task-related control blocks of the address space active at the time the dump is taken. SQA should be printed for the valuable data it contains such as trace table, and LOGREC buffers. PRINT CURRENT prints only the current address space of the processor from which the SADMP program was IPLed; except in cross memory mode, PRINT CURRENT also includes the home, secondary, primary, and CML address spaces.

- PRINT NUC,CSA - should not be included in the initial run of PRDMP because of the volume of data it produces. Once a problem is suspected in this area, the PRDMP program should be rerun specifying only these parameters.

- PRINT STORAGE=ASID(x)=(xxxx,xxxx) - should not be included in the initial run of PRDMP. Once a problem is isolated to an address space or a range of storage addresses, rerun PRDMP specifying only these parameters. Several ASIDs and several address ranges can be requested with one run of

PRDMP. PRDMP does not duplicate address ranges for every ASID but prints all storage dumped (NUC, CSA, SWA, LPA in storage) if only ASIDs are specified without address ranges. PRINT STORAGE is useful for printing SVC dumps. See the discussion "How to Print SYS1.DUMPxx" later in this chapter.

- PRINT JOBNAME=(jobnames) - produces output equivalent to PRINT CURRENT except it prints the private address space of job(s) requested. It should not be used for the initial run of PRDMP unless the jobname is known from another source, such as the system log.
- PRINT REAL=(xxxx,xxxx) - prints real storage in specified address range pairs. Use this option only when the system cannot find adequate data to format the dump.

ASMDATA - formats and prints all ASM control blocks. It produces voluminous data and should not be run until an ASM failure is suspected.

How to Automatically Establish System Options For SVC Dump

A potential problem is that the SVC dumps written to the SYS1.DUMPxx contains only those address ranges that the FRR or ESTAE routine passes to SDUMP. When these dumps are subsequently printed by PRDMP, the PRDMP formatting program might not find sufficient data to format the dump properly. This can make it difficult to find data in an SVC dump and it can provide erroneous indicators to the problem solver.

The CHNGDUMP command can be used to alter the SVC dump system options and provide a complete dump. The following job updates the COMMND00 member of SYS1.PARMLIB to issue the CHNGDUMP command automatically at IPL time. The CHNGDUMP command can also be entered by the operator. (See *Operator's Library: System Commands* for a description of the CHNGDUMP command.)

```
//UPDAT JOB (,,5,5),MSGLEVEL=1,REGION=100K
// EXEC PGM=IEBUPDTE
//SYSPRINT DD SYSOUT=A
//SYSUT1 DD UNIT=SYSDA,VOL=SER=SYSRES,DISP=OLD,
//   DSN=SYS1.PARMLIB
//SYSUT2 DD UNIT=SYSDA,VOL=SER=SYSRES,DISP=OLD,
//   DSN=SYS1.PARMLIB
//SYSIN  DD DATA
./ REPL NAME=COMMND00,LIST=ALL
./ NUMBER NEW1=10,INCR=20
COM='TRACE ON'
COM='CD SET,SDUMP=(PSA,NUC,SQA,LSQA,RGN,TRT),Q=YES,ADD'
./ ENDUP
```

How to Copy PRDMP Tapes

It is sometimes necessary to copy dump tapes to supply another location with a copy of the dump while retaining your own. It is particularly useful to be able to supply a dump tape with an APAR.

A simple way to do this is to use PRDMP as a copy program. Define the input tape with the TAPE DD statement and the output tape with the SYSUT2 DD statement. It is also possible to put several dumps on one tape or take one dump from a multiple dump tape by manipulating the file number parameters in the label parameter. The following example shows how this is done:

```
//ASIDDMP JOB MSGLEVEL=1
//  EXEC PGM=AMDPRDMP
//PRINTER DD SYSOUT=A
//SYSPRINT DD SYSOUT=A
//TAPE DD UNIT=TAPE,LABEL=(2,NL),VOL=SER=DMPIN,DISP=OLD
//SYSUT2 DD UNIT=TAPE,LABEL=(,NL),VOL=SER=DMPOUT,
//   DISP=(NEW,KEEP)
//SYSIN DD *
  END
/*
```

After copying a PRDMP tape, a quick run through PRDMP to verify that the CVT can be formatted and printed will prove that the copy was successful.

```
//ADMP JOB MSGLEVEL=1
//  EXEC PGM=AMDPRDMP
//PRINTER DD SYSOUT=A
//SYSPRINT DD SYSOUT=A
//TAPE DD UNIT=TAPE,LABEL=(1,NL),VOL=SER=DMPTPE,DISP=OLD
//SYSUT1 DD UNIT=SYSDA,SPACE=(TRK,(400,20)),DISP=NEW
//SYSIN DD *
  CVTMAP
 END
/*
```

Another, and faster way to copy PRDMP tapes is to use the IEBGENER utility program. The following example shows how this is done:

```
//COPYDMP JOB MSGLEVEL=1
// EXEC PGM=IEBGENER
//SYSIN DD DUMMY
//SYSPRINT DD SYSOUT=A
//SYSUT1 DD UNIT=TAPE,LABEL=(2,NL),VOL=SER=DMPIN,
//   DCB=(RECFM=FB,LRECL=4104,BLKSIZE=4104)
//SYSUT2 DD UNIT=TAPE,LABEL=(1,NL),VOL=SER=DMPOUT,
//   DCB=(RECFM=FB,LRECL=4104,BLKSIZE=4104)
/*
```

How to Rebuild SYS1.UADS

The loss of the SYS1.UADS data set can significantly impact a TSO environment. However, it is possible to run the TMP as a batch job and recreate SYS1.UADS in the background. The following is an example of a job that has been run successfully to scratch and recreate a SYS1.UADS data set.

```
//BLDUADS JOB MSGLEVEL=1
// EXEC PGM=IEFBR14
//DD2 DD VOL=SER=SYSRES,DISP=(OLD,DELETE),UNIT=3330,
//  DSN=SYS1.UADS
//  EXEC PGM=IKJEFT01
//SYSPRINT DD SYSOUT=A
//SYSUADS DD DSN=SYS1.UADS,DISP=(NEW,KEEP),
//   SPACE=(800,(20,9,30)),UNIT=3330,
//   VOL=SER=SYSRES,DCB=(RECFM=FB,
//   DSORG=PO,LRECL=80,BLKSIZE=800)
//SYSLBC DD DSN=SYS1.BRODCAST,DISP=SHR
//SYSIN DD *
ACCOUNT
SYNC
ADD (USER01 TSOTE01 * IKJACC01) UNIT(SYSDA) ACCT OPER JCL MOUNT
ADD (USER02 TSOTE02 * IKJACC02) UNIT(SYSDA) OPER JCL MOUNT
ADD (USER03 TSOTE03 * IKJACC03) UNIT(SYSDA) JCL MOUNT
ADD (USER04 TSOTE04 * IKJACC04) UNIT(SYSDA) JCL MOUNT
ADD (USER05 TSOTE05 * IKJACC05) UNIT(SYSDA) JCL MOUNT
ADD (USER06 TSOTE06 * IKJACC06) UNIT(SYSDA) JCL MOUNT
ADD (USER07 TSOTE07 * IKJACC07) UNIT(SYSDA) JCL
ADD (USER08 TSOTE08 * IKJACC08) UNIT(SYSDA) JCL
ADD (USER09 TSOTE09 * IKJACC09) UNIT(SYSDA) OPER
ADD (USER0A TSOTE0A * IKJACC0A) UNIT(SYSDA)
ADD (USER0B TSOTE0B * IKJACC0B) UNIT(SYSDA)
ADD (USER0C TSOTE0C * IKJACC0C) UNIT(SYSDA)
LIST (*)
END
/*
```

How to Print SYS1.DUMPxx

See the discussion under "How to Print Dumps" earlier in this chapter to define the control statements required. The same rules apply except in this case the TAPE DD statement points to one of the SYS1.DUMPxx data sets. These are cataloged data sets and require no further definition.

Be aware that the dump data sets contain only those address ranges passed to SVC dump by the dump requestor and might not contain sufficient data for PRDMP to properly format all requested control blocks.

Because SVC dumps usually contain a limited number of address ranges, printing the entire SYS1.DUMPxx data set is feasible and assures that all the information about the problem will be available.

See the next topic "How to Clear SYS1.DUMPxx Without Printing" for a description of how to clear the dump data sets for reuse. **Note:** Printing the dump data sets does not clear them as it did on previous systems.

The following example shows how to print SYS1.DUMP00:

```
//ASIDDMP JOB MSGLEVEL=1
//   EXEC PGM=AMDPRDMP
//PRINTER DD SYSOUT=A
//SYSPRINT DD SYSOUT=A
//TAPE DD DSN=SYS1.DUMP00,DISP=OLD
//SYSUT1 DD UNIT=SYSDA,DISP=NEW,SPACE=(CYL,(10,5))
//SYSIN DD *
  SUMMARY
  CVTMAP
  CPUDATA
  SUMDUMP
  LPAMAP
  PRINT STORAGE
/*
```

How to Clear SYS1.DUMPxx Without Printing

In previous systems, printing the dump data set also cleared it and made it available for reuse. In MVS this is no longer true. The dump data sets can be cleared at 'SPECIFY SYSTEM PARAMETERS' time during IPL. They can also be cleared and made available for reuse by using PRDMP to copy the data set to tape with the SYSUT2 DD statement pointing to the output data set. This must be a separate job step from printing the dump. If it has been determined that the SYS1.DUMPxx data set need not be saved, it can be cleared and made available for reuse by running PRDMP with the SYSUT2 DD statement defined as DUMMY. The following example shows how to clear SYS1.DUMP00. See the example in the discussion "How to Copy PRDMP Tapes" earlier in this chapter for how to define the SYSUT2 DD statement to unload the SYS1.DUMPxx data sets.

```
//ASIDDMP JOB MSGLEVEL=1
//   EXEC PGM=AMDPRDMP
//PRINTER DD SYSOUT=A
//SYSPRINT DD SYSOUT=A
//TAPE DD DSN=SYS1.DUMP00,DISP=OLD
//SYSUT2 DD DUMMY
//SYSIN DD *
  END
```

Another, and faster way to clear a SYS1.DUMPxx data set without printing is to use the IEBGENER utility program. The following example shows how to clear SYS1.DUMP00:

```
//CLEARDMP JOB MSGLEVEL=1
//      EXEC PGM=IEBGENER
//SYSIN DD DUMMY
//SYSPRINT DD SYSOUT=A
//SYSUT1 DD DUMMY,DCB=SYS1.DUMP00
//SYSUT2 DD DSN=SYS1.DUMP00,DISP=OLD,DCB=SYS1.DUMP00
/*
```

How to Print the SYS1.COMWRITE Data Set

The following job will format and print the TCAM SYS1.COMWRITE data set. Note that the PARM fields in the EXEC statement define the traces to be formatted and printed. See *OS/VS TCAM Debugging Guide Level 10* for more information on the use of the SYS1.COMWRITE data set.

```
//COMWRITE JOB MSGLEVEL=1
//STEP1 EXEC PGM=IEDQXB,PARM='STCB,IOTR,BUFF'
//SYSPRINT DD SYSOUT=A
//SYSUT1 DD DSN=SYS1.COMWRITE,DISP=SHR
/*
```

How to Print an LMOD Map of a Module

The following job produces a module cross-reference of the nucleus, module IEFW21SD, and a link pack area map. In addition, AMBLIST produces an IDR listing or a complete hexadecimal dump of an object module. If you include the RELOC parameter, the cross-reference listing is based at the address the module is loaded in LPA.

Note that the JCL must contain a DD statement for every data set containing a module you referenced in the control card section.

For more information about AMBLIST, see *SPL: Service Aids.*

```
//AMBLIST JOB MSGLEVEL=1
//  EXEC PGM=AMBLIST
//SYSLIB DD DSN=SYS1.LPALIB,DISP=OLD
//LOADLIB DD DSN=SYS1.NUCLEUS,DISP=OLD
//SYSPRINT DD SYSOUT=A
//SYSIN DD *
  LISTLOAD OUTPUT=XREF,MEMBER=IEANUC01,DDN=LOADLIB
  LISTLPA
  LISTLOAD OUTPUT=XREF,MEMBER=IEFW21SD
/*
```

How to Re-Create SYS1.STGINDEX

It is possible for the SYS1.STGINDEX data set to be destroyed because of system failure or operator intervention during an IPL with the cold start (CLPA,CVIO) option. Loss of this data set prevents warmstarting the system or restarting jobs using VIO data sets.

The following job can recreate this data set. Remember to change the VOLUME and CYLINDERS parameters to apply to your system.

```
//STGINDEX JOB MSGLEVEL=1
// EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=A
//VOL DD DISP=OLD,UNIT=3330,VOL=SER=SYSRES
//SYSIN DD *
 DEFINE SPACE(VOL(SYSRES)FILE(VOL)CYL(7))
 DEFINE CLUSTER-
 (NAME(SYS1.STGINDEX)-
 VOLUME(SYSRES)-
 CYLINDERS(7)-
 KEYS(128)-
 BUFFERSPACE(5120)-
 RECORDSIZE(2041 2041)-
 REUSE)-
 DATA-
 (CONTROLINTERVALSIZE(2048))-
 INDEX-
 (CONTROLINTERVALSIZE(1024))
```

Software LOGREC Recording

The following JCL defines a two-step job. The first step prints an event history report for all SYS1.LOGREC records. The second step formats each software, IPL, and EOD record individually. The event history report is printed as a result of the EVENT = Y parameter on the EXEC statement of the first step. It can be a very useful tool to the problem solver because it prints the records in the same sequence they were recorded and therefore shows an interaction between hardware error records and software error records.

```
//EREP JOB MSGLEVEL=1
//EREPA EXEC PGM=IFCEREP1,PARM='EVENT=Y,ACC=N',
//    REGION=128K
//SERLOG DD DSN=SYS1.LOGREC,DISP=SHR
//TOURIST DD SYSOUT=A
//EREPPT DD SYSOUT=A,DCB=BLKSIZE=133
//EREPB EXEC PGM=IFCEREP1,PARM='TYPE=SIE,PRINT=PS,ACC=N',
//    REGION=128K
//SERLOG DD DSN=SYS1.LOGREC,DISP=SHR
//TOURIST DD SYSOUT=A
//EREPPT DD SYSOUT=A,DCB=BLKSIZE=133
/*
```

See the discussion on LOGREC analysis in the "Use of Recovery Work Areas" chapter earlier in this section for an explanation of its use and for examples of the output produced.

Using the PSA as a Patch Area

There are areas in the PSA reserved for future expansion. They can be used for quick implementation of a trap without having to consider base registers. Check the mapping of the PSA (IHAPSA) for possible areas to be used. Once an area is chosen, verify that the value of this storage is zero.

CAUTION: Use extreme care when you use this method. Patches should be made only to disabled code unless the patch is completely reentrant. Saving registers and data in the PSA while the system is enabled could produce unpredictable results, especially in an MP environment where more than one PSA exists and the code could be interrupted and subsequently redispatched on the other processor. Extreme care must be used when considering a system alteration in order to gather additional data about a problem. No superzaps should be applied before the system programmer has verified the logic being zapped and the trap logic itself. Remember, if any one location or offset within the module or trap changes, all offsets and base registers *must* be verified.

Using the SLIP Command

SLIP (serviceability level indication processing) provides a way of getting information concerning an error prior to ESTAE or FRR recovery processing. This is in addition to the information ordinarily supplied by dumping services during abnormal termination. The SLIP command is also used to establish PER monitoring for instruction fetch, storage alteration, and successful branch PER events within a range of virtual addresses. When the requested PER event occurs, a PER interrupt causes control to be given to the SLIP processor. The purpose of the SLIP command is to establish SLIP traps that describe the system conditions which must exist at the time of the error or interrupt so that an action will be taken.

The SLIP command is usually entered by a system programmer, either at the console or via the input stream. It can also reside in the COMMNDxx PARMLIB member. The SLIP command can also be entered as a subcommand of the OPERATOR command from a TSO terminal or TSO CLIST. Information about SLIP traps can be displayed by using the DISPLAY operator command.

As long as enough system queue area storage is available, SLIP traps may be established at any time. The recovery termination manager (RTM) compares the SLIP trap event qualifiers with the dynamic system conditions at the time of the error or interrupt. If RTM detects a match, the requested action is taken.

SLIP Event Qualifier Keywords

When specified on a trap, an event qualifier keyword is checked against current system conditions to determine a match or no match condition. The following keywords are described in this topic.

ADDRESS
ASID
ASIDSA
COMP
DATA
ERRTYP
JOBNAME
JSPGM
LPAMOD
MODE
PVTMOD
RANGE
RBLEVEL

Because the RTM/SLIP processor runs in one of four environments, the checking done for each event qualifier keyword is described in terms of the applicable environment. The four environments are:

RTS an error has occurred that will cause routing to FRRs.

RT2 an error has occurred while in enabled unlocked task mode which will cause routing to ESTAEs but not FRRs.

RTM an abnormal address space termination has occurred.

PER a PER interrupt has occurred.

Note: The names of these four environments are taken from the names of the modules that call the RTM/SLIP processor; namely IEAVTRTS, IEAVTRT2, IEAVTRTM, and IEAVTPER.

ADDRESS Event Qualifier Keyword

The checking done for the ADDRESS keyword is:

RTS the address from the PSW at the time of the error (SDWANXT1) is used in the comparison.

RT2 the RBLEVEL keyword determines which RB is used to get the address (RBPSWNXT) used in the comparison.

RTM the address of the instruction that branched to RTM is used in the comparison.

PER the address of the instruction that caused the PER interrupt (LCCAPERA) is used in the comparison.

ASID Event Qualifier Keyword

The checking done for the ASID keyword is:

RTS The PASID of the failing address space (field SDWAPRIM) is used in the comparison.

RT2 The PASID in the XSB of the RB located from the RBLEVEL keyword (field XSBPASID) is used in the comparison.

RTM The PASID of the address space being terminated is used in the comparison.

PER The PASID at the time of the interruption is used in the comparison.

ASIDSA Event Qualifier Keyword

The ASIDSA keyword associates an address space with a storage alteration (SA). ASIDSA is only valid when it is specified with SA on the SLIP command. The PASID, SASID, HASID, and the S-bit at the time of the interruption are used to determine the address space where the storage alteration occurred.

COMP Event Qualifier Keyword

The checking done for the COMP keyword is:

RTS field SDWACMPC is used in the comparison.

RT2 field RTM2CC is used in the comparison.

RTM the completion code for the address space being terminated (ASCBMCC) is used in the comparison.

Because many recovery routines change the abend completion code to make it more specific, the value supplied on the COMP keyword must be the original value before a recovery routine changes the code. This is because RTM/SLIP checking is done before processing by the recovery routines.

For example, a SLIP trap will not match if any of the following completion codes are specified: 11A, 12E, 15D, 15F (reason codes 12 and 16), 200, 212, 25F, 279, 282, 402, 42A, 57D, 6FC, 700, 72A, A00, B00, and E00. Most of these codes were originally a program check (0C4) that has been converted to a more specific value. If you want to specify a program check, use COMP=0C4 or ERRTYP=PROG. To avoid having the SLIP action occur for all program checks, you should also specify some other event qualifier such as program name or module name.

Similarly, specification of 13E or 33E might prevent a trap from matching if these completion codes occur for any active subtasks associated with a task that is abending. These secondary abends occur for the purpose of clean-up and therefore the SLIP processor is not called when they occur.

Note: The SDUMP and ABDUMP dumping programs might cause many 0C4 program checks while taking a dump. Therefore, when you specify COMP=0C4 on a trap, you should avoid these unwanted matches by also specifying another event qualifier, such as MATCHLIM.

DATA Event Qualifier Keyword

The DATA keyword checks data in the system at the time of the error or interrupt against the data conditions specified in the SLIP trap. Addressing is established to the address space specified with the address. Indirect addresses are resolved by using the registers at the time of the error or interrupt.

For some errors, register contents at the time of error are not valid. In such a case, if registers are used, the data is considered unavailable and the trap does not match. Other conditions that can cause the data unavailable situation are: the address space specified with the address does not exist; data is paged out; or a pointer required for an indirect address is paged out.

When data is unavailable, a counter associated with the trap is incremented and message IEA413I is sent to the SLIP user. For a PER trap, message IEA413I is issued only the first time that the data unavailable situation occurs. However, the counter is made available by displaying the trap. The data unavailable counter is also part of the SLIP standard, SLIP standard/user, and SLIP DEBUG GTF trace records.

The SLIP command processor does not perform any reasonability checking on the data tests requested. For example, the SLIP command processor would allow the following DATA keyword specification even though its specification prevents the trap from ever matching.

```
DATA=(H.CD300,EQ,00,HASID.CD300,NE,00)
```

The DATA keyword may be used as a validity check for RANGE addresses or LPAMOD offsets when used on an IF (instruction fetch) or SB (successful branch) PER trap. For example, if RANGE=(CD300,CD303) is used to establish the range of addresses for an IF trap, you can ensure that the expected instruction is being monitored by specifying DATA=(CD300,EQ,47F0B020) where 47F0B020 is the expected instruction. If the wrong instruction is being monitored (for example, due to a typing error or a change in the system), the trap would not match because the DATA keyword does not match. This technique can be especially useful on traps that take potentially disruptive actions (for example, WAIT or RECOVERY actions) in order to ensure the action is taken only when desired.

ERRTYP Event Qualifier Keyword

The checking done for the ERRTYP keyword is:

RTS the RT1TENPT field is used in the comparison. The value field RT1TENPT of the RTM1 work area indicates the reason for entry into RTM1:

1=PROG	3=SVCERR	5=MACH
2=REST	4=DAT	10=PGIO

The SLIP processor recognizes an SVC error for SVC 13 as either an ABEND or SVCERR error and allows a match for the ERRTYP keyword if either is specified.

RT2 the RTM2ERRA field is used in the comparison. The reason for entry into RTM2 is indicated by flags in the RTM2 work area as follows:

RTM2MCHK=MACH	RTM2ABTM=ABEND
RTM2PCHK=PROG	RTM2TEXC=DAT
RTM2RKEY=RESTART	RTM2PGIO=PGIO
RTM2SVCD=ABEND	

RTM an abnormal address space termination (MEMTERM) causes a match.

JOBNAME Event Qualifier Keyword

The checking done for the JOBNAME keyword is:

RTS if the failing address space has been identified by RTM (field SDWAFMID), both the failing and current address space job names are tested for a match. The job names pointed to by fields ASCBJBNI and ASCBJBNS are tested and either job name may match the job name specified in the trap.

if the failing address space has not been identified, then only the current address space is tested for a job name match.

RT2 if the failing address space has been identified by RTM (field RTM2FMID), both the failing and current address space job names are tested for a match. The job names pointed to by fields ASCBJBNI and ASCBJBNS are tested and either job name may match the job name specified in the trap.

if the failing address space has not been identified, then only the current address space is tested for a job name match.

RTM the job names for the address space being terminated are used in the comparison.

PER the job names for the current address space are used in the comparison.

Note: The job, logon, or started task named by JOBNAME need not be active when the trap is set.

JSPGM Event Qualifier Keyword

The checking done for the JSPGM keyword is:

RTS if a job step program name is available (field JSCBPGMN), it is compared to the job step program name specified in the trap.

if a job step program name is not available (PSATOLD or TCBJSCBB=0), the trap will not match.

if the reason for entry is a DAT error, the trap will not match.

RT2 if a job step program name is available (field JSCBPGMN), it is compared to the job step program name specified in the trap.

if a job step program name is not available (PSATOLD or TCBJSCBB = 0), the trap will not match.

RTM the trap will not match because the job step program name in the address space being terminated is not available.

PER if a job step program name is available (field JSCBPGMN), it is compared to the job step program name specified in the trap.

if a job step program name is not available (PSATOLD or TCBJSCBB = 0), the trap will not match.

LPAMOD Event Qualifier Keyword

The checking done for the LPAMOD keyword is:

RTS the address from the PSW at the time of error (field SDWANXT1) is used in the comparison.

RT2 the RBLEVEL keyword determines which RB is used to get the address (field RBPSWNXT) to be used in the comparison.

RTM the address of the instruction that branched to RTM is used in the comparison.

PER the address of the instruction that caused the PER interrupt (field LCCAPERA) is used in the comparison. (For additional information, refer to the note under the description of the RANGE keyword.)

Note: If the name specified on the LPAMOD keyword is an alias of a load module name, all monitoring is done by SLIP as though the load module name was specified.

MODE Event Qualifier Keyword

The system mode at the time of error is indicated in the MODEBYTE as follows:

1...	MODESUPR	Supervisor control
.1..	MODEDIS	Physically disabled
..1.	MODEGSPN	Global spin lock held
...1	MODEGSUS	Global suspend lock held
.... 1...	MODELOC	Locally locked
.... .1..	MODETYP1	Type 1 SVC
.... ..1.	MODESRB	SRB mode
....1	MODETCB	Task mode (unlocked)

The checking done for the MODE keyword is:

RTS as a part of error processing, RTM determines the mode of the system (MODEBYTE value in field RT1WMODE). Also, the PSW at the time of the error (field SDWAEC1) is examined to determine key and state. The SDWASTAF bit indicates if the error occurred while a recovery routine was in control. The PASID at the time of the error (SDWAPRIM) is compared to the HASID. If they are equal, the instruction executed in home mode.

RT2 the system mode, as determined by RTM, is obtained from the ABEND SVRB extended save area (MODEBYTE value in field ESAMODE). Also, the PSW (field RBOPSW) in the RB (as determined by RBLEVEL processing) is examined for key and state. The RTM2RECR and RTM2XIP bits indicate if a recovery routine was in control at the time of error. The PASID at the time of the error (obtained by the RBLEVEL keyword, field XSBPASID) is compared to the HASID. If they are equal, the instruction executed in home mode.

RTM because RTM has not determined the system mode for an address space termination, various fields are tested by the RTM/SLIP processor to determine the system mode at the time of the MEMTERM request. (The mode will always indicate supervisor state and system key because these are requirements for issuing a MEMTERM request.)

PER various fields are tested to determine the system mode at the time of the interrupt. In the SLIP trace record (system mode indicators), all bits are filled in. However, no attempt is made to determine if a recovery routine was in control when the interrupt occurred. Therefore, the recovery-routine-in-control bit will always be zero for a PER interrupt. Because of this, MODE=RECV is invalid for a PER trap and if MODE=ALL is specified for a PER trap, ALL does not include RECV (recovery-routine-in- control). If the PASID and HASID are equal at the time of the interruption, the home mode indicator is set.

PVTMOD Event Qualifier Keyword

The checking done for the PVTMOD keyword is:

RTS if RTM has identified a failing address space (field SDWAFMID) and it is not current, the trap will not match.

to check for a private area module, the local lock must be obtained. If it cannot be obtained, the trap will not match. If the local lock is already held, the chain that is to be searched for a private area module may be in the process of being changed. The search is performed, but the results may not be valid. The address obtained from the PSW at the time of error (field SDWANXT1) is used in the comparison.

RT2 the RBLEVEL keyword determines which RB is used to get the address (field RBPSWNXT) used in the comparison.

RTM this keyword test will not match because the private area chain in the failing address space is not available for searching.

PER the trap will not match for the PVTMOD keyword test if the interrupt occurs in the nucleus or FLPA because these areas cannot contain private area modules.

to check for a private area module, the local lock must be obtained. If it cannot be obtained, the trap will not match. If the local lock is already held, the chain that is to be searched for a private area module may be in the process of being changed. The search is performed, but the results may not be valid. The address of the instruction that caused the PER interrupt (field LCCAPERA) is used in the comparison.

If offsets are specified on the PVTMOD keyword, the RTM/SLIP processor does not check to make sure that the offsets define an area wholly within the private area module.

RANGE Event Qualifier Keyword

The checking done for the RANGE keyword is:

PER the address of the instruction that caused the PER interrupt (field LCCAPERA) is used in the comparison. Note that if the first address specified is greater than the second, the monitored range wraps storage addresses.

Note: For successful branch monitoring, hardware PER processing does not check the address range specified on the RANGE and LPAMOD keywords. This means that a branch taken by an instruction anywhere in the system would cause a successful branch PER interrupt. However, to simulate an address range for successful branch monitoring, SLIP initially sets up instruction fetch monitoring for the desired address range. Then when instruction processing enters the requested range (indicated by an instruction fetch PER interrupt), PER monitoring is automatically switched to successful branch mode. You should be aware that the first PER event that occurs when instruction processing enters the requested range may not be a successful branch event. This "extra" event (instruction fetch) may affect values supplied for other keywords such as MATCHLIM. When instruction processing leaves the requested range, PER monitoring returns to instruction fetch monitoring on the requested range to avoid unnecessary PER interrupts. If the instructions being monitored are enabled for I/O and/or external interrupts, control may leave and then re-enter the monitored range due to normal interrupt processing.

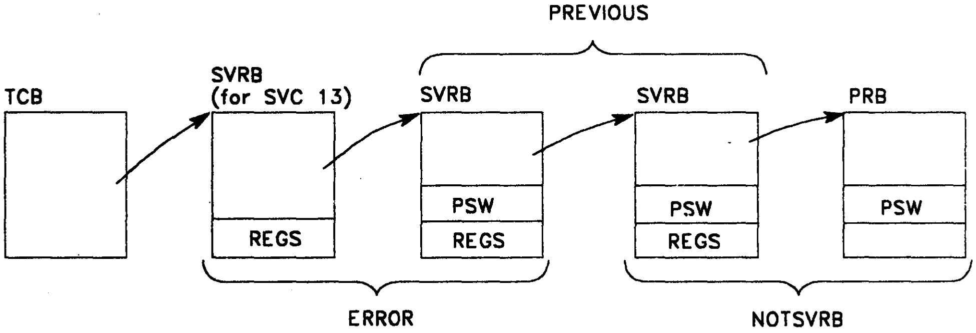
The previous note applies to processing on behalf of a non-IGNORE successful branch PER trap. Mode switching does not occur for successful branch PER traps with ACTION=IGNORE specified. This means that if the initial entry into a monitored area matches an IGNORE trap, the mode remains instruction fetch and the "extra" event is delayed. Also, output that appears to be a contiguous successful branch trace may not actually be contiguous if an IGNORE trap matches intermittently.

For successful branch monitoring, if an EXECUTE instruction has a successful branch target, the location of the EXECUTE instruction is used to determine whether or not the branch was within the monitored area without regard to the location of the executed branch.

RBLEVEL Event Qualifier Keyword

The RBLEVEL keyword applies only to enabled unlocked task mode errors. It is used to direct the SLIP processor to the registers and PSW of interest for a particular error. The SLIP processor will use the PSW identified by the RBLEVEL keyword when processing the LPAMOD, PVTMOD, ADDRESS, and MODE keywords. The SLIP processor will use the registers identified by the RBLEVEL keyword when processing the DATA, TRDATA, LIST, and SUMLIST keywords.

The following diagram shows which RBs are chosen by the three RBLEVEL keyword options for an example RB chain.



The RBLEVEL keyword can be used in an error situation where several nested services are involved. For example, program A calls service B which calls service C. If an error occurs in service C, the default RBLEVEL=ERROR can be used to qualify the error or obtain information concerning the error. However, if the error in service C is the result of incorrect input supplied by service B or program A, the RBLEVEL=PREVIOUS or RBLEVEL=NOTSVRB can be used to qualify the error or obtain information concerning the input supplied by service B or program A respectively.

Using the ACTION Keyword

The ACTION keyword is used to specify the action to be taken when a SLIP trap matches the specified system conditions. The following ACTION options are described in this topic:

ACTION=SVCD	- schedule an SVC dump.
ACTION=WAIT	- put the system in a wait state.
ACTION=TRACE	- write a GTF trace record.
ACTION=TRDUMP	- write a GTF trace record and then schedule an SVC dump.
ACTION=NODUMP	- suppress dump requests.
ACTION=IGNORE	- take the IGNORE action.
ACTION Keyword	- with RECOVERY option (PER traps only), initiate recovery processing.

ACTION=SVCD Option

ACTION=SVCD indicates that an SVC dump will be scheduled for the failing or home ASID. This is the default option if ACTION is not specified. If the SVC dump cannot be taken, message IEA412I is issued and SLIP processing continues. No attempt is made to reschedule the SVC dump.

One of the advantages of the SVC dump over one taken by a recovery routine is that nothing has been done to correct the error situation. Although the bulk of the SVC dump is not taken until later, the summary dump portion preserves as much volatile data as possible. An SVC dump also contains more data (for example, more than one address space can be dumped) than a SYSABEND or SYSUDUMP, and because it is machine readable, it can, if necessary, be copied onto a tape to accompany an APAR, or used with interactive dump display programs. SYSMDUMP also provides a machine readable dump. The biggest advantage is in situations where no dump was occurring.

When ACTION=SVCD is specified or defaulted, the default SDATA parameters are: SQA, RGN, TRT, LPA, CSA, NUC, ALLPSA, and SUM. These default SDATA parameters are affected by the current CHNGDUMP command settings which may add to or override the requested dump options. The SDATA parameters can be changed by the SLIP user. Refer to "Dump Tailoring" later in this section.

If an SVC dump is already in progress, another dump cannot be taken and message IEA412I is issued. When an SVC dump is scheduled on behalf of a SLIP trap by the SLIP processor, debugging information is placed in the SDUMP 4K buffer (if the buffer is available). This buffer is pointed to by field CVTSDBF and contains:

Offset	Length	Content
0(0)	4	The characters 'TYPE' to identify the following field.
4(4)	4	RTM/SLIP processor environment indicator: X'00000001' - RTS X'00000002' - RT2 X'00000003' - RTM X'00000004' - PER
8(8)	4	The characters 'CPU' to identify the following field.
12(C)	4	Logical CPUID.
16(10)	4	The characters 'REGS' to identify the following field.
20(14)	64	Registers at the time of error or interrupt. (R0-R15)
84(54)	4	The characters 'PSW' to identify the following field.
88(58)	8	The PSW at the time of error or interrupt.
96(60)	4	The characters 'PASD' to identify the following field.
100(64)	2	The primary ASID at time of error or interruption.
102(66)	4	The characters 'SASD' to identify the following field.
106(6A)	2	The secondary ASID at time of error or interruption.
108(6C)	variable	The SDWA if offset 4 is 1 (RTS). The RTM2WA if offset 4 is 2 (RT2). The ASCB if offset 4 is 3 (RTM). The PER interrupt code if offset 4 is 4 (PER).

When a summary dump is requested, the storage on either side of the addresses in the registers used are from the SDUMP 4K buffer. You can use the SUMLIST keyword in the form: 0R%-800,+1000,1R%-800,+1000,2R%-800,+1000,... to dump 2K bytes of information on either side of the addresses in the registers at the time of the error or interrupt.

If ASIDLST is not specified with the SLIP trap, the following information describes which address space will be dumped depending on the environment of the RTM/SLIP processor.

RTS the failing address space (field SDWAFMID) or the home address space is dumped.

RT2 the failing address space (field RTM2FMID) or the home address space is dumped.

RTM the master address space is dumped because the address space that is terminating cannot be used to take the dump. The summary dump information is collected in the home address space (of the issuer of the CALLRTM TYPE=MEMTERM macro), and the asynchronous dump runs later in the master address space.

PER the home address space is dumped.

If a dump request for a failing address space fails (such as SDUMP returning a bad return code), then a second attempt is made to schedule a dump in the home address space. In the second attempt, no information is put in the SDUMP 4K buffer. If the second attempt fails, the message IEA412I is issued.

If a summary dump is requested, it may be suppressed under certain conditions. Refer to the topic "Placement of PER Traps" later in this section.

The entire dump may be suppressed if the operator has chosen the CHNGDUMP NODUMP option.

ACTION=WAIT Option

ACTION=WAIT indicates that the system will be placed in an 01B wait state.

If a PER trap is used to put the system into the wait state, the time spent in the wait state is attributed to the PER interrupt that caused the wait state. This makes it look as though a lot of time has been spent processing PER interrupts. Therefore, if the trap is intended to be used so that the system is restarted and the trap is to remain enabled, you may want to use PRCNTLIM=99 on the trap. Otherwise, the trap may be disabled after the system has been restarted. Use PRCNTLIM=99 with caution because limit checking is not performed while waiting for the trap to match.

Once in the wait state, you can use the debugging work area provided by SLIP to begin debugging a problem. This area is pointed to by PSA+X'40C' and contains:

Offset	Length	Content
0(0)	1	RTM/SLIP processor environment indicator: X'01' - RTS X'02' - RT2 X'03' - RTM X'04' - PER
1(1)	2	Logical CPUID.
3(3)	1	System mask (if offset 0 is 2).
4(4)	4	Pointer to registers at the time of error or interrupt. (R0-R15)
8(8)	4	Pointer to PSW at the time of error or interrupt.
12(C)	4	Pointer to SDWA if offset 0 is 1. Pointer to RTM2WA if offset 0 is 2. Pointer to ASCB being terminated if offset 0 is 3. Pointer to PER code if offset 0 is 4.
16(10)	4	Pointer to cross memory information (control registers 3 and 4) at the time of the error or interruption.

ACTION=TRACE Option

ACTION=TRACE indicates that a GTF SLIP trace record is written each time that the SLIP trap matches. GTF must be active and the GTF SLIP option specified in order for the record to be built and recorded. (Use the TRDATA keyword if you want to tailor the GTF trace records.)

The TRACE option is designed for those situations where a relatively small amount of data is required each time that a matching event occurs. Such a situation might occur when you are trying to determine the path through a module. But the TRACE option can handle a relatively large amount of data when required. Refer to the TRDATA keyword.

The registers at the time of the error or interrupt are used to resolve indirect addresses specified for the trace record fields. Under some circumstances, registers at the time of error may not be available. If this is the case, indirect addresses that contain a register value cannot be resolved and related fields cannot be collected. A zero length field is used in the user portion of a SLIP standard/user or SLIP user record to indicate that the requested field was not available. Also, a field is not available if it is paged out or if one of the pointers to it is paged out. When using indirect addresses, use the REGS keyword to get the contents of the registers used to resolve indirect addresses.

Checking for too much data is done at the time that the GTF SLIP trace record is built, not at the time the trap is entered. Therefore, you may want to exceed the trace record size when setting a trap if you expect that some of the data will not be available. If data is unavailable, trap information takes up only one byte in the record rather than the amount of space it would take if data were available. When using this technique, prioritize the fields so that the most important fields are earliest in the record so that they are collected. If all the data is available, and the maximum size of the trace record is exceeded, the record is truncated.

Another technique that is useful when using long indirect addresses is to request the same field twice if there is more than one path to the field. In this way, if a pointer is bad or is paged out in one path to the data, it may be available via the other path.

GTF Considerations: When using ACTION=TRACE, be aware that starting GTF will suppress the system trace (if it is active). Therefore, you may want to choose other GTF trace options in addition to SLIP to obtain other valuable diagnostic data that is available in a trace of system events. GTF options SYS or SYSM can be used to have GTF collect information similar to that collected by the normal system trace. Note that using the internal GTF trace instead of the external trace helps to reduce system overhead. Be sure to stop GTF after the traps which require the TRACE option are disabled or deleted.

ACTION=TRDUMP Option

ACTION=TRDUMP is a combination of the SVCD and TRACE options. While the trap remains enabled, a GTF SLIP trace record is written when the trap matches. When the trap is disabled (automatically by MATCHLIM or PRCNTLIM or via the SLIP MOD operand) or deleted (via the SLIP DEL operand), a dump is scheduled. When you use the SLIP MOD or DEL operand to disable or delete a trap that has the TRDUMP option specified, the dump does not contain diagnostic data in the SDUMP 4K buffer.

The default SDATA parameters are TRT, NOSQA, NOALLPSA, and NOSUM. These default are affected by the current CHNGDUMP command settings which may add to or override the requested dump options. The SDATA parameters can be changed by the SLIP user. Refer to "Dump Tailoring" later in this section.

When used in conjunction with the MATCHLIM keyword, the TRDUMP option can be useful in getting an idea of what events lead up to an error. For example, a problem is narrowed to a particular module. You could use a successful branch PER trap and the TRDUMP option. The trace records that are written could trace the fields that are critical to the operation of the module. An estimate of the number of successful branches that would enable you to determine the path through the module could be specified on the MATCHLIM keyword in order to automatically disable the trap and initiate the dump.

The TRDUMP option can also be used to obtain GTF SLIP trace records without tracing to an external data set (and then using PRDMP to print the data set). When starting GTF, specify the number of 4K GTF trace buffers (on the BUF parameter) to be saved for a dump. When the dump is taken, the trace records are passed to the SVC dump routine and become a part of the dump.

ACTION=NODUMP Option

ACTION=NODUMP indicates that SLIP is to set a flag in the RTM work area which is checked by the dump programs ABEND and SVC dump. If the bit is on, all dump requests are ignored. Because the bit is in the RTM work area, only dumps requested during processing of this error by RTM (requested by an FRR and/or an ESTAE) are suppressed. Should the error involve recursive entry into RTM, the bit setting is propagated to the next RTM work area. ACTION=NODUMP applies only to non-PER traps for errors in the RTS and RT2 environments.

This action is useful for preventing dumps that may not be needed (for example, X37, etc.) because accompanying messages provide sufficient information. It can also be used to prevent duplicate dumps for known problems which have already been documented.

ACTION=IGNORE Option

ACTION=IGNORE indicates that the SLIP processor is to take the IGNORE action. The IGNORE action does not result in any specific action being taken but a match is indicated for the trap and other processing for the trap occurs normally (such as messages being issued, and processing for the MATCHLIM, PRCNTLIM, RECOVERY, and DEBUG options).

This option is generally used on a trap to prevent a different, and more general trap, from matching. (Note that for any event, the SLIP processor stops examining SLIP traps for a match condition when a matching trap is found.) Because SLIP traps are tested in last-in-first-out order, IGNORE traps used in this way must be entered after the more general non-IGNORE trap. Also, the traps should be specified in the disabled state to prevent the non-IGNORE trap from matching while the IGNORE traps are being specified. After the non-IGNORE and all related IGNORE traps have been set, they can be enabled in a last-in-first-out order by using the MOD operand of the SLIP command.

For PER traps, the IGNORE trap must be of the same type (IF, SA, or SB) as the non-IGNORE trap or it will not be tested. For IF and SB PER traps, IGNORE traps can be used to simulate multiple ranges for monitoring as shown in Example 14 in the following topic "Examples of Using the SLIP Command." This technique cannot be used for SA PER traps. The use of the IGNORE trap with a more general IF or SB PER trap does not prevent PER interrupts from occurring in the range specified on the IGNORE trap. You should consider this when you are selecting a percent limit value.

In general, there is no limit to the number of IGNORE traps that can be set to work in conjunction with a non-IGNORE trap. You should be aware that IGNORE traps are considered as independent traps, and the SLIP command processor does not know when IGNORE traps are being used in conjunction with a non-IGNORE trap. For example, at the time a trap is being set, no checking is done between traps to ensure that the range to be ignored falls within the range specified on the non-IGNORE PER trap. Such checking is the responsibility of the user.

ACTION Keyword With RECOVERY Option (PER Traps Only)

Normal processing of a PER interrupt causes control to be returned to the next sequential instruction after the PER interrupt is processed. The RECOVERY keyword can be used to force recovery processing to be initiated after the PER interrupt has been processed.

The RECOVERY keyword is used to initiate recovery processing in those situations when an error is occurring, but the error is not being detected by the system or it is being detected too late for recovery routines to adequately handle the error situation. By initiating recovery processing via a SLIP trap, you have a way to use the error correction function that is built into MVS recovery routines.

To avoid unexpected results when using the RECOVERY keyword, you should be thoroughly familiar with the MVS recovery concepts and ensure that:

- Recovery is initiated at an appropriate point in the program.
- The recovery routine is designed to handle the error situation that exists at that point.

The RECOVERY action initially causes an 06F abend code to be generated.

Dump Tailoring

When ACTION=SVCD or ACTION=TRDUMP is specified on a SLIP trap, the ASIDLST, SDATA, SUMLIST, and LIST keywords can be used to tailor the dump to the particular problem that is being trapped.

ASIDLST Keyword

The ASIDLST keyword is used to specify the address spaces that are to be dumped. Note that a specification of zero indicates the home address space (pointed to by PSAAOLD).

SDATA Keyword

The SDATA keyword is used to specify the system data areas that are to be dumped. If the default SDATA specification is used, the current system CHNGDUMP setting can affect (add to or override) the areas requested. The system CHNGDUMP settings do not affect (add to or override) the areas specified on SDATA except when CHNGDUMP has been set with the NODUMP option. In this case, the SLIP trap does not produce a dump when the trap matches.

When SDATA is specified, the areas specified to be dumped completely replace the default specification on the dump request. For example, on an ACTION=SVCD dump, if SDATA=(NOSQA) is specified, the NOSQA completely replaces the default SDATA specification of SQA, RGN, TRT, LPA, CSA, NUC, ALLPSA, and SUM. The dump request of NOSQA is presented to SDUMP which merges it with its own defaults (SQA, SUM, and ALLPSA) and, in this case, produces a dump that contains only a summary dump and ALLPSAs.

Also, the SLIP command processor does not make any reasonability checks on the SDATA options specified. For example, SDATA=(SQA,NOSQA) is allowed even though the SQA and NOSQA options are contradictory. In this case, SQA would not be part of the dump produced.

SUMLIST and LIST Keywords

The SUMLIST and LIST keywords are used to dump user-defined areas of storage. The storage areas are defined by specifying address space qualifiers followed by address pairs that specify the beginning and ending addresses of storage to be dumped. Address space qualifiers can be either explicit or symbolic. They specify the address space to which the address pairs refer to. If a qualifier is not specified, the previous qualifier is used as the default. If the first address pair does not have a qualifier, CURRENT is used as the default. The beginning address must be less than or equal to the ending address. If the beginning address is greater, then the characters *A1>A2* are dumped instead of the requested area. Direct or indirect addresses can be used to specify the address pairs and can be mixed. Indirect addresses are resolved using the registers at the time of error or interrupt. If for any reason an indirect address cannot be resolved, the characters *RC=4* are dumped rather than the requested area.

The difference between SUMLIST and LIST is the point in time when the requested information is collected. SUMLIST collects information as a part of SDUMP summary dump processing. This is close to the time of error or PER interrupt and the information that is collected will probably be unchanged since the time of error or interrupt. (Note that storage areas must be paged in; and if not paged in, they are ignored by SDUMP.) LIST collects information when the scheduled dump is processed. (Note that storage areas are paged in if necessary.) This is some time after the error or interrupt and the information that is collected may have been changed since the time of error or interrupt.

Examples of Using the SLIP Command

The following examples briefly describe a system problem and show the SLIP command that can be used to match on a system event. The resulting dump or GTF SLIP record can then be used by the debugger to obtain diagnostic information in order to solve the problem.

Example 1: Match on Storage Alteration

Problem: An unknown program is incorrectly modifying location CD3010 in the LPA.

Action: The debugger sets the following SLIP trap:

```
SLIP SET,SA,ENABLE,ACTION=SVCD,
     RANGE=(CD3010),END
```

Result: When location CD3010 is altered, a SLIP match occurs and an SVC dump is scheduled. For this PER trap, MATCHLIM defaults to 1 which prevents a dump from being taken each time that location CD3010 is altered.

Example 2: Match on Storage Alteration

Problem: Same as Example 1 except location CD3010 is normally modified by JES2 (ASID=3) but should not be modified by any other program (in ASIDs 1, 2, 4, 5, 6, 7, 8, and 9).

Action: The debugger sets the following SLIP trap:

```
SLIP SET,SA,ENABLE,ACTION=SVCD,
     RANGE=(CD3010),ASID=(1,2,4,5,6,7,8,9),END
```

Result: When any program in an address space specified on ASID= alters location CD3010, a SLIP match occurs and an SVC dump is scheduled.

Example 3: Match on Storage Alteration

Problem: Same as Example 2 except there is an unknown number of address spaces (in addition to JES2 in ASID 3).

Action: An IGNORE trap is used in conjunction with the non-IGNORE PER trap. The debugger sets the following SLIP traps:

```
SLIP SET,SA,ID=TRP1,DISABLE,ACTION=SVCD,
     RANGE=(CD3010),END
SLIP SET,SA,ID=TRP2,DISABLE,ACTION=IGNORE,
     ASID=(3),END
```

Then issues the following SLIP commands:

```
SLIP MOD,ENABLE,ID=TRP2
SLIP MOD,ENABLE,ID=TRP1
```

Result: Any alterations to location CD3010 by JES2 (ASID=3) are ignored. Alterations to location CD3010 by programs in any other address space result in a SLIP match, and an SVC dump is scheduled. Note that the SLIP traps are inspected in a last-in-first-out (LIFO) order.

Example 4: Match on Storage Alteration

Problem: Location CD3010 contains an address that is normally modified by many programs. Intermittently, it is set to zero and causes an error.

Action: The debugger sets the following trap:

```
SLIP SET,SA,ENABLE,ACTION=SVCD,
     RANGE=(CD3010),DATA=(CD3010,EQ,00000000),END
```

Result: When location CD3010 is set to zero, a SLIP match occurs and an SVC dump is scheduled.

Example 5: Match on Instruction Fetch

Problem: An LPA routine is consistently abending and the debugger does not know if the routine is in error or it is being passed bad parameters. The LPA routine entry point is CD3100.

Action: The debugger sets the following SLIP trap:

```
SLIP SET,IF,ENABLE,ACTION=SVCD,
     RANGE=(CD3100),END
```

Result: The routine still abends. However, when entry is made to the routine at location CD3100, a SLIP match occurs and an SVC dump is scheduled. Information in the SVC dump allows the debugger to determine the validity of the parameter data.

Example 6: Match on Successful Branch

Problem: The debugger needs a "branch trace" of the instruction path taken through module MOD01 starting at offset X'108' through X'4FC' during the execution of the JOBX.

Action: GTF must be active with the GTF trace option SLIP and MODE=EXT specified in order to collect the GTF SLIP trace records in an external data set. The debugger sets the following SLIP trap:

```
SLIP SET,SB,ENABLE,ID=PER1,ACTION=TRACE,
     LPAMOD=(MOD01,108,4FC),JOBNAME=JOBX,
     MATCHLIM=20,END
```

Result: When 20 successful branch events have occurred during the execution of MOD01 when JOBX is in control, the trap is automatically disabled because MATCHLIM=20. The collected GTF SLIP standard trace records may be printed by the debugger via the EDIT function of the AMDPRDMP service aid.

Example 7: Match on Successful Branch

Problem: For Example 6, if the debugger wants to collect the GTF SLIP standard trace records in GTF's address space and obtain these records as a part of an SVC dump, then GTF must be active with GTF trace option SLIP and MODE=INT specified.

Action: The debugger sets the following SLIP trap:

```
SLIP SET,SB,ENABLE,ACTION=TRDUMP,
     LPAMOD=(MOD01,108,4FC),JOBNAME=JOBX,
     MATCHLIM=20,END
```

Result: When 20 successful branch events have occurred in the range of addresses from 108 to 4FC in MOD01 while JOBX is in control, an SVC dump is scheduled. The dump contains the GTF trace buffers (assuming the SDUMP TRT trace option is in effect). The number of GTF buffers dumped is determined by the BUF parameter on the GTF START command. Note that if the CHNGDUMP command has been invoked, then the latest areas defined by CHNGDUMP are dumped.

Example 8: Match on Instruction Fetch

Problem: The debugger needs to collect specific data (as a part of a summary dump) when the instruction at offset X'200' in MOD02 is executed in ASID 27.

Action: The debugger sets the following trap:

```
SLIP SET,IF,ENABLE,ACTION=SVCD,
     LPAMOD=(MOD02,200),ASID=(27),
     SUMLIST=(2R%%,2R%%+DCF,4R%%+28,4R%%+1C9),
     SDATA=SUMDUMP,MATCHLIM=1,END
```

Result: When the instruction at offset X'200' in module MOD02 is executed in ASID 27, the selected data (specified on SUMLIST) is gathered. Control will resume at the next sequential instruction after the point of interrupt. Additionally, the trap is disabled after a single match occurs because MATCHLIM = 1.

Note: To obtain the same data, the SUMLIST keyword could have been specified as:

```
     SUMLIST=(2R%%,+DCF,4R%%+28,+1C9),
```

Example 9: Match on Program Check

Problem: The debugger wishes to take a dump of the current private region when an 0C7 program check occurs during task mode processing in module MOD03.

Action: The debugger sets the following SLIP trap:

```
SLIP SET,ENABLE,ERRTYP=PROG,ACTION=SVCD,
     COMP=0C7,PVTMOD=MOD03,MODE=TCB,
     SDATA=RGN,END
```

Result: When the 0C7 program check occurs in TCB mode, the trap matches and an SVC dump is scheduled. Normal recovery processing then takes place.

Example 10: Match on Completion Code

Problem: The debugger wishes to take an SVC dump when a 806 completion code occurs for a job TEST99 when program PGM5 is in control.

Action: The debugger sets the following trap:

```
SLIP SET,ENABLE,COMP=806,ACTION=SVCD,
     JOBNAME=TEST99,JSPGM=PGM5,END
```

Result: When the job step that executes program PGM5 of job TEST99 is abended with a completion code of 806, the trap matches and an SVC dump is scheduled.

Example 11: Match on SVC Error

Problem: The debugger wishes to force the system into a wait state when an SVC error occurs in job TEST98 to take a stand-alone dump.

Action: The debugger sets the following trap:

```
SLIP SET,ENABLE,ERRTYP=SVCERR,ACTION=WAIT,
     JOBNAME=TEST98,END
```

Result: When job TEST98 encounters an SVC error, all processors in the system are put into the wait state. The operator may then initiate a Stand-alone dump (SADMP).

Example 12: Match on Data

Problem: The debugger wishes to force entry into recovery processing for LPA module MODX when MODX is processing a specified input (X‘0105’).

Action: The debugger sets the following SLIP trap but does not start GTF:

```
SLIP SET,IF,ENABLE,ACTION=(TRACE,RECOVERY),
     LPAMOD=(MODX,16),DATA=(1R%,EQ,0105),
     MATCHLIM=1,END
```

Result: When the input parameter pointed to by general purpose register 1 is equal to X‘0105’, the trap matches and the SLIP processor forces the recovery path to be taken. Because GTF is not active, no trace record is written. The trap is then disabled because MATCHLIM = 1.

Example 13: Match on Storage Alteration

Problem: The debugger wants to monitor a common storage location in a production system for alteration to X‘F1F2F3F4’.

Action: To keep the trap overhead to a minimum, the debugger specifies the JOBNAME and ASID keywords. Also, because the trap is being set on a production system, the PRCNTLIM keyword is specified to prevent the trap from using more than 20% of the available system processing time. The debugger sets the following trap:

```
SLIP SET,SA,ENABLE,ACTION=SVCD,
     ASID=(7,9),JOBNAME=JOBX,
     RANGE=(CD3100,CD3103),
     DATA=(CD3100,EQ,F1F2F3F4),
     PRCNTLIM=20,END
```

Result: When the specified area is altered by JOBX in ASID 7 or 9 and the pattern of data is X‘F1F2F3F4’, then the trap matches and an SVC dump is scheduled. The processing time used by the PER interrupts is being monitored and if the time exceeds 20% of the available system time, the trap is disabled and the debugger is notified.

Example 14: Match on Instruction Fetch

Problem: The debugger wants to monitor a range of instruction addresses in LPA module MODX but ignore instructions that form an iterative loop within a subset of this range.

Action: The debugger sets the following traps:

```
SLIP SET,IF,DISABLE,ID=TRP1,ACTION=TRACE,
     LPAMOD=(MODX,110,1FB),JOBNAME=JOB1,
     TRDATA=(STD,REGS),MATCHLIM=500,END
SLIP SET,IF,DISABLED,ID=TRP2,ACTION=IGNORE,
     LPAMOD=(MODX,1C4,1D7),END
```

Then the debugger issues the following SLIP commands:

```
SLIP MOD,ENABLE,ID=TRP2
SLIP MOD,ENABLE,ID=TRP1
```

Result: PER interrupts are taken for each instruction that is executed within the range specified on trap TRP1, but those interrupts that fall within the range specified on trap TRP2 are ignored. Therefore, tracing occurs in MODX for those instructions that fall in the ranges of X'110' to X'1C3' and X'1D8' to X'1FB'. Note that the IGNORE trap must be defined after the non-IGNORE trap because the traps are processed for match tests in last-in-first-out order.

Note: The use of IGNORE traps with non-IGNORE PER traps effectively allows you to discard selected events that occur in one or more subsets of the monitored range. Be aware that PER interrupts occur within the ignored ranges and cause system degradation.

Example 15: Match on Instruction Fetch

Problem: The debugger wants to force the system into a wait state when JOB1 executes in address space 5, 7, or 10 within a given address range.

Action: The debugger sets the following trap:

```
SLIP SET,IF,ENABLE,ID=PNK1,ACTION=WAIT,MODE=(HOME),
     JOBNAME=JOB1,ASID=(5,7,10),RANGE=(E300,E000),END
```

Result: When job JOB1 starts in address space 5, 7, or 10, and an instruction is fetched within the specified address range, then the trap matches and the system is put in a wait state.

Note: Because MODE=HOME was specified, if job JOB1 starts in address space 5, issues a program call to address space 7, and then an instruction is fetched within the specified address range, the trap will not match.

Example of SLIP Command From TSO Terminal

The following example shows the use of the SLIP command from a TSO terminal and the prompting that can occur.

Problem: A debugger suspects that a module is being passed an improper parameter list that causes the module ISTAPC11 to abend during job APPLE1. By the time the abend occurs, all evidence of the cause has been eliminated by recovery processing. A history of the caller's parameter list can be obtained by using the SLIP IF PER function with ACTION=TRDUMP.

Action: The debugger issues the following commands:

```
tso user:   OPERATOR

system:     OPERATOR

tso user:   slip set,if,enable,id = per1,action = trdump,
            jobname = apple1,lpamod = (sstapc11,16),
            trdata = (std,regs,1r%,1r% + 32),
            matchlim = 5,end

system:     IEE736D SLIP ID = PER1,SSTAPC11 IS NOT IN THE LPA.
            ENTER KEYWORD, NULL LINE, OR 'CANCEL'

tso user:   lpamod = (istapc11,16)

system:     IEE727I SLIP TRAP ID = PER1 SET BUT GTF IS NOT ACTIVE

tso user:   send 'please start gtf with mode = int and trace = slip and notify me when done'

system:     OPER GTF IS ACTIVE

tso user:   send 'please start apple1'

system:     IEA992I SLIP TRAP ID = PER1 MATCHED

system:     IEA411I SLIP TRAP ID = PER1 DISABLED FOR MATCHLIM

system:     IEA911A COMPLETE DUMP ON SYS1.DUMP01 TSO USERID (D10XYZ1)

tso user:   send 'please stop gtf'
```

Result: The dump has been taken. The debugger may now want to copy the dump to another data set, clear the SYS1.DUMP01 data set, and obtain a hardcopy of the dump. This additional processing can be accomplished from the TSO terminal by using TSO commands to execute the print dump (AMDPRDMP) program.

Note: You may want to establish a cataloged procedure to invoke GTF. This procedure could specify all of the desired GTF options and keywords for using GTF with the SLIP command. Using such a procedure would allow you (when working from a TSO terminal) to pass only the name of the procedure to be started to the operator instead of the GTF parameters as shown in the example. This reduces the burden on the operator and is more effective when communicating with the operator.

Designing an Effective SLIP Trap

The design of a SLIP trap requires knowledge of the error conditions and what makes the error unique. An effective trap should catch only the intended error. To do this, the description should be as specific as possible.

Note that SLIP does not detect any events that occur while running in TRASMODE mode. (Control register 1 points to the segment table origin for an address space other than the current address space.)

The best way to design a trap is from a dump of the error. In the case of the NODUMP action, a dump should be available. In other cases, an approximate dump (one taken near the time of the error) or one without sufficient information to debug might be available.

It should be understood that for error events (non-PER traps), SLIP operates as a subroutine within the RTM. SLIP is called from either RTM1 or RTM2, depending on whether the error environment allowed FRR or only ESTAE recovery respectively. The level of RTM in control affects the data areas available. The calls to SLIP are prior to calls to any error recovery routine, therefore it is possible that the data areas contained in a dump may have been changed since SLIP examined them. This is especially true of the COMP keyword value. Many recovery routines change the abend completion code to make it more specific. For example, a system service that receives a bad address from a user parameter list will get an 0C4 which it converts to its own completion code meaning a bad parameter list.

Controlling SLIP Traps

This topic describes how the MATCHLIM and PRCNTLIM keywords are used to limit the system resources that a SLIP trap is allowed to use. Also, for PER traps, it includes some performance hints.

MATCHLIM Keyword

The MATCHLIM (match limit) keyword provides a way to set an upper limit on the number of times that a trap is allowed to match. This keyword can be specified to ensure that resources are not used unnecessarily (for example, using dump data sets when ACTION=SVCD is chosen) or to remove the overhead associated with a PER trap as soon as possible.

The MATCHLIM keyword should always be specified for enabled traps that are unattended in order to avoid undesirable results, such as filling several dump data sets for multiple occurrences of the same error.

If MATCHLIM is not specified on a trap, there is no limit to the number of times the trap can match. However, if MATCHLIM is not specified on a PER trap with ACTION=SVCD, the trap is disabled after one match.

If a MATCHLIM value has been specified for a trap, you can tell if the trap is matching and approaching the limit set by displaying the trap via the DISPLAY command.

When the MATCHLIM keyword is used on an IGNORE type trap, it can provide the effect of ignoring a specified number of events before an action is taken by the associated non-IGNORE trap.

PRCNTLIM Keyword

The PRCNTLIM (percent limit) keyword provides a way to set a limit on the amount of system time that is committed to processing on behalf of a PER trap. The percentage of time that is computed is based on the amount of time spent processing PER interrupts and space switch interrupts (as compared to the amount of time elapsed since the first PER interrupt for the trap). The percentage that is computed is related only to software processing and does not include any PER hardware processing. The percentage is computed each time that a PER interrupt is processed and is compared to the percent limit specified on the PRCNTLIM keyword.

Percent limit checking is not performed for the first 33 seconds (approximately) after the first interrupt is taken for the trap. This avoids a high initial percentage which might disable the trap immediately.

The accuracy of the percent limit calculation is affected by the instructions that are executed on behalf of the PER interrupt and space switch interrupt but are not included in the calculation. These instructions include:

- instructions that are executed before the first timestamp is taken. (For example, instructions in the program check first level interrupt handler.)
- instructions that are executed after the last timestamp is taken. (For example, the percent limit calculation itself.)
- instructions that are indirectly related to the PER interrupt. (For example, instructions used for PER trap messages.)

Because these instructions are not accounted for in the percent limit calculation, the accuracy of the calculation may vary between different traps. For example, if there are many very explicit traps to be checked and one of them takes an action, the large number of instructions executed between the timestamps taken will result in a fairly accurate percentage calculation even though some instructions were not accounted for in the calculation. Conversely, if there is one very simple trap that does not match, the instructions that are not accounted for in the calculation represent a large portion of the instructions executed and their exclusion makes the percentage calculation more inaccurate.

Also, the percentage calculation is affected because the calculated percentage is truncated to an integer.

If you have any doubt as to whether or not a PER trap will work properly in a system, a conservative value (such as the default PRCNTLIM=10) should be chosen. Thus, if the trap should consume large amounts of processing, it will be quickly disabled. After approximately 33 seconds, you can display the PER trap (via the DISPLAY command) and obtain the current system percent utilization by the trap.

Specifying PRCNTLIM=99 indicates that percent limit checking is not to be performed. It is intended for non-critical environments (such as testing) and certain special situations (see the ACTION=WAIT option). In general, all non-IGNORE PER traps should have a reasonable value specified for percent limit.

Performance Hints For PER Traps

For PER traps, you can minimize system performance degradation by:

- Choosing values for the RANGE and LPAMOD keywords which will reduce the range of storage that is monitored by the PER hardware. This will avoid unnecessary PER interrupt processing.
- Specifying the ASID and/or JOBNAME event qualifier keywords to avoid having PER active in all address spaces in the system. When ASID and/or JOBNAME is specified, PER monitoring is set up in only the requested address spaces. Performance degradation due to PER monitoring occurs only when the requested address spaces are in control. Also, when MODE=HOME is specified, PER monitoring is set up only when the unit of work is executing in the home address space.

Placement of PER Traps

The PER support provided by SLIP is designed to be non-disruptive at the possible expense of not collecting data or performing a user requested action. Several aspects of this non-disruptive characteristic are discussed in this topic:

Certain parts of the system cannot tolerate PER interrupts. For those parts, the PSW PER bit is set off to prevent interrupts. Most notably, the PER bit is set off in the program check, machine check and restart new PSWs. PER remains off in such critical paths until processing reaches a point where a PER interrupt is considered "safe." For example, if a SLIP IF PER trap is set on for the first instruction in the program check FLIH, no PER interrupts would occur and the trap would not match. Note, however, that you are not prevented from setting such a trap.

Some PER interrupts that occur are not always processed by the SLIP processor. The SLIP processor ignores (that is, does not process) PER interrupts if the interrupt:

- occurred while DAT was off (PER support for SLIP applies only to virtual addresses).
- is redundant. (Refer to *S/370 Principles of Operation* for a description of redundant PER interrupts.)
- occurred while an enabled non-IGNORE PER trap does not exist. Note that this implies that PER interrupts caused by a non-SLIP tool which has set up the PER control registers is ignored by SLIP and the PER bit is turned off by SLIP in the resume PSW before returning to the program check FLIH.

Because the SLIP processor uses certain system services, SLIP is sensitive to the recursive use of those services where a recursive entry could cause an error. Recursive calls to a function may occur in one of two ways; either directly or indirectly. A direct recursion is the result of placing a PER trap in a function and then causing SLIP to use that function. For example, suppose a SLIP trap is placed in GTF entry code and the action specified is TRACE. If the trap matched and SLIP tried to take the action, a GTF trace record would not be written because of the recursive checks within GTF. A similar situation exists with other tracing actions, dump actions, and wait. In general, direct recursions result in the action not being taken. Such direct recursions can be avoided by the appropriate choice of another SLIP action. Note that you are not prevented from setting a trap that can cause a direct recursion.

Indirect recursions are a result of a PER interrupt occurring in a system service that is called by a service that SLIP uses. For example, suppose a PER interrupt occurred in the lock manager, the non-IGNORE PER trap matched and the action was SVCD with a summary dump requested. To produce a summary dump, SVC dump calls the lock manager to obtain certain locks. If the recursive call to the lock manager is allowed, an error could result due to information being overlaid because of the recursive call. In this case, SLIP suppresses the summary portion of the SVC dump to avoid the recursive call to the lock manager. A similar situation can occur if a PER interrupt occurred in code where the SALLOC lock is held (for example, in an RSM module) and a summary dump is requested (thus causing RSM to be recursively entered). In this situation also, the summary dump is suppressed. For both of these situations, the debugging information in the SDUMP 4K buffer and the asynchronous portion of the dump are available for debugging.

PER Monitoring and Checkpoint/Restart

For PER monitoring, specific PER support is not included in the checkpoint/restart function. Therefore, two possibilities exist for a checkpointed program.

Case 1. A program is running in an address space that has not been selected for PER monitoring and the program is checkpointed.

Case 2. A program is running in an address space that has been selected for PER monitoring and the program is checkpointed.

These two cases could result in one of three conditions when the checkpointed program is restarted.

- In Case 1, if the program is restarted in an address space that has been selected for PER monitoring, the restarted program is not monitored.

- In Case 2, if the program is restarted in an address space that has not been selected for PER monitoring, but other address spaces are being monitored, unwanted PER interrupts may occur (depending on the PER control register settings). If unwanted PER interrupts occur in the restarted program, the PSW PER bit is turned off in the restarted program. This may eventually remove all possible degradation due to the unwanted PER interrupts from the restarted program.

- In Case 2, if the program is restarted and PER monitoring is not active in the system (that is, control register nine is zero), the system may suffer some degradation due to the PSW PER bit being on in the restarted program as the restarted program is running.

SLIP Command Keyword Summary

Figure 2-17 is a summary of the SLIP command keywords. The keywords are shown across the top of the figure and are grouped by the function that they perform. For each keyword, a brief description is given for the use of the keyword, the valid options, required and default options, restrictions, comments and other information.

Figure 2-17 (Part 1 of 3). SLIP Command Summary

	SLIP Control Keywords			Trap Type Keywords			Trap Control Keywords		Specialized Keywords		
	SET	DEL	MOD	IF	SB	SA	ENABLE	DISABLE	ID	END	DEBUG
PER	Set a PER trap	Delete one or more PER traps.	Enable or disable one or more PER traps.	Set an Instruction fetch PER trap.	Set a successful branch PER trap.	Set a storage alteration PER trap	Enable a PER trap.	Disable a PER trap.	Provide an identifier for a PER trap.	Indicates the trap definition is complete.	Causes the GTF SLIP DEBUG record to be written when the trap is checked.
Non–PER	Set a non-PER trap.	Delete one or more non-PER traps.	Enable or disable one or more non-PER traps.	– – –	– – –	– – –	Eanble a non-PER trap.	Disable a non-PER trap.	Provide an identifier for a non-PER trap.	Indicates the trap definition is complete.	Causes the GTF SLIP DEBUG record to be written when the trap is checked.
Valid Options	ACTION ADDRESS ASID ASIDLST ASIDSA COMP DATA DEBUG DISABLE ENABLE END ERRTYP IF ID JOBNAME JSPGM LIST LPAMOD MATCHLIM MODE PRCNTLIM PVTMOD RANGE RBLEVEL SA SB SDATA SUMLIST TRDATA	ALL ID	ALL DISABLE ENABLE ID	ACTION ASID ASIDLST DATA DEBUG DISABLE ENABLE END ID JOBNAME JSPGM LIST LPAMOD MATCHLIM MODE PRCNTLIM RANGE SDATA SUMLIST TRDATA	Same as IF.	ACTION ADDRESS ASID ASIDLST ASIDSA DATA DEBUG DISABLE ENABLE END ID JOBNAME JSPGM LIST LPAMOD MATCHLIM MODE PRCNTLIM PVTMOD RANGE SDATA SUMLIST TRDATA	– – –	– – –	Four-character identifier.	– – –	– – –
Required	END	ID or ALL	ID or ALL, and ENABLE or DISABLE	RANGE or LPAMOD	RANGE or LPAMOD	RANGE (unless ACTION=IGNORE is specified)	– – –	– – –	– – –	– – –	– – –
Default	ENABLE, ACTION=SVCD, and RBLEVEL=ERROR	– – –	– – –	– – –	– – –	– – –	– – –	– – –	System supplied.	– – –	– – –
Restrictions	Must follow SLIP	Must follow SLIP.	Must follow SLIP.	Must follow SET.	Must follow SET.	Must follow SET.	– – –	– – –	– – –	Specify at end of the command.	– – –
Comments	– – –	– – –	– – –	Qualify with ASID and/or JOBNAME to minimize performance degradation.	Same as IF.	Same as IF.	Only one non-IGNORE PER trap may be enabled.	– – –	Supplied by system if not specified on SET.	Use with SET.	GTF must be active and with the SLIP option chosen.
Minimum Value	– – –	– – –	– – –	– – –	– – –	– – –	– – –	– – –	– – –	– – –	– – –
Maximum Value	– – –	– – –	– – –	– – –	– – –	– – –	– – –	– – –	– – –	– – –	– – –
Special Value	– – –	ALL indicates all SLIP traps.	ALL indicates all SLIP traps.	– – –	– – –	– – –	– – –	– – –	– – –	– – –	– – –
Abbreviation	– – –	– – –	– – –	– – –	– – –	– – –	EN	D	– – –	E	– – –
Example	SLIP SET, . . .	SLIP DEL, . . .	SLIP MOD, . .	SLIP SET, IF, . .	SLIP SET, SB,. .	SLIP SET, SA,. . .	SLIP . . . , EN , . . .	SLIP . . . , D, . . .	ID=TRP1	SLIP SET, . . . ,END	DEBUG

Figure 2-17 (Part 2 of 3). SLIP Command Summary

	Trap Event Qualifier Keywords										
	ADDRESS	ASID	ASIDSA	COMP	DATA	ERRTYP	JOBNAME	JSPGM	LPAMOD	MODE	PVTMOD
PER	For SA, specifies the address range of the instruction that must cause the storage alteration.	Specifies the ASIDs (address spaces) to be monitored.	Specifies the ASIDs (address spaces) in which the storage being altered resides.	– – –	Specifies condition of storage or registers.	– – –	Specifies the job to be monitored.	Specifies the job step program name that must be in control.	For IF and SB, defines the range to be monitored. For SA, specifies the LPA module that must cause the storage alteration	Specifies system mode.	For SA, specifies private module that must cause storage alteration.
Non-PER	Specifies the address range that must be in control when the error occurs.	Specifies the ASIDs (address spaces) that must be in control when the error occurs.	– – –	Specifies a user or system completion code.	Specifies condition of storage or registers.	Specifies system-detected error type.	Specifies the job that must be in control when the error occurs.	Specifies the job step program name that must be in control when the error occurs.	Specifies the LPA module that must be in control when the error occurs.	Specifies system mode.	Specifies private module that must be in control when the error occurs.
Valid Options	Start address End address	– – –	– – –	Udddd – user code hhh – system code	Direct address Register Indirect address Operator – EQ, NE, LT, GT, NG, NL.	ABEND DAT MACH MEMTERM PGIO PROG REST SVCERR	Any valid job name, TSO ID, or started task name.	Any valid job step program name.	Name Start displacement End displacement	ALL DIS GLOC GLOCSD GLOCSP HOME LLOC LOCK PKEY PP RECV SKEY SRB SUPER SUPR TCB TYP1 – – with ANY or EVERY	Name Start dispalcement End displacement
Required	– – –	– – –	– – –	– – –	– – –	– – –	– – –	– – –	Name	– – –	Name
Default	– – –	– – –	– – –	– – –	– – –	– – –	– – –	– – –	Entire module.	ANY (if EVERY is not specified with an option).	Entire module
Restrictions	For PER traps, SA only. Start address must be less than or equal to end address	Maximum of 16 ASIDs can be specified.	– – –	– – –	– – –	– – –	– – –	– – –	– – –	RECV cannot be specified for a PER trap. If ALL is specified for a PER trap. RECV is not included.	For PER traps, SA only.
Comments	Specify virtual addresses in hexadecimal.	Limits PER monitoring to the ASIDs specified.	– – –	Specify user code as four decimal digits. Specify system code as three hexadecimal digits.	– – –	Logical "or" if more than one error is specified.	Limits PER monitoring to the ASID of the job.	– – –	Specify displacement in hexadecimal	HOME is not included when ALL is specified. If HOME is specified, the unit of work must be executing in HOME even if ANY is also specified.	Specify displacement in hexadecimal.
Minimum Value	0	0	– – –	0	– – –	– – –	– – –	– – –	– – –	– – –	– – –
Maximum Value	00FFFFFF	Installation-defined (ASVTMAXU).	Maximum of 16 ASIDs can be specified.	U9999 or FFF.	– – –	– – –	– – –	– – –	– –	– – –	– – –
Special Value	– – –	– – –	– – –	x means don't care.	– – –	ALL – indicates all valid options.	– – –	– – –	– – –	EVERY – logical "and" ANY – logical "or" ALL – all modes	– – –
Abbreviation	AD	AS	ASA	C	DA	ER	J	JS	L	M	P
Example	AD=(200,300)	AS=(1,6,12A)	ASA=(5,S,3,HASID)	C=06F	DA=(2R,EQ,00)	ER=(DAT)	J=MYJOB	JS=IFOX00	L-MYMOD2	M-(GLOC)	P=MYMOD1

(continued)

RANGE	RBLEVEL
Specifies the range of addresses to be monitored.	– – –
– – –	Specifies the source RB for the registers and PSW at the time of error.
Start address End address	ERROR NOTSVRB PREVIOUS
Start address	– – –
– – –	ERROR
For SA PER traps, cannot be specified with ACTION=IGNORE.	– – –
Specify virtual addresses in hexadecimal. Start address may be greater than, less than, or equal to end address.	Applies to unlocked task mode errors only.
0	– – –
00FFFFFF	– – –
– – –	– – –
RA	RB
RA=(600,700)	RB=NOTSVRB

Figure 2-17 (Part 2 of 3). SLIP Command Summary

	Action Keyword	Dump Tailoring Keywords				Trace Tailoring	Automatic Control Keywords	
	ACTION	ASIDLST	LIST	SDATA	SUMLIST	TRDATA	MATCHLIM	PRCNTLIM
PER	Specifies the action to be taken when the trap matches.	Specifies the ASIDs (address spaces) to be dumped.	Specifies a list of storage ranges to be included in the dump.	Specifies the system information to be included in the dump.	Specifies a list of storage ranges to be included in the summary dump.	Specifies the type and content of the GTF records to be collected.	Specifies the maximum number of times that the trap can match.	Specifies the maximum amount of system time that PER processing is allowed.
Non-PER	Specifies the action to be taken when the trap matches.	Specifies the ASIDs (address spaces) to be dumped.	Specifies a list of storage ranges to be included in the dump.	Specifies the system information to be included in the dump.	Specifies a list of storage ranges to be included in the summary dump.	Specifies the type and content of the GTF records to be collected.	Specifies the maximum number of times that the trap can match.	– – –
Valid Options	IGNORE NODUMP RECOVERY SVCD TRACE TRDUMP WAIT	Hexadecimal ASID value.	Direct address pairs. Indirect address pairs.	ALLPSA CSA GRSQ LPA LSQA NOALLPSA/NOALL NOSQA NOSUMDUMP/NOSUM NUC PSA RGN SQA SUMDUMP/SUM SWA TRT	Direct address pairs. Indirect address pairs.	Direct address pairs. Indirect address pairs. REGS STD	Decimal integer.	Decimal integer.
Required	– – –	– – –	– – –	– – –	– – –	– – –	– – –	– – –
Default	SVCD	– – –	– – –	For ACTION=SVCD, SDATA=(ALLPSA, CSA, LPA, NUC, RGN, SQA, SUMDUMP, TRT). For ACTION=TRDUMP, SDATA=TRT.	– – –	STD	1 – for PER traps with ACTION=SVCD specified or defaulted.	10
Restrictions	WAIT – not available to TSO user. RECOVERY -- PER traps only.	Specify a maximum of 15 ASIDs. ACTION=SVCD or ACTION=TRDUMP must be specified or defaulted.	Start storage range address must be less than or equal to end storage range address. ACTION=SVCD or ACTION=TRDUMP must be specified or defaulted.	ACTION=SVCD or ACTION=TRDUMP must be specified or defaulted.	Start storage range address must be less then or equal to end storage range address. SUMDUMP/SUM must be specified or defaulted. ACTION=SVCD or ACTION=TRDUMP must be specified or defaulted.	ACTION=TRACE or ACTION=TRDUMP must be specified.	– – –	Applies to non-IGNORE PER traps only.
Comments	Choose only one option – except RECOVERY which must be specified with one of the other options.	– – –	– – –	CHNGDUMP settings are overridden when SDATA is specified (except for NODUMP option).	– – –	GTF record is 256 bytes maximum. STD uses 120 bytes. REGS uses 65 bytes. GTF must be active with the SLIP option.	– – –	– – –
Minimum Value	– – –	0	– – –	– – –	– – –	– – –	1	1
Maximum Value	– – –	Installation-defined (ASVTMAXU).	– – –	– – –	– – –	– – –	65535	99
Special Value	– – –	0 - indicates the current ASID.	– – –	– – –	– – –	STD – standard record information. REGS – registers.	– – –	99 -- no checking is performed
Abbreviation	A	AL	LS	SD	SL	TD	ML	PL
Example	ACTION=(TRACE)	ASIDLST=(0,1,LLOC,P)	LIST=(H.200,300, 2R%%, +20)	SDATA=(SQA,NUC)	SUMLIST=(S.400,500, 2R%%,+20)	TRDATA=(STD,REGS, 20,30,2R%,+40)	MATCHLIM=40	PRCNTLIM=20

Figure 2-17 (Part 3 of 3). SLIP Command Summary

System Stop Routine

On occasion it is necessary to stop the system and take a stand-alone dump to fully document a problem. Loading a wait state PSW is sufficient on a uniprocessor. Stopping only one processor on an MP system is not adequate. This routine will stop an MVS MP or UP system. The caller must be supervisor state and key zero. The wait state code you wish displayed is placed at location X'762'. This trap also moves the wait state PSW to storage location zero and loads the PSW from there to prevent inadvertent restarts when the trap is hit.

```
NAME     IEANUC01  IEAVFX00
 VER       0700    41F'00'
 REP       0700    ACFC075A                  DISABLE
 REP       0704    B6000764                  STORE CRO
 REP       0708    94EF0764                  TURN OFF PSA PROTECT
 REP       070C    B7000764                  RELOAD CRO
 REP       0710    900F0764                  SAVE REGISTERS
 REP       0714    58F00010                  GET CVT POINTER
 REP       0718    58E0F294                  GET CSD POINTER
 REP       071C    91C0E008                  TEST IF MP
 REP       0720    47E00750                  NO JUST LOAD WAIT PSW
 REP       0724    41200000                  SET REG 2 TO CPU 0
 REP       0728    41300001                  SET REG 3 TO CPU 1
 REP       072C    48400204                  GET CPU ADDRESS
 REP       0730    1244                      TEST FOR CPU 0
 REP       0732    47700748                  NO, STOP CPU 0 FIRST
 REP       0736    AE030009                  YES, STOP CPU 1 FIRST
 REP       073A    47600736                  SPIN TIL CC=0
 REP       073E    D2070000075C              MOVE THE WAIT PSW TO ZERO
 REP       0744    82000000                  LOAD WAIT STATE ON CPU 0
 REP       0748    AE020009                  SIGP STOP CPU 0
 REP       074C    47600748                  SPIN TIL CC=0
 REP       0750    D2070000075C              MOVE THE WAIT PSW TO ZERO
 REP       0756    82000000,0000             LOAD WAIT STATE ON CPU 1
 REP       075C    000E0000,0000DEAD         WAIT PSW
 REP       0764    00000000                  SAVE AREA
```

Note: Extreme care must be used when considering a system alteration in order to gather additional data about a problem. No superzaps should be applied before the system programmer has verified the logic being zapped and the trap logic itself. Remember if any one location or offset within the module or trap changes, all offsets and base registers *must* be verified.

How to Expand the Trace Table

To increase the size of the trace table, you may zap module IEAVNIP0 at label NVTTRACE to a greater value. It defaults to X'190' (400 decimal). Do not exceed a value of X'400' for the size of the trace table; 806-4 and 0C4 abends can occur when the link pack area directory is accessed.

```
NAME     IEANUC01  IEAVNIP0
 VER       3BB0     0190
 REP       3BB0     XXXX            WHERE X IS THE NEW VALUE DESIRED.
```

Note: Extreme care must be used when considering a system alteration in order to gather additional data about a problem. No superzaps should be applied before the system programmer has verified the logic being zapped and the trap logic itself. Remember if any one location or offset within the module or trap changes, all offsets and base registers *must* be verified.

Section 3. Diagnostic Materials Approach

This section provides guidelines for analyzing storage dumps to find which data areas were affected by the error and to isolate internal symptoms of the problem.

The three chapters in this section are:

- Stand-alone Dumps
- SVC Dumps
- SYSABENDs, SYSMDUMPs, and SYSUDUMPs

Stand-alone Dumps

The stand-alone dump provides the problem solver with a larger quantity of data than system-initiated dumps because it contains areas that belong to the entire operating system rather than just a single address space or component. One of the major problems for the analyst is finding the important data for his problem and then isolating the problem area. Once this isolation is achieved, the debugger uses unique system/component techniques to gain further insight into the exact cause of the problem.

This chapter points out where to look in a stand-alone dump to determine various problem symptoms. The general approach is to analyze a stand-alone dump to find out what the system is doing (or not doing). Important areas will be described and possible reasons for their current state/contents will be explained. The analysis starts at the global system level and, by gathering data and gaining an understanding of the environment, works down to the address space and task level.

The experienced problem solver realizes that under certain conditions it may be necessary or advantageous to omit interpreting various areas. For example, if during system operation you observe that a given segment of the segment (such as VTAM) is not functioning (other areas appear okay - jobs are executing, SYSIN/SYSOUT is appearing, etc.), you may decide to take a stand-alone dump. In this case, the current state of the system is probably not important. You would not be interested in current PSW, registers, etc.; you would be interested only in the address spaces that are using VTAM and the state of the TP network. The dump is not taken for a problem that is "active" now, but to give you data with which to determine a problem that appears to have originated some time ago. The point is that knowing why the dump was taken will often govern which, if any, of the stand-alone dump areas are of significance for a given problem.

Information contained in the chapter on "Waits" in Section 4 can be used as a supplement to the following discussions. (Also, a step-by-step approach to analyzing a stand-alone dump is contained in Appendix B of this manual.)

To analyze a stand-alone dump, you should always ask the following questions:

1. *Why was the dump taken?*

   Console sheets/logs are very important in stand-alone dump analysis. They are often the key to solving "enabled wait" situations and may present valuable information about system activity prior to taking the dump. Messages concerning I/O errors, condition code=3, SVC dumps, abnormal job terminations, device mounts, etc. should be thoroughly investigated to determine if they could possibly contribute to the problem you are tracking.

   The dump title gives an indication of the problem's external signs or, possibly, a specific situation that must be investigated, such as "VTAM NOT FUNCTIONING."

2. *What is the current state of the system?*

   Examine the available global data areas to determine what the system is currently doing. The "Global System Analysis" chapter in Section 4 aids in this process. Remember that at this point, you are gathering information and trying to understand the system environment in order to isolate the *internal* symptom; you are not ready yet to debug.

3. *Has your global analysis isolated the problem to an internal symptom?*

   If so, refer to the discussion of that symptom in Section 4 of this manual.

4. *What previous errors have occurred within the system; could they possibly have any affect on your current problem?*

   The interpretation of SYS1.LOGREC and the in-storage LOGREC buffers are most important in determining error history. See the chapter on "Use of Recovery Work Areas" in Section 2.

5. *What is the recent system activity?*

   The chapter on "MVS Trace Analysis" in Section 2 aids in trace table interpretation.

6. *What is the work status within the system?*

   Your objective is to determine if the system has for some reason not completed all scheduled work. Determining what that work is and why it is not progressing can provide insight into the problem as well as answer some questions that may have arisen during an earlier analysis. Understanding the major control block structure and work queue status should aid in determining the possible source of the error. Refer to the discussion of "Work Queues and Address Space Status" in the "Global System Analysis" chapter of Section 2.

At this point, you should have gathered enough data to have a definition of the internal problem symptom. You should also have considerable information about the system's state, error history, and job status. You should refer to the appropriate chapter in Section 4 "Symptom Analysis Approach" or, if you have isolated the error to a component or process, Section 5 or Appendix A, respectively.

SVC Dumps

SVC dumps (invoked by the SDUMP macro) are usually taken as a result of an entry into a functional recovery routine (FRR) or ESTAE routine. The component recovery routine specifies the addresses that will be dumped.

The "Component Analysis" chapters in Section 5 should help you identify what areas of the system were dumped and what they contain.

Also, "Appendix C: SDUMP Title Directory" lists the titles of SVC dumps initiated by system components and provides diagnostic information for the modules that issue the SDUMP macro.

SDUMP options SQA, ALLPSA, and SUMDUMP are the defaults for all requests. The SUMDUMP option of SDUMP provides a summary dump within an SVC dump. There is a twofold purpose for this. First, since dump requests from disabled, locked, or SRB-mode routines cannot be handled by SVC dump immediately, system activity destroys much useful diagnostic data. With SUMDUMP, copies of selected data areas are saved at the time of the request and then included in the SVC dump when it is taken. Second, SUMDUMP provides a means of dumping many predefined data areas simply by specifying one option.

The data areas saved in SUMDUMP can be printed out by using the AMDPRDMP control statement SUMDUMP. This summary dump data is not mixed with the SVC dump because in most cases it is chronologically out of step. Instead, each data area selected in the summary dump is separately formatted and identified.

For information on print dump statements needed to print the summary dump, and multiple address-space output from SVC dump, see *SPL: Service Aids.*

The RTM2WA pointed to by the TCB upon whose behalf the dump is being taken is the most valid system status indicator available. The dump task is usually the current task; the task upon whose behalf the dump is being taken will contain a completion code in the TCB completion code field. It is possible for the ESTAE routine to issue SVC D itself, in which case the current task is also the failing task.

Because of MVS recovery (retry and percolation), the SVC dump may be only part of the documentation at the problem solver's disposal. The problem solver should attempt to obtain:

1. The system log for the time the dump was taken to ascertain if:

   - Any other SVC dumps were taken before or after the one he is investigating.

   - Any task subsequently abended. If so, a system dump that displays other areas of storage that have meaningful data may be available.

2. The LOGREC formatted listing for the time immediately preceding the time of the SVC dump. If the component analysis procedure fails to determine the cause of the problem, analyze the dump as you would a stand-alone dump. Keep in mind that the information obtained via the CPUDATA option on AMDPRDMP is probably meaningless. Refer to the "Global System Analysis" chapter in Section 2 for information on how to do a task analysis of available address-space-related control blocks.

Keep in mind that the system has detected the error and has attempted recovery, at least on a system basis. Therefore, there will be a good indication of the type (internal symptom) of error (loop, abend, problem check, etc.) that caused the problem. (See Section 4, "Symptom Analysis Approach.")

How to Change the Contents of an SVC Dump Issued by an Individual Recovery Routine

At times, SVC dump contents are not sufficient to solve a problem. The most convenient way to change the contents is the CHNGDUMP command. It can be used to establish system options to be added to the options on each SDUMP request, or to totally override the SDUMP options. See "Using the CHNGDUMP Command" in Section 2. If you do not want to affect all SVC dumps or if storage lists are involved, you may want to change the parameter list in a particular ESTAE exit instead.

You can usually find the name of the recovery routine by looking at the user data (or title) on the SVC dump printout. If not, search the ESTAE's PRB for the virtual address of the SDUMP SVC instruction.

The following description of SDUMP's parameter list can help you decide which bits will provide the data you want. The SDUMP macro expansion generates the parameter list and puts the address of the list in register 1.

SDUMP Parameter List

Offset		
0	1...	user-supplied DCB=
	.1..	BUFFER=YES
	..1.	user-specified STORAGE= or LIST=
	...1	user-specified HDR= or HDRAD=
	.... 1...	user-specified ECB=
	.... .1..	user-specified ASID=
	.... ..1.	QUIESCE=YES
	1	BRANCH=YES
1	1...	indicates SDUMP (as opposed to SNAP)
	.1..	indicates a SYSMDUMP request
	..1.	indicates the MVS/SP level of the SDUMP macro expansion
	...1	user-specified ASIDLIST=
	.... 1...	user-specified SUMLIST=
	.... .1..	ignore the change dump options (used by SLIP)
	.... ..1.	dump came from TSO user
	1	parameter list applies for MVS/SP

2		SDATA options
	1...	ALLPSA
	.1..	PSA
	..1.	NUC
	...1	SQA
	.... 1...	LSQA
	.... .1..	RGN
	.... ..1.	LPA
	1	TRT (MVS trace table, master trace table, GTF buffers)

Offset

3		more SDATA options
	1...	CSA
	.1..	SWA
	..1.	SUMDUMP
	...1	NOSUMDUMP
	.... 1...	NOALLPSA
	.... .1..	NOSQA
	others	reserved
4		DCB address
8		address of storage list (STORAGE, LIST, LISTA)
C		address of header record (HDR, HDRAD)
10		address of ECB
14		caller's ASID
16		target ASID of scheduled dump
18		address of ASID list (ASIDLST)
1C		address of summary dump storage list (SUMLIST/SUMLSTA)
20		address of SYSMDUMP 4K SQA area (or TSO USERID if the DUMP command was from TSO)
24		address of SYSMDUMP CSA work area
28		SDUMP control flags
	1...	LISTA option specified
	.1..	SUMLSTA option specified
	..1.	SUSPEND = YES option specified
	others	reserved
29		reserved
2A		TYPE = parameter options
	1...	TYPE = XMEM specified
	.1..	TYPE = XMEME specified
	others	reserved
2B		reserved
2C		exit options
	1...	GRSQ data requested
	others	reserved
2D		reserved
2E		reserved

SYSABENDs, SYSMDUMPs, and SYSUDUMPs

SYSABENDs, SYSMDUMPs, and SYSUDUMPs are produced by the system when a job abnormally terminates and a SYSABEND, SYSMDUMP, or SYSUDUMP DD statement was included in the JCL for the terminating step. In an MVS system, the output produced is dependent on parameters supplied in the SYS1.PARMLIB members IEAABD00, IEADMR00, and IEADMP00 for SYSABENDs, SYSMDUMPs, and SYSUDUMPs, respectively. See *SPL: Initialization and Tuning Guide* for the IBM-supplied defaults and options that are available.

If the IBM defaults are used, a hexadecimal dump of LSQA is produced when the SYSABEND DD statement is specified. MVS systems do not dump the nucleus or SQA as a default for SYSABEND or SYSUDUMPs. SYSMDUMP defaults include NUC and SQA.

With a SYSABEND, SYSMDUMP, or SYSUDUMP, the system has detected the error and therefore provided a starting point (such as a job step completion code) for analysis. The analyst should always look at the JCL and allocation messages that accompany the dump. The allocation messages contain error messages that can sometimes be helpful. There will also be a JES2 job log that shows the operator messages and responses that relate to the job. The error messages also contain valuable information about the error and should always be investigated.

SYSABEND, SYSMDUMP, and SYSUDUMP errors can generally be divided into two categories: software-detected errors and hardware-detected errors.

Software-Detected Errors

Software-detected errors are those in which one or more of the following occurs:

- A module detects an invalid control block queue.
- A called module returns with a bad return code.
- A program check occurs in system code and a recovery routine changes the program check to a completion code and abnormally terminates the task.

The best approach for a software-detected error is:

1. Use the JES2 job log and allocation messages to investigate all error messages produced. (Refer to the appropriate *Message* manual to determine the causes and corrective action of each message.)

2. Check the abend code defined in the dump. (Refer to *Message Library: System Codes* to determine causes and corrective actions of the code.) Some abend codes define problem determination areas that can be used to help define the problem.

3. In the event that sufficient data is not available in the *Messages* and *Codes* manuals to resolve the problem, the analyst can go directly to the program listing. The diagnostic sections of most PLMs contain a message/module and

abend/module cross-reference. Once the correct module has been located, the program list (supplied in the system microfiche) helps to define the problem.

SYSABENDs, SYSMDUMPs, and SYSUDUMPs normally do not produce system-related data areas other than those which are formatted. Because of this and the fact that error recovery will attempt to reconstruct invalid control block chains before terminating the task, any error that does not occur in the private area may be difficult to resolve from a SYSABEND, SYSMDUMP, or SYSUDUMP alone.

Because of the recovery and percolation aspects of MVS, the SYSABEND, SYSMDUMP, or SYSUDUMP could be the end result of an earlier system error. If so, the analyst should determine if any LOGREC entries were made pertaining to this task and if any SVC dumps were taken while this task was running. The system error is normally reflected in either the LOGREC entries, the dump data sets, or both.

Hardware-Detected Errors

A hardware-detected error is a program check that is not interrupted by a recovery routine. This is identified by a system completion code of X'0Cx' where x is the program check type. For this type of error, the analyst needs to know the address of the module where the program check occurred, and the register contents when the program check occurred. The best place to locate this information is in the RTM2WA that is pointed to by the abending TCB.

Given the registers and PSW at the time of the error, the analyst should determine the module that program checked by using the load list link edit maps of the program. (If the module is outside the private area, a NUCMAP or LPA map may be necessary.) Then he should examine the program listing for the module until the cause of the program check is defined.

Section 4. Symptom Analysis Approach

This section describes how to identify correctly an external symptom, and provides an analysis procedure for determining what kind of problem is causing the symptom.

Each external symptom is described in a separate chapter, as follows:

- Waits
- Loops
- TP Problems
- Performance Degradation
- Incorrect Output

Waits

Wait states may be either disabled or enabled. The characteristics of and an analysis approach for each type are described below.

Note: Be aware that hardware problems, such as waiting on I/O or the timer, can be the cause of enabled waits. Use the information in these topics if you feel that the cause of the wait is a software problem.

Characteristics of Disabled Waits

Situations can develop during execution of the MVS system that require the software to abruptly terminate the system by loading a disabled PSW with the wait bit set to 1. In previous systems, this occurred much more frequently than it does in MVS because, in MVS, many of these situations were removed from the code and replaced with software error recovery. However, a few cases still remain that cause this symptom. To understand these situations better, refer to the 'Wait State Codes' section of *Message Library: System Codes.*

A more critical situation for the analyst is a disabled wait that is caused when data areas containing PSWs referenced by the dispatcher, RTM, or hardware are overlaid and subsequently fetched for use in an LPSW, or for interrupt processing (interrupt new PSWs). This might occur when a PSA overlay condition exists, that is, these PSWs have been inadvertently overlaid by a program running in supervisor state key 0. Other data areas, such as PRBs, may contain PSWs used by the dispatcher and are also potential sources of the disabled wait state. Loading of bad PSWs are difficult to track down. Low address protection prevents inadvertent destruction of the PSWs used by the hardware to give control to the first level interrupt handlers. Bytes 0 through 511 of storage are protected by low address protection which should eliminate many occurrences of invalid disabled wait conditions. See "Low Storage Overlays" in Section 2. The most common MVS uses of the LPSW are:

- hardware loading from low storage for an interruption-processing sequence
- dispatcher loading from fields X'420' and X'468' in the PSA. The PSW loaded by the dispatcher may have come from an RB or an SSRB.
- RTM (IEAVTRTS) passing control to FRRs
- the system termination routine
- I/O and external FLIH's LPSW to resume task or SRB.

Storage overlays resulting in wait state PSWs are approached in the same manner as other storage overlays. The important step is to realize the storage overlay has occurred, then re-create the process that was possibly responsible. The discussion of pattern recognition in the chapter "Miscellaneous Debugging Hints" in Section 2 should be helpful.

Analysis Approach For Disabled Waits

The following is a list of objectives that provides a systematic approach to analyzing a disabled wait.

Objective 1 - Determine positively that an actual disabled wait condition exists. Is the PSW the type that is used when MVS loads an explicit wait or is this an overlaid PSW with the wait bit on?

Analysis - Examine the *current* PSW contained in the dump according to the technique described in the chapter "Stand-alone Dumps" in Section 3. The PSA overlay should also be analyzed to determine if key PSWs have been overlaid.

If the PSW shows an explicit wait, look up the wait state in *Message Library: System Codes* to find what conditions could cause the explicit wait. You may need to do some extra analyzing before the condition can be related to a component. (*Note:* No further analysis for explicit wait situations is discussed in this book.)

If the PSW suggests an overlaid PSA or some other error source, proceed to Objective 3; otherwise proceed to Objective 2.

Objective 2 - Determine if the situation has been improperly diagnosed as a disabled wait. This will eliminate a situation in which the locked console is diagnosed as a disabled wait.

Analysis - In previous operating systems, the operator's inability to communicate with the system through the console was an external indication of a disabled wait condition. In MVS, this same external symptom is often not a true disabled wait. Console communication is dependent upon other services of the operating system, such as paging, and the I/O subsystem. A problem in any of these services often terminates console activity and causes an apparent "disabled wait" situation, when the PSW does not actually reflect a disabled wait.

If the current PSW is not disabled for external and I/O interrupts or if the wait bit (X'0002') in the PSW is not set to one (PSW=X'070E0000 00000000'), you should proceed to either the "Enabled Wait Analysis" topic later in this chapter or to the chapter on "Loops" later in this section.

Objective 3 - Once you know that the disabled PSW is the result of an overlay in low storage or in another data area, you must gather specific data about the overlay. Ask such questions as: What was the damage to the PSW? When did the overlay most likely occur? Where did the PSW come from?

Analysis - It is important to try to find out how the PSW was overlaid - was it a byte, an entire word or doubleword, a single bit, or was a large portion of the surrounding area destroyed along with the PSW? (The discussion of Pattern Recognition in the chapter "Miscellaneous Debugging Hints" in Section 2 will help you determine this.) Much of this analysis depends on your experience and familiarity with the normal data for the subject PSW and the surrounding area. You should try to gather enough data to know, for example, that "n" bytes were overlaid beginning at location xyz.

Also, examine the trace table, if available, and try to determine when the PSW was probably last valid. Look for interrupts and unusual conditions in the trace entries to try to reconstruct the process(es) leading up to the incorrect PSW.

If the trace indicates the overlay occurred *after* the most recent trace entry, the registers are important because they may show recent BALs and BALRs and they may contain the address of a routine or control block that was used to overlay the subject PSW. This is actually a good situation because it will not take long to relate the overlay to some bad pointer in a control block and, hopefully, your analysis will proceed to a specific component.

If the overlay occurred several trace entries earlier, determine a possible save area that might contain the registers that were active at the time of the overlay by examining interrupt entries or dispatch entries in the trace table.

If there is no trace table, it is almost impossible to define when the overlay occurred. You might try to analyze, for example, TCB save areas, hoping for a clue as to when the overlay occurred and to gather information concerning the problem. However, this process is basically undefined and undisciplined. In most cases, a trap for the overlay can be generated at this point and used as soon as possible.

Objective 4 - Determine which component most likely caused the overlay and choose a likely set of modules from that component to analyze at an instruction level. Determine which data area field contains the bad address and who set up the field.

Analysis - As mentioned earlier, by using the registers and trace table it is possible to identify which code actually overlaid the PSW, but the source of the error must still be found. This mostly involves screening code to reconstruct the path which caused the overlay and locating the data that generated the bad address. At this point, you want to learn which module set the bad field so you can start backtracking.

Shortcuts are possible according to the analyst's familiarity with the modules that are involved. Certainly the main objective should be to decide which component is most likely responsible and then to proceed to the discussion of that component's analysis (in Section 5).

Characteristics of Enabled Waits

Enabled waits have traditionally been the most difficult problem to analyze because of the lack of an obvious failure. The enabled wait provides no indication of error other than that the system apparently has nothing to do. In fact the enabled wait has been accurately described as an end symptom of a problem with no obvious causes. The task of determining the possible cause is left to the debugger. Other types of software failures - abends, program checks, loops, messages - provide a starting point for analysis; that is, software or hardware has indicated a violation of interfaces or data integrity and has halted the erroneous process at the point of error. The enabled wait provides none of these.

Note: The subsystem design of many components includes a dispatching mechanism and internal control block structure not generally recognized by the operating system. When these subsystems (for example, VTAM, TCAM, JES2) malfunction, work through these components is often halted. Because of the critical nature of these processes, external signs of the problem are often detectable. Within this debugging discussion, these problems are often treated as wait states, that is, the system may be capable of running batch work, but the TP network appears "hung-up." This general discussion of analysis-approach applies for problems such as "permanently" swapped-out address spaces, TP network hung, and no batch running. The advantage is that the external symptoms may allow you to more easily isolate the problem component or at least a starting point - it may be obvious that TCAM is not responding, or that JES2 is not processing input.

Experience has shown that in MVS a much greater percentage of re-IPL situations are caused by enabled waits than in previous systems. One reason for this characteristic of MVS is software recovery. Software recovery attempts to repair the damage caused by a failure and allow the system to continue meaningful operation. The general philosophy of recovery is to isolate the error to a job, terminate the job, and allow the system to continue. This philosophy dictates that under certain conditions innocent work may be forcefully terminated.

Software recovery obviously may cause the termination of some critical process which in turn causes dependent processes to wait indefinitely. For example, assume that while processing a page-fault, an error occurred during the I/O interruption processing; software recovery was invoked and subsequently caused a cleanup of the bad control blocks, but did not post the I/O requestor. It is possible that the paging mechanism will wait indefinitely for the missing interrupt. This in turn could cause a problem program to wait indefinitely for the paging operation to complete. The end result is no work accomplished and also no external problem symptom, although a problem clearly exists. The debugger must find the bottleneck - the paging exception - and subsequently back-track enough to determine why the bottleneck still exists. Very often, this back-tracking requires analysis of several components in order to determine the original cause.

Analysis Approach For Enabled Waits

It is most important that you understand the actions that must take place in order to accomplish work in the operating system. This requires a basic understanding of the key system processes in MVS - paging, I/O, dispatching, locking, WAIT/POST, ENQ/DEQ, VTAM, TCAM, SRM, JES2/3. These areas of the system are responsible for directing work through MVS; a malfunction in any one may cause global system problems. Several, if not all, must be investigated in order to determine why work is not progressing.

This investigation requires a disciplined approach. The relationships of component interfaces and their mutual dependencies must be understood. With this in mind, the debugger should proceed to gather information about the various processes and try to integrate his findings with his other information and assumptions about the problem, always trying to isolate one cause of the bottleneck. He must avoid the tendency to guess, assume, and go off on tangents once the first irregular item is uncovered. Instead, he should continue to gather known facts and piece them together in some logical pattern that recreates the situation.

In the majority of wait state cases, more than one key process will appear backlogged. The challenge is to determine how these problem processes relate and which is the fundamental cause of the wait situation. After you gather the facts and understand the bottlenecks, you must answer one question. If I "pull the cork" on this given bottleneck will all the other intertwined situations resolve themselves? In *every* problem there is only one bottleneck for which the answer to this question is "yes." The other problems are consequences of this key process's failure to complete its designed function. Isolating the process is half your battle; the other half is determining the cause of this *one* process's failure.

Following is a suggested disciplined approach for the problem solver who is approaching a system wait problem. The approach involves three distinct stages of problem analysis:

Stage 1 - Preliminary global system understanding, including

- system externals
- current system state
- LOGREC analysis
- trace analysis
- determining the reason for waiting

Stage 2 - Key subsystem analysis - an in-depth analysis of the MVS components that are responsible for accomplishing work.

Stage 3 - System analysis - using the information gathered in Stages 1 and 2 the problem solver must "step back," get perspective about the known facts by piecing them together in a logical fashion, and isolate the error to a process, component, module, etc.

This approach is described in detail in the following sections.

Stage 1: Preliminary Global System Analysis

1. ***System Externals*** - Completely understand the system externals of the situation. Console sheets and the system log should be inspected.

   - For any enabled wait (operators call it "system hung") find out if a display requests command was issued. (Lack of operator action can cause system bottlenecks.)

   - Often many pages of console sheets must be investigated to uncover operational problems and explain events uncovered in the dump. Scanning provides a feeling for the events, jobs, requests, etc. leading up to the problem.

   - Make sure all DDR SWAP requests, I/O error messages, SQA shortage messages, etc. can be explained.

     Always take the time to examine these external areas because a small effort here could save many hours of detailed dump analysis. Do not overlook obvious items such as a MOUNT PENDING message in the console log that can cause system problems.

   - If an IPL for an alternate nucleus was executed, make sure that the correct nucleus was specified when responding to message IEA101A SPECIFY SYSTEM PARAMETERS. Additionally, CVTNUCLS can be checked to verify that the IPL was for the correct nucleus.

CVTEXT2(CVT+X'148') points to the CVT extension. CVTNUCLS is located at offset X'04' in the CVT extension.

2. ***Current System State*** - Investigate fully the current situation as depicted by the dump.

   For enabled waits, the PSW should equal X'070E000000000000' (often called the "no-work" wait). The last entry should be the wait or there should be a considerable recurrence of the no-work wait in the trace table - see the chapter on "MVS Trace Analysis" in Section 2. If this is not the case, use the disabled wait analysis approach (earlier in this chapter).

   If the PSW indicates the no-work wait situation, you have an enabled wait. You should now check other global system data areas indicators to get the whole picture. Following are key global indicators:

   - PSASUPER field (PSA+X'228') should have the dispatcher super bit (PSADISP) set. PSADISP is left on when wait is dispatched and will be turned off on the next I/O or external interrupt. If any other bits are set, some supervisor routine is in control. This situation can indicate incomplete processing by the associated routine. All possibilities should be pursued until the situation can be explained.

   - Because of SRM timer/analysis processing, even when the system is in the enabled wait situation, the state of the processor at the very instant the dump was taken can indicate, via the "super bits" or locks indicator (PSACLHS), that some process was occurring. You must determine in this case that these fields being set is normal and continue with wait analysis. If the fields cannot be explained, you have isolated the error.

   - There should be no locks held, as indicated by PSACLHS on either processor. This situation is similar to the one described just above. You must try to discover the owner of the lock and determine why it is still held despite the fact that the system is waiting. Often the purpose of the lock will provide insight as to who the owner might be. The chapter on "Locking" in Section 2 should be of help in your analysis.

3. ***LOGREC Analysis*** - Determine if key components have encountered difficulty; determine previous errors encountered by the system. This can be accomplished by inspecting SYS1.LOGREC as well as the in-storage LOGREC buffer. Errors encountered in any of the key processes noted earlier (RSM, ASM, IOS, JES2/3, SRM, ENQ/DEQ, VTAM, etc.) may provide further information. If you do find an error associated with any of these areas, determine whether it could lead to the bottleneck.

   The LOGREC records generally contain the names of the error-encountering routines and often the job on whose behalf the system was processing at the time of the error. If the routine names are not present, you may have to use system maps and the PSW/register information in the LOGREC records in order to associate errors with components. The discussion of LOGREC analysis in the "Use of Recovery Work Areas" chapter in Section 2 should be helpful in your analysis.

4. ***Trace Analysis*** - Determine the last activity within the system.

   Because of SRM's timer processing, the trace table for most wait conditions is not useful. However, on the rare occasion that the system has been stopped or if for some reason the trace is not overlaid with timer interrupts (X'1004' external interrupt entries), the trace should be analyzed to ensure normal processing, for example, page faults are being processed, I/O is being accomplished. Be suspicious of large (relative to most entries) time gaps in the trace table. If the table has not wrapped-around, process re-creation may be of some use in determining what the system was doing up to the point of incident. (The chapter on "MVS Trace Analysis" in Section 2 should be helpful.)

5. ***Determine the reason for waiting*** - Once it has been determined that the system is waiting, it is always useful to determine what the various address spaces or jobs are waiting for. This is accomplished by inspecting and scanning the various tasks and their associated RB structure in a formatted stand-alone dump. Remember the RCT, started task control (STC)/LOGON, and dump task may all be waiting in each address space - this is normal. The question you should ask is: Why are the subtasks below the STC/LOGON waiting?

   Generally in an active system more than one address space will be waiting for the same or similar resource in a problem situation. Therefore, as you scan and analyze address space status, look for suspensions in common modules (RB resume PSWs containing similar addresses):

   - Many tasks in page-fault wait can indicate the paging or I/O mechanism is faulty.

   - The PVT can indicate a real frame shortage.

   - Many tasks in terminal I/O wait can indicate something is wrong with the TP access method or some part of the network.

   - Several Resume PSWs pointing into the ENQ/DEQ routine, IEAVENQ1, can indicate an ENQ resource contention problem.

   In general, be on the look-out. Try to compare and relate the system activities as you encounter them. Often more than one process or address space is held up because of a common bottleneck. It may be a global resource required by more than one address space, for example, a lock or data set. It is important that the *exact* cause be determined.

Stage 2: Key Subsystem Analysis

As part of this investigation, if nothing can be easily determined from a cursory address space scan, you may have to delve into the key components. Following are some highlights of the important and potentially suspect areas:

1. ***I/O Subsystem*** - Check for unprocessed I/O requests, bottlenecks in the I/O process will almost always log-jam the system. Since IOS is the central facility for controlling I/O operations, I/O problems should always be suspected in an enabled wait condition. Therefore, the IOS component and

its associated queues should be analyzed early in the subsystem analysis stage of debugging. Two important IOS queues and control blocks will indicate whether problems exist in the I/O process:

- Logical channel queues (LCH) contain lists of elements for I/O requests. If these queues (pointed to by the CVT+X'8C') are not empty in a waiting system, IOS must be further investigated.

- Unit control blocks (UCBs) are a logical representation of each I/O device containing I/O active indicators at offset 6/7. If any indicator is set, this device must be further investigated. This condition can indicate either a hardware or software problem.

Both the queued (LCH) and active (UCB/IOQE) requests must be further investigated to determine the associated requestors and what effect their I/O not being serviced will have on system operation (for example, if paging I/O or console I/O is not being serviced, the system will usually stop).

The UCB contains indicators for DDR, intervention required, and missing interrupt handler processing. Any such indication must be further investigated.

An ENQ on the SYSZEC16 resource is an indication of a waiting condition generally associated with swapping. The swapping process cannot complete until active I/O finishes. In a quiesced system, an ENQ on this resource must be further investigated.

2. ***Paging Mechanism*** - Check for unserviced page faults. ASM, RSM, and SRM are closely related and depend upon each other to maintain real storage, the swapping process, and page fault resolution. If, when you determined the reason for waiting as described in stage 1, you discovered several page fault wait conditions, be suspicious. Some key indicators in determining page fault waits are:

   - ASCBLOCK=X'7FFFFFFF' - indicates suspension while holding a local lock. If in task mode at the time of suspension, the resume PSW instruction address (saved at IHSA+X'10' of the locally locked address space) should be checked. When the instruction address = RBRTRAN(X'C' offset) and RBXWAIT = 0, it indicates the task is suspended while it waits for a page fault resolution. Note that a data reference page fault might have occurred; in this case, the instruction address does not equal RBTRAN. RBTRAN may be checked against the register and displacement as determined from the page-faulting instruction. The page fault occurred when a new module (paged-out) was referenced. If in SRB mode at the time of suspension, an SSRB will be queued from a PCB. The anchor for these PCBs is the RSMHDR (private area page fault) or the PVT (common area page fault).

   - ASCBLOCK=X'00000000' - indicates the local lock for this address space is not held. The RB structure can reveal the same situation as described above for RBOPSW instruction address = RBRTRAN, RBXWAIT=0 *and*, in addition, an RB wait count =1. Note that a data reference page fault might have occurred; in this case, the instruction address does not equal RBTRAN. RBTRAN may be checked against the register and displacement as determined from the page-faulting instruction. If you find several tasks in this state, check the dump for the page represented by RBTRAN. Is it in storage? (Remember for private area addresses to be sure that the address space you are investigating is printed.) If the page is not in storage you may have a potential paging problem. Again, if in SRB mode at page fault time, the SSRB must be found to determine more about the process.

If you suspect that paging is a potential problem in the system, several SRM and RSM/ASM control block fields can be examined. Periodically, SRM calculates the total paging rate (swap, VIO, and demand), the demand paging rate (separately), and the average maximum unreferenced interval count (UIC). These values are saved in the resource management control table (RCT) for use by the algorithms that adjust the multiprogramming level (MPL) of the domains. The RCT can be found by locating the SRM control table (RMCT) which is pointed to by the CVT (field CVTOPCTP at offset X'25C'). The RCT, like many other SRM control blocks, follows the RMCT and can be located by examining the program listing for module IRARMCNS. The total paging rate per second is identified by field name RCVPAGRT. The demand paging rate is identified by field name RCVDPR. The average maximum UIC is identified by field name RCVUICA. Two other fields of interest in the RCT are RCVASMQA, which is the average number of requests in progress or queued by the ASM, and RCVMSPP, which is the average time in milliseconds for the ASM to process a page request. Also examine the page vector table (PVT), which can be found from the CVT (field CVTPVTP at offset X'164'). The available frame count is contained in bytes 2 and 3 of the PVT. Previously a low available frame count (less than 15-20) was an indication that storage contention or paging was a problem. The SRM attempts to better utilize the real storage resource and a low available frame count is not uncommon, although it can still be a source of concern. The combination of low available frame count, a low UIC, and a high demand paging rate indicate storage contention great enough to adversely affect response time and throughput.

ASM maintains a count of the number of paging requests received and the number for which processing has completed in the ASMVT. If these counts are not equal, ASM is backed-up and page faults have not been resolved. This can be caused by an I/O problem or some internal ASM problem. The ASM Component Analysis chapter in Section 5 describes the work queues in the paging activity reference table entries (PARTEs). Finding unprocessed work on these queues will aid in determining whether ASM is the problem component. But again be careful: you are still gathering data about the wait state. Your purpose now is not to debug ASM - it may not be the problem. Note the apparent ASM problems and continue your investigation. Later when you piece together your findings and find the real source of the problem, detailed debugging and logic flow will be required.

3. ***ENQ/DEQ*** - Check for unresolveable resource contention. Finding an ENQ/DEQ interlock and determining what work is being held up because of this interlock can provide important information about the overall problem. The QCBTRACE, Q, or GRSTRACE option of AMDPRDMP provides a formatted structure of the resources and the work that is in contention for them. Determining who owns the resources and the current status of the

owners (if swapped-out, why? or if waiting, for what?) often provides important clues in understanding the bottleneck.

Also in your scanning process, you should be on the alert for address spaces that contain subtasks (usually below the STC/LOGON level) with multiple RB levels, and with the lowest RB containing a resume PSW with an address somewhere within ENQ code (nucleus resident) and with the RB wait count RBWCF=1. The previous RB should be an RB with the ENQ SVC (SVC X'38') indication in the "WC-L-IC" portion of the RB prefix (-4 offset). This indicates that this task and probably the address space are suspended because of an unsatisfied ENQ request. If several address spaces or tasks are found in this state you should find out why. The QCBTRACE, Q, or GRSTRACE facility of AMDPRDMP can be most helpful. An illustration follows:

Investigation of QCBTRACE, Q, or GRSTRACE data shows many requests backed-up on resource A. The analyst notes this and determines what ASID or TCB owns resource A at this time (in this example, ASID 9). The other resources represented in the QCBTRACE, Q, or GRSTRACE are now scanned. If ASID 9 is backed-up behind someone else (ASID 10) waiting for another resource (B), you must now determine ASID 10's status with respect to other resources, including resource A. Essentially you are looking for cases where:

- An address space has resource A and is waiting for resource B *and* a second address space has resource B and is waiting for resource A. This indicates a deadlock. You must determine the faulty process. In this case you have probably isolated the error to the ENQ process and the way it is being used. You must analyze the task structure of each address space to determine how this situation occurred. Do not forget the SYS1.LOGREC buffers. They may contain clues like errors in ENQ/DEQ or one of the tied-up address spaces (jobs). Faulty recovery should be suspected if the latter is the case.

  It may be that a job requests control (via ENQ) of a resource and subsequently encounters a software error. The task's associated recovery gains control and "recovers" from the error but does not dequeue (DEQ), and therefore does not release the resource. Eventually, the contention for this resource, depending on its importance, could cause severe problems.

- An address space has control of a resource and a lot of address spaces are queued-up behind this address space. In this case, you must find out why the holder is not releasing the resource. Also know your system. It is not unusual to see activity on the master catalog resource: "SYSIGGV1 - Master Catalog Name." But be suspicious of most resources. Determine from the holder's task structure what process it is attempting. Determine whether the address space is waiting or swapped-out and why. If it is not waiting or swapped-out, check the non-dispatchability bits and the possibility that the address space is looping.

  This second case is much more likely to be a sign of some other system problem. Your clue is what is preventing the holder's execution; this will point you to another process which *must* be investigated and may lead to the detection of the final problem.

Note: When analyzing a dump of a quiesced system you should be suspicious of "unusual" ENQ resource names - resources that should not be a contention factor in a quiesced system. The presence of these names should be understood and explained because they very often will point you to the problem area. Common resource names are:

SYSZEC16 - PURGE	Can indicate a problem in the I/O process related to the resource holders address space Can indicate a bottleneck in the swapping process
SYSZVARY - x	indicates the reconfiguration component has been invoked - why is it not completing?

4. ***Dispatching*** - Determine if there is work to do in the system. A common trouble indicator is an MVS dispatching queue containing elements that indicate work is ready to execute in a waiting system. The SVTGSMQ, SVTGSPL, SVTLSMQ, and each ASCBLSMQ and ASCBLSPL should be empty. (The chapter on "System Modes and Status Saving" in Section 2 contains details of these queues and how to find them.) Generally it is not a problem in the dispatching mechanism itself but merely an error indication. Often the most useful information is just that 'yes, there is work.' Why is it not being dispatched? Is there a problem in some other area of the system? Is the address space swapped out? Yes, there may be a real storage problem delaying swap-in. Or perhaps SRM has not been told to swap-in the address space via a "user-ready" SYSEVENT. (The ASCBURR bit indicates that a "user-ready" SYSEVENT is required). In summary, investigate the OUCB for the address space you are concerned with.

   Another useful point is to find out what problems could arise if this work were not dispatched. Investigating the queued work will indicate what would be accomplished if this work were executed. This is usually important because it can clear up much of the "smoke" you may be encountering in your overall system investigation.

   Likewise, investigate the task structure. Generally, you can ask the same questions as above, but you must look in different places for the key indicators. Among the most important indicators are:

   - The ASCB, which contains two counts of ready TCBs in the address space: ASCBTCBL, which is a count of ready TCBs that require the local lock; and ASCBTCBS, which is a count of ready TCBs that do not require the local lock.

   - The TCB non-dispatchability flags.

   - The RTM work area, which contains status at time of error.

   - The RB structure. Look for long RB chains or unusual SVCs and interrupt codes. Look for page fault waits.

   Again, use this information to lead you to processes or problems that hold-up the system.

5. ***Locking*** - Determine if there is a locking conflict. The locking mechanism causes system bottlenecks when it is not used properly. The global spin locks cause obvious problem symptoms such as one processor spinning in the lock manager (IEAVELK) in an MP environment. (In a UP environment, global spin locks are generally not a problem unless a lock word or interface is overlaid or bad, causing a disabled spin. The enabled suspend locks (local/cross memory services) are generally the problem ones.) The chapter "Locking" in Section 2 describes in detail the considerations with which you should be concerned. Elements on the local/cross memory services suspend queues may indicate a problem. The technique you adopt to resolve the conflicts is exactly the same as the ENQ interlock or logjam situation.

6. ***Teleprocessing*** - Determine if the TP network is responding. Problems in the TP network often manifest themselves as waiting network or waiting terminals, even waiting systems. The chapter "TP Problems" in Section 4 contains a detailed description of TP problem analysis. The VTAM chapter in Section 5 contains techniques for VTAM problem analysis.

   An important fact for the problem solver here is that these are subsystems. As such, they maintain their own control blocks, queues, and dispatching mechanisms. They are responsible for work being processed once it enters the subsystem and they often have little direct dependency on MVS. That is, normal MVS problem indicators will not generally solve the problem. You must understand the subsystem's work-processing mechanism in order to be an effective analyst. For example, VTAM has its own address space with a number of tasks used primarily for network start-up, shut-down, and operator commands. In most VTAM problems, a look at the VTAM address space will show these tasks are waiting. However, this is normal when no operator processing is required. Even though VTAM is waiting, this is not the place to be distracted. Again, remember this VTAM task structure, put it aside as part of your information gathering, and then proceed to the analysis of VTAM's internal work queues as described in the VTAM chapter of Section 5.

7. ***Console Communications*** - Determine whether console communication is possible. The system can appear or actually prove to be waiting because the operator is not able to communicate with MVS. This could be the sign of a problem almost anywhere in MVS, but it often indicates an error in the communications task or its associated processing.

   The communications task (comm task) runs as a task in the master scheduler's address space and is usually represented by the third TCB in the formatted portion of the stand-alone dump and identified by a X'FD' in the TCBTID field (TCB+X'EE'). By inspecting the RB structure associated with this task, you can determine the current status. It is not unusual to find one RB with a resume PSW address in the LPA and an RB wait count of one. If more than one RB is chained from the TCB and you were not able to enter commands, analyze the RB structure because this is not a normal condition.

The key control block is the unit control module (UCM) which is located in the nucleus. CVTCUCB (CVT+X'64') points to the base UCM. The base UCM-4 contains the address of the UCM MCS prefix and the base UCM-8 contains the address of the UCM extension. From the UCM you can determine the status of the various consoles. The following should be considered and can warrant further investigation:

- Important WTORs are outstanding.
- An out-of-buffer (WQEs, OREs) situation exists.
- There are unusual flags in the UCM.
- There is a full-screen condition.
- There is a console out of ready.

Remember that comm task processing is dependent on the rest of the operating system. Most likely, some external service or process has caused comm task to back-up, and this possibility should be investigated. Remember the debug process: gather all the facts, then proceed with analysis.

Stage 3: System Analysis

At this point you should have a detailed understanding of the system and its key components. You should know which components or processes are back-logged and, correspondingly, what work (jobs) is not being processed by the system because of these back-logs. You must now stand back from the problem.

Answer this question: Which of these problems and situations can be related to or attributed to each other? For example, if I/O is queued for the paging devices (indicated by IOQEs on the LCHs associated with the paging devices' UCBs) and you also found several address spaces are in "page-fault wait," you can now relate these findings. And if one of these address spaces performed an ENQ for a resource and did not yet DEQ because of the page-fault suspension, it is very likely other address spaces are also backlogged behind this job's processing. Initially your ENQ/DEQ analysis showed the problem, but at this point you can attribute the ENQ contention problem to the page-fault suspension problem that you have already attributed to the I/O problem.

This process must be repeated for all the potential error situations you uncovered in your investigation. Do not forget to use the system indicators in your attempt to arrive at the source of the problem. And most importantly, ask yourself: If I unplug this bottleneck, will all the other intertwined situations resolve themselves? In the previous example, resolving the ENQ situation *will* allow the work queued in the ENQ/DEQ component to execute *but* the "page-fault waiting" job will still be hung. That is, ENQ/DEQ is not the problem to pursue. Indeed, if you resolve the I/O problem, this page fault is resolved, the DEQ will be performed, and *all* work in the system will resume normal operation. Yes, the I/O problem is the important consideration in this case. The I/O problem is the one that must be pursued. When this problem is resolved, the enabled wait state condition has been resolved. Global system areas, recovery work areas, LOGREC analysis, and IOS component analysis will be necessary to further isolate, and eventually solve, the problem.

Loops

Loops are defined as disabled or enabled, depending upon their external appearances. A *disabled* loop can be recognized externally by a solid system light or 100% CPU busy on the system activity display and the inability to communicate with the system through the consoles (that is, no input or output). Usually, a disabled loop indicates a hardware and/or software malfunction. There are several cases in MVS however in which a disabled loop is purposely used and is not an error indication. These cases are discussed later in this chapter.

An *enabled* loop is generally much larger than a disabled loop. Observed from the console it appears as a bottleneck: the system seems to be slowing down periodically, suggesting performance degradation. The operator may notice that a particular job remains in the system for a long time and does not terminate.

Common Loop Situations

There are three common loop situations:

1. Processors of an MP environment communicate via the signal processor (SIGP) instruction. Often the SIGP-issuing processor enters a disabled loop until the receiving processor either accepts the SIGP-caused interrupt or performs the operation requested by the issuing processor. This loop serializes the processors in the MP configuration. The SIGP-issuing processor loops in a nucleus-resident module, IEAVERI.

   Often during an MP dump analysis you will find that one processor was in this loop. This is not an error if:

   - The operator stopped one processor and not others to investigate a suspected problem.

   - The receiving processor is disabled for external interrupts, thereby preventing the SIGP-issuing processor from proceeding.

   If this situation continues for an extended period, it means there is a system problem but the loop is a result of that problem and is not an error itself. Most often, other processors' activities must be analyzed to determine the problem. For a more detailed discussion of MP communication, refer to the chapter "Effects of MP on Problem Analysis" in Section 2.

2. The lock manager (IEAVELK), which resides in the nucleus and controls the locking mechanism of MVS, contains a section of code that enters a disabled loop when a global spin lock is requested but is not available. On a UP this is an invalid condition and always signifies an overlaid lockword or invalid lockword address. On an MP/AP system, this usually indicates that another processor is holding the lock and not releasing it. But it may indicate an overlaid lockword; if not, the problem is definitely on the processor that is not releasing the lock. In either case, register 11 contains the pointer to the requested lockword and PSALKR14 contains the address of the requestor. Check the value in the lockword. Valid values are a fullword of zeros, or three bytes of zeros and the logical processor address in the fourth byte. Any other bit configuration will cause the system to spin in a disabled loop and

signifies an overlaid lockword or invalid lockword address. If the lockword is not valid, it is necessary to identify who overlaid the lockword. It is possible that the lockword was overlaid in conjunction with some other problem. Again, since the disabled loop may not be the problem but a symptom of a possible error on the other processor, determine why the requested lock is not available. For a detailed discussion of "Locking" see Section 2.

3. The intersect service routine (IEAVEINT), which resides in the nucleus, controls the intersect serialization in a manner similar to the lock manager. Intersect provides the serialization between a routine holding either the dispatcher or local lock and the dispatcher (IEAVEDS0). The intersect service routine contains a disabled loop which is entered (for MP or AP systems only) when a routine requests the intersect on one processor and the dispatcher is active on another processor. The dispatcher active field is located via PSA+X'B4C', which points to the SVT. SVT+X'6C' (SVTDACTV) contains one byte per processor and is indexed by the processor address. If the loop in the intersect routine is in control, register 2 contains the physical processor address where the dispatcher is processing, and register 14 contains the return address of the routine requesting the intersect. It is possible that the dispatcher active field is overlaid; its contents should be checked for validity. Each byte of the dispatcher active field corresponds to a possible online processor. For an AP or MP system the last two bytes should always be zero. Other valid contents are logical processor address, or the logical processor address with the high-order bit on.

   Since the disabled loop might not be itself a problem, but a symptom of a possible error, determine how the dispatcher active field became overlaid or why the dispatcher is looping on the other processor.

Analysis Procedure

Generally for loop analysis, you will have a stand-alone dump if the operator considered the problem serious enough to re-IPL the system, or an SVC dump, SYSUDUMP, SYSMDUMP, or SYSABEND (provided by the software recovery) if the operator pressed the RESTART key in order to break the apparent loop. For the SVC dump, SYSUDUMP, SYSMDUMP, and SYSABEND dumps there is an abnormal completion code of X'071' associated with the looping task of a job if the RESTART key was pressed when the program was actually looping. In addition, a formatted SYS1.LOGREC listing should be available.

Note: If the loop recording option is available on your 3033 or 3081 processor, you can record information about the loop by selecting the option on the system control (SC) frame of the 3033 or the CC012 frame of the 3081. This option records several hundred values of the instruction counter in the form of an instruction trace. After recording the instruction trace, the processor is put in the STOP state. The information becomes available when you take a stand-alone dump or when the SVC dump is invoked.

Before you can determine what problem is causing the loop, you must determine first that a loop really exists, and second whether it is enabled or disabled.

First, verify that a loop exists. The *disabled* loop situation is fairly straightforward. The PSW contains a disabled mask (X‘04’ or X‘00’) and all other system activity will have stopped.

Recognizing that there is an *enabled* loop is often the most difficult step. Enabled loops are often quite large and may encompass several distinct operations - I/O events, SVCs, module linkage, etc. Because the loop is enabled, it is often interrupted, preempted and eventually resumed many times. This makes it difficult to recognize the loop pattern. Following are some indicators of a potential enabled loop:

- The current PSW has an enabled mask, X‘07’, in the first byte and the instruction address portion ”0. This alone does not prove there is a loop, but the information may help your analysis of the problem later.

- The MVS trace table shows a repetitive pattern of events, for example, SVCs issued from the same virtual addresses, or dispatcher entries for virtual addresses that are relatively close together. Determine if the entries are related to the same address space by using the ASID field (offset X‘16’ into the trace entry). If so, you can now examine the task and control block structure indicated by the trace entries. The chapter on “MVS Trace Analysis” in Section 2 should prove helpful.

- Many tasks (TCBs) or address spaces (ASIDs) appear to be bottlenecked waiting for some resource(s). This can be determined by using the QCBTRACE or GRSTRACE option for AMDPRDMP and analyzing the output. If there appears to be a bottleneck, determine what job owns the resource(s) and what that job is currently doing. It may be that the job that acquired the resource(s) is in an infinite enabled loop; therefore, when other jobs request the same resource(s), their requests cannot be satisfied, which eventually causes a major performance throughput problem. See the chapter on “System Execution Modes and Status Saving” in Section 2 for how to recreate the job′s current status. A reconstruction of the PSW and registers helps you to determine if there was an enabled loop.

- TCB/RB structure analysis. Look for unusual or long RB structures chained from TCBs. These may indicate a loop that includes several levels of supervisor linkage.

Enabled Loop Exception:

The system resources manager (SRM) of MVS constantly monitors resources, gathers data, and analyzes the system. Periodically, SRM uses a timer interrupt in order to gather its statistics. This timer interrupt occurs even when the system is in an enabled wait condition. Because of this, the enabled wait is often referred to by operators as an enabled loop. (They observe the “WAIT” indicator from the console, followed by a burst of activity (SRM processing), followed by the “WAIT” indicator, etc. It may even be possible to enter certain operator commands.) However, this is really an enabled wait condition and analysis should proceed according to the discussion on “Enabled Waits” in the “Waits” chapter earlier in this section.

The dump you are analyzing may show the MVS trace table containing a no-work wait (070E0000 00000000) PSW followed by a timer interrupt, SRB dispatch, MP communication, etc. This pattern indicates an enabled wait condition, not an enabled loop. (See the “Pattern Recognition” topic in the “Miscellaneous Debugging Hints” chapter in Section 2.)

Once you have determined the type of loop, the following analysis procedure should help determine what problem is causing the loop.

***Objective 1** - Who is looping?*

The PSW and registers saved at the time of the dump indicate the active work. (See the chapter "Global System Analysis" in Section 2.) The register save areas in the LCCA/PSA indicate important environmental data at the time of the last I/O interrupt, external interrupt, etc. (See the chapter "System Execution Modes and Status Saving" in Section 2.)

The PSA indicators contain valid information about disabled loops. Also remember the recovery areas active at the time of the loop are valid and may provide hints as to the current process. (See the chapter "Use of Recovery Work Areas" in Section 2.)

Unlike the disabled loop situation, the enabled loop may not have the current registers associated with it. This is true if the loop was interrupted and new processing was initiated before the dump was taken. For the enabled loop, find the current registers and status from the ASCB/ASXB/TCB/RB structure and the associated save areas (for example, IHSA). The chapter "System Execution Modes and Status Saving" in Section 2 will be helpful for this phase.

***Objective 2** - What is the system mode?*

It is important to know whether the system is in SRB or task mode and the implications of these modes. In all cases of true disabled loops, the PSW, LCCA, and PCCA contain valid status indicators such as the last dispatched routine. The old PSWs reflect the last interrupt status. The register save areas in the LCCA are valid. The LCCA+X'21D' set to 1 indicates SRB mode; set to 0 indicates task mode. Additionally, PSAMODE (PSA+X'49F') indicates the state of the system. If set to X'00', the system is in task mode; if set to X'04', the system is in SRB mode. This byte can also contain flags to indicate wait mode, spin mode, or non-preemptive mode. The ASCB NEW/OLD and TCB NEW/OLD pointers reflect the current task. (*Note:* If the TCB OLD pointer is zero, the system is in SRB mode or possibly in supervisor mode - that is, dispatcher or supervisor recovery. The discussion in the "System Execution Modes and Status Saving" chapter in Section 2 and the "Dispatcher" chapter in Section 5 are useful.

By scanning the MVS trace table, you will be able to determine system events leading up to the loop. See the chapter on "MVS Trace Analysis" in Section 2.

SYS1.LOGREC and the in-storage LOGREC buffer may contain indications of previous occurrences of the loop (records with X'071' completion codes) or records of previous errors in the currently looping process that could possibly contribute to the current loop. See the "Locking" chapter and the discussion on LOGREC in the "Use of Recovery Work Areas" chapter in Section 2.

***Objective 3** - What is the extent of the loop and why is the system looping?*

Using the current PSW and the global data areas in combination with the general purpose registers, you should be able to determine the extent of the loop. One register often contains the key to a loop-causing value. Try to isolate that one register. It may be necessary to inspect the actual object/source to determine the basic logic in case there is an encoded loop that is supposed to end when a certain value is reached. If that value cannot be reached for some reason, the loop will not end.

Isolating the cause of the loop is important in loop analysis. Once the cause is isolated, you can proceed the same as with a system-detected error such as a program check.

***Objective 4** - Determine the cause of the error - how is the value that is causing the loop developed?*

To determine how the bad value was developed, it is necessary to back through the logic leading to the loop. Be aware of bad control blocks. Look at the bad value itself and the areas from which it was developed. Try to determine if the value is the result of a storage overlay or if it was calculated from bad logic. See the "Pattern Recognition" topic in the "Miscellaneous Debugging Hints" chapter of Section 2 to help make this determination.

In addition to bad control blocks and data fields, consider the loop control mechanism used for encoded loops. Often a common cause of problems is that the BCT instruction is used and the loop control register contains a negative value. Scanning the active registers at the time of the dump often aids in discovering this type of problem.

Figuring out how the erroneous field could possibly contain the value it does is the most challenging part of the process. Again, the contents of the field often provide the clue to determining the error-causing process.

Also, consider how serialization is accomplished for the field in question. Is it possible for several processors to be updating the field simultaneously? The MVS trace helps you recreate recent processes, but you also must understand the modes and structure of the code that updates the field. (Your work in Objective 2 should be helpful.)

It is possible that the code setting up the field was physically interrupted and, because it was non-reentrant or the logic was faulty, another process updated the field or control fields and subsequently caused the first process to encounter unexpected data.

TP Problems

A common problem in teleprocessing (TP) environments is incorrect data, which may affect one terminal or an entire component. The symptoms include no data, wrong data, or too much data, but the general problem symptom is that something is wrong with one or more messages. The problem is usually not tied directly to a component or access method, as a program check would be; often an error message is from a component not directly causing the problem.

Typical symptoms are:

- An error response from an application that suggests incorrect data was entered from a terminal, when in fact the data was correct
- A "hung" terminal - the system will not respond
- Wait states, in which message traffic gradually dies off
- Incorrect characters in a message (the data may be going into or out of the system)

This chapter discusses TP problem analysis, in the following topics:

- Message Flow Through the System
- Types of Traces

Message Flow Through the System

Data exchanged between programs in the system and terminals follows a route through several components. The first step in solving "typical" problems is to determine where along that route something is happening incorrectly.

By far the most valuable tools for doing this are the traces in the various components. To use the traces effectively it is necessary to understand how messages flow through the system. For example, consider a message from an ACF/TCAM application to an ACF/TCAM terminal. The path might go from the application program buffer to an ACF/TCAM queue data set, to an ACF/TCAM buffer while ACF/TCAM processes it, over the channel into the 3705, then into an NCP buffer, and finally over a communications line to a terminal. Traces allows the message to be checkpointed at certain spots along the path; therefore, understanding the path is vital to knowing what traces to use and what you should see for a message that flows correctly.

To use traces effectively you must also understand how components refer to terminals or lines and how they communicate with each other regarding these terminals and lines. Terminals or lines are identified in traces by a line control block (LCB), a logical name, a network address, a polling/addressing sequence, or a subchannel address. Not only must these relationships be known in order to use multiple traces, but certain correspondences must be correct in order for data to move through the network.

When using traces, the general approach to a problem of incorrect data is to track the data flow from a point where everything was all right to a point where the messages stopped or were incorrect. Messages that are flowing correctly can be used to establish time relationships between different traces. Then each message can be followed along its route past each checkpoint, with the goal of isolating a gap between two checkpoints where the message stopped or became bad. The next step is to focus on this narrower area to learn what is wrong.

If a message stops, what is wrong or what is missing? How does the flow up to that point compare with a normal flow? You must understand what resources and what processes are required for a message to move from where it appeared last to where it should have appeared next. What buffers and/or control blocks would have been used? Were they available? A single terminal or all terminals may encounter a "wait state," and it is necessary to dig into the component to determine what processing has taken place and what condition or resource is preventing further processing. The *ACF/TCAM Diagnosis Guide* should be referenced for problem isolation in ACF/TCAM.

If a part of the data moving through the system becomes bad, the traces should isolate a component or an interface over which it was transformed. Comparison with normal message flow will indicate whether any change at all should have taken place. If no change should have occurred, an overlay ("clobber") or incorrect pointers to data buffers may be the problem. The exact amount and positioning of bad data should be determined, for it might provide an obvious correlation with other known variables such as a buffer length. If some transformation normally occurs to a message, the controlling process under which it is performed must be examined. What could cause an incorrect transformation of data? Examples are translate/edit tables or mappings from one resource name into another name, such as mapping the logical network name into the network address.

Types of Traces

The following is a summary of TP-related traces.

GTF Traces

The following GTF trace options can be used to provide traces of TP-related activities. (Refer to *SPL: Service Aids* for additional information.) The GTF traces include:

- SIO and IO trace options - record start I/O and I/O interruptions for either an emulation mode subchannel or a 3705 controller.
- RNIO trace option - records the header and first few bytes of data for each PIU coming into or going out of ACF/VTAM.

You can correlate the GTF traces with the ACF/VTAM, ACF/TCAM, and NCP and EP traces to compare relationships of the system and TP activity.

ACF/VTAM Traces

ACF/VTAM provides several kinds of traces to record the flow of path information units (PIUs) between nodes in the ACF/VTAM network. (Refer to *ACF/VTAM Diagnosis Guide* for additional information.) The ACF/VTAM traces include:

- Buffer contents trace - records the contents of inbound and outbound message buffers.
- I/O trace - provides a chronological history of all I/O activity between ACF/VTAM and a particular network node.
- Line trace - records the status of a line each time that data is sent or received along the line.
- Generalized PIU trace - records the PIU trace data obtained by the NCP.
- Transmission group trace - records the sequence of PIUs being sent through a transmission group.
- SMS (buffer use) trace - records information about the use of buffers.

ACF/TCAM Traces

ACF/TCAM provides several service aids for diagnosing ACF/TCAM problems. (Refer to *ACF/TCAM Diagnosis Guide* for additional information.) The ACF/TCAM traces include:

- Channel I/O interrupt trace - provides a sequential record of the I/O interrupts occurring for specified non-NCP line addresses or for the channel addresses of 3704/05 controllers.
- NCP line trace - provides a sequential record of the level two (I/O) interrupts occurring for specified NCP lines.
- PIU trace - provides a sequential record of all PIUs sent to or received from NCP network resources.
- Buffer trace - provides a record of ACF/TCAM buffer contents before and after message handler (MH) processing.

NCP and EP Traces

The network control program (NCP) and emulation program (EP) provide several diagnostic aids to diagnose difficulties in network operations. (Refer to *IBM 3704 and 3705 Control Program Generation and Utilities Guide and Reference* for additional information.) The NCP and EP traces include:

- Address trace facility for network control mode - records the contents of selected areas of communications controller's storage and registers for successive interrupts in the NCP.
- Line trace facility for network control mode - records the operating parameters of a line each time that a level two interrupt occurs for the line.
- Line trace facility for emulation mode - records the operating parameters of a line each time that a level two interrupt occurs for the line.

Performance Degradation

This chapter describes how to investigate performance degradation problems. It is not intended to serve as a tuning guide or as a reference for general performance analysis (which should be performed through SMF, GTF, etc.)

The following points should be considered when a problem is suspected in the operating system itself or in the manner in which applications use the operating system.

Operator Commands

When a bottleneck or system failure, hardware or software, is degrading throughput, the following operator commands can help identify the source of degradation and, possibly eliminate it.

D A,L Displays current system status. A job step with a name of STARTING indicates initiation has not successfully completed. Also, if a job step is marked with an 'S,' it is considered swapped out. Other jobs may be queuing behind these jobs in an allocation/deallocation path.

D R,L Displays any outstanding requests. Operator action is required (for example, to mount a volume). Other jobs may need to wait until action has been taken.

D M Displays configuration information. The loss of a hardware component (for example, a channel) may have been noted on a hard copy console and missed by the operator.

D SLIP Displays the name and status of current SLIP traps. Display those traps that are enabled (D SLIP=xxxx (where xxxx is the trap id) to see if any are PER traps (PER-IF, PER-SA, or PER-SB). If an enabled PER trap with an action other than IGNORE exists, check with the system programmer to determine if it should remain enabled or if it can be disabled.

If a resource queue "snooper" program exists, it should be started and output examined to find any ENQ bottlenecks. If no such program is available, take a dump of the global resource serialization address space, the nucleus, and request SQA. The PRDMP service aid (with the QCBTRACE, Q, or GRSTRACE option) can then be used to print the dump so the resource queue can be examined.

Use the job entry subsystem display commands to find the status of jobs, queues, printer setups, requirements of SYSOUT data sets, etc. to find reasons why JES2 is not able to schedule work. Some JES2 commands that may be useful are shown in Figure 4-1.

$D	J1-9999 S1-9999 T1-9999	Status of jobs, started tasks, or time-sharing users. If a range of jobs has been held they may be released using $AJ.
$AJ		Release jobs.
$DF		Status of output forms queue.
$DU,PRTS		Status of printer setup characteristics.
$TPRTN		Change setup to needs of queue output.
$LJ1-9999,H		List held SYSOUT data sets.
$OJ1		Release held SYSOUT data sets.
$DQ		Display queue.
$AQ		Release queue.
$AA		Release all jobs held by a $H command.

Figure 4-1. JES2 Commands for Status Information

If the use of previous commands does not make it obvious why JES2 is not scheduling work, take a dump of the JES2 address space. Print the SYS1.DUMPxx data set to help determine the problem.

Find the number of the IPS member that should be active and issue T IPS=na to ensure that it is active. Print the IPS member in SYS1.PARMLIB and analyze the IPS for an explanation of degraded service. Then, enter the W command to print the system log to obtain the history of system execution.

Figure 4-2 shows important hardware components used by the system that should be understood when a degradation problem is suspected.

Dump Analysis Areas

The following areas in a storage dump may provide a starting point for further analysis. Problems in these areas may indicate a bug or some unexpected use of system services.

1. ENQ/DEQ - A check of ENQ/DEQ's processing queues may indicate contention problems. A queue of many QELs off a particular QCB should be explained. An indication of a possible problem is a mixture of shared and exclusive requests intertwined for one resource. The state (running/waiting/swapped-out/etc) of the holder of the resource should be determined.

   The QCBs defining all enqueued resources are chained from either the local or global hash table. To locate the hash tables:

   - CVTGVT (CVT+X'1B0') points to the global resource serialization vector table (GVT), which resides in the nucleus.

   - GVTGVTX(GVT+X'10') points to the GVT extension (GVTX), which resides in the GRS address space.

- GVTXGQHT (GVTX+X'A4') points to the global queue hash table in the GRS address space.

- GVTXLQHT (GVTX+X'A8') points to the local queue hash table in the GRS address space.

Additionally, if the ENQ/DEQ request was initiated from this system, you can locate the resources and the requesters of resources (QELs) for an address space by following the queue of QELs chained from the fields ASCBGQEL (ASCB+X'110') or ASCBLQEL (ASCB+X'114'). If the request was initiated from another GRS system, the QELs are chained from the SYSID/ASID hash table, which is pointed to by GVTXSAHT (GVTX+X'AC').

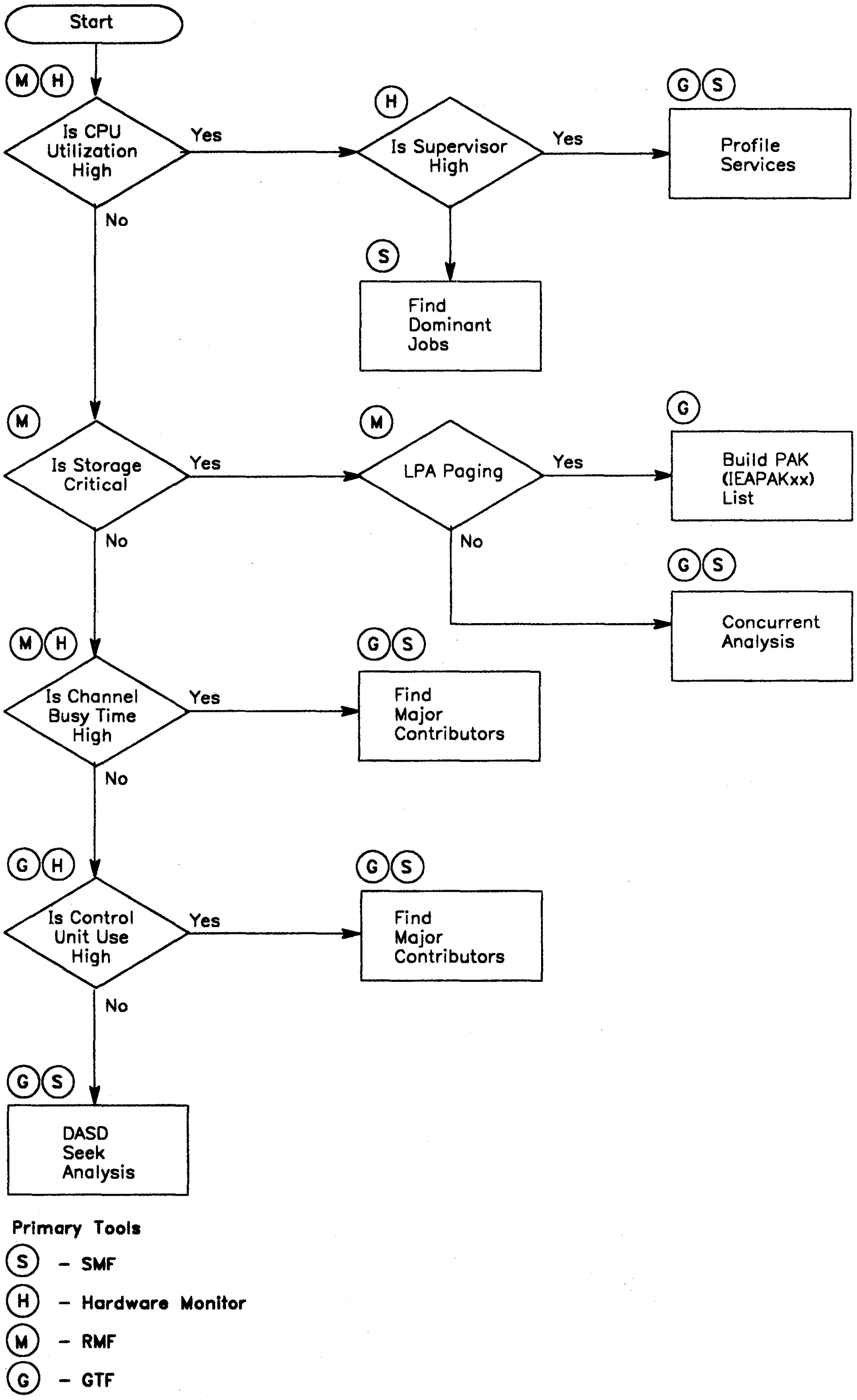


Figure 4-2. System Use of Hardware Components

2. IOS storage manager queues should be inspected. The anchors for the various pools (small, medium, and large block pools) are located at the end of IECVSMGR at external symbol IEVVSHDR, which should show up in a NUCMAP. Generally there should be one 2K page of small blocks (used for IOQEs) and one 4K page of medium blocks (used for RQEs). Examine the large blocks in detail. If the system was quiesced, there should be two 4K pages of large blocks and all blocks should be on the free queue. Many heavily-loaded systems require 8-10 pages of large blocks. If the actual number is much higher than this, determine the ASID that each in-use block is assigned to (the two bytes at block address-8 contains the ASID address). System address spaces can have many blocks, but any user address space with a large number of blocks should be explained.

   Common problems are: I/O loop, I/O errors, and storage not being freed at I/O termination time. These page frames occupy real storage, which depletes the pool of available real storage and possibly causes excessive paging.

3. Check page frame table entries (PFTEs) for large fix counts. The CVT+X'164' contains the address of the PVT; PVT+X'C' contains the address of the *apparent* PFTE origin - you must index several hundred bytes (X'10' times the number of pages in the nucleus). Large fix counts may indicate a page fix macro loop, or page fix without page free. Frames allocated to a private area space may indicate a user error. Try to analyze the contents of the page for a clue as to who is page fixing without page freeing.

4. Check PFTEs for bad frames caused by hardware storage errors that rendered these frames unusable (in PFTE+X'C', the X'04' flag should be set if this is the case). Contact hardware personnel to determine if a machine malfunction has occurred.

5. CDE (contents directory entry). These blocks represent modules loaded into virtual storage; CDEs reside in SQA and the queue is anchored at CVTQLPAQ (CVT+X'BC'). The loaded module's name and starting address reside in the CDE. Those with starting addresses less than the value in CVTLPDIA (CVT+X'168') were members of either an IEAFIXnn list or an IEALPAnn list. For members of these lists, CDEs are built by NIP and they occupy real, fixed storage even when the module is not in use. If fixed storage or fragmentation is a problem, moving these modules to LPA can provide a partial solution.

6. The BLDL table, pointed to by the nucleus external symbols IEARESBL and/or IEARESBS, should be checked. The address(es) should be less than the value at CVTNUCB, the upper nucleus boundary. If not, try changing the BLDL=nn system initialization parameter to BLDF=nn. This will cause the BLDL list to occupy real storage at all times. If the number of entries is less than 93, one frame is used.

7. In a quiesced system, the number of paging requests received should equal the number of paging requests completed by ASM. The fields ASMIORQR (ASMVT+X'28') and ASMIORQC (ASMVT+X'2C') in the ASMVT represent the number of requests received and completed, respectively. The difference between the two counts represents requests not completed. A large number of uncompleted requests can indicate ASM is either not processing at

all or is taking considerable time for each operation. Examine the PAT (page allocation table) to determine whether the page data sets are almost full. Also examine ASMERRS (ASMVT + X'7C'), PAREFLG1 (PARTE + X'9'), and the IOSB for paging requests (IOSCOD = X'D') to determine if I/O errors have occurred and the data sets are no longer in use.

8. CSA use should be examined. If SQA is depleted, requests are filled from CSA. This can be determined by inspecting the SQA DQE (descriptor queue element):

   - the CVT + X'230' points to the GDA (global data area)
   - the GDA + X'18' points to the SQA SPQE (subpool queue element)
   - the SPQE + X'4' points to the SQA DQE.

   The DQEs are chained together. If more than one DQE exists for the SQA, it has expanded into the CSA. This causes the frame to be fixed. Also, often CSA users page fix. In this case fragmentation, if present, could cause performance degradation.

9. A possible real frame shortage can be indicated by inordinately large counts in the PVT fields: PVTRSQA (a count of the number of times the SQA reserved frame was allocated), and PVTDFRS (a count of the number of times real frame allocation was deferred because of a lack of frame availability). These counts by themselves mean little, but can be of some use when analyzing an overall problem.

Incorrect Output

The problem of missing, unexpected, or erroneous output is one of the most difficult. This incorrect output might take the form of a message on the console log or in SYSOUT, or an incorrect total in a report. There is usually very little documentation that assists the debugger in analyzing incorrect output.

Initial Analysis Steps

To resolve the problem of missing or incorrect output the analyst must have a complete understanding of the job environment. There is no fast, clear cut approach to these errors. This section only tries to assist your thought processes as you begin to work on a problem of this type.

There are four basic categories of incorrect output: missing, unexpected, erroneous, or a combination of these. The steps in resolving the problem must take the category into account.

Initially, consider the following steps:

1. Gather all possible documentation. You will probably need additional information as you begin to understand the problem in more detail.

2. Consider all recent hardware and software changes to the system and to the application(s) if relevant. A change to an application that updates a data base affects all other data base users.

3. Remember that output requires input. Consider the possibility of bad input.

4. Consider whether the problem is associated with some new function or application. Most incorrect output errors occur in the installation and test phase.

Isolating the Component

Next, attempt to locate the component causing the error. Do this by thinking through the flow. Listed below are some questions that might assist you.

- Is the problem related to a user function or application? If yes, have there been recent changes or is testing still in progress?

- Is the job control language correct? Have there been recent changes to the JCL?

- Have any user exits been added or modified?

- Have any user supervisor calls (SVCs) been added or modified?

- Are there operator interactions that could affect the input/output?

- From which access method or function is the output expected? Some examples are: JES, VSAM, BTAM, TCAM, and WTO.

- Was RJE involved in the input and/or output?

- Was there any cross-address-space communication involved in the data movement? In MVS, most telecommunication requires data passing between address spaces.

- Is there any evidence of I/O error activity? Refer to the console log and LOGREC data.

- Do you have a storage dump, or should you obtain one? See the chapter on "Additional Data Gathering" in Section 2.

- Would a trace be helpful in understanding the flow? Consider tracing the activity with GTF.

Many of the above questions have to be answered in order to get a better understanding of the problem area. In many cases, the problem has to be recreated with various traces or traps. These questions help to determine what data is needed to solve the problem.

Analyzing System Functions

To solve an incorrect output problem, you must understand the mode of operation and the processes required to accomplish the function in question.

The first question must be the following: where does the output originate? Then you must be able to verify that the activity did occur. There must be some means for understanding the path the data should take from the origin to the final location (device).

Consider the following example:

1. A TSO user invokes his program which should write a message to the terminal and then wait.

2. The program waits after the I/O but no message appears.

3. What are the system functions involved?

   a. A language translator and the linkage editor that created the load module.

   b. OPEN code necessary to complete the link between the device and the user PUT macro.

   c. TSO TIOC flow. The user issues PUT which branches to the TIOC module IGG019T4. This module issues TPUT. What is the TPUT path through TIOC?

   d. TSO TIOC interfaces with TCAM. What is the data path through TCAM?

e. TCAM interfaces with the I/O supervisor. Can evidence be found of the SIO? What types of trace would be helpful?

In this example it may be necessary to take a series of dumps to resolve where the message was lost. But first be certain that the correct message is in the correct buffer at the time of the user PUT macro.

It could be necessary to apply this type of thinking all the way down to the CSECT level.

Summary

In analyzing incorrect output, there are two key points. The first is that a better understanding of the system flow is probably required for this type of problem than for any other. The second point is that it is very important to be able to obtain the correct documentation at the correct time.

Note: The chapter on TP (teleprocessing) problem analysis earlier in this section provides some specific steps for analyzing incorrect output in the TP environment. Many of the techniques in that chapter can be applied to incorrect output analysis.

Section 5. Component Analysis

This section describes the operating characteristics and recovery procedures of selected system components and facilities, and provides debugging techniques for determining the cause of an error that has been isolated to a component or facility.

The section contains the following chapters:

- Supervisor Control
- IOS
- Program Manager
- Virtual Fetch
- VSM
- RSM
- ASM
- SRM
- VTAM
- VSAM
- Catalog Management
- Allocation/Unallocation
- JES2
- SSI
- ENF
- RTM
- Communications Task
- RMS
- Service Processor Call SVC and MSSFCALL DIAGNOSE Instruction
- Service Processor Call SVC and SERVICE CALL Instruction
- Cross Memory Services
- Global Resource Serialization

Supervisor Control

This chapter includes diagnostic techniques for the following supervisor control functions:

- Dispatcher - which controls the initiating of work within the system.
- SRB/SSRB pool manager - which obtains and frees SRBs and SSRBs.
- Stop/reset services - which suspend and then reset units of work.
- SUSPEND/RESUME/TCTL services - which suspend and resume tasks, and transfer control to tasks.

Dispatcher

For effective problem analysis, it is important to understand how work is processed by the MVS system. The MVS dispatcher plays a large role in processing work by controlling the initiating of all work within the system. An understanding of the dispatcher's processing and control block structure is imperative for the debugger.

This chapter describes the following items about the MVS dispatcher:

- Important dispatcher entry points
- Dispatchable units and sequence of dispatching
- Dispatchability tests
- Dispatcher recovery considerations
- Dispatcher error conditions

Important Dispatcher Entry Points

The dispatcher's main entry points are the following:

IEA0DS - Entered disabled, key 0, supervisor state. If there is status to save (PSATOLD"0), the task being preempted can hold the LOCAL lock, a CML lock, or one or more of the CMS locks. The dispatcher saves the cross memory environment (control registers 3 and 4) in the XSB chained off of the TCB.

Callers of this entry point include:

- CALLDISP (type 6 SVC) when cross memory status has not been saved.
- Program FLIH when the FLIH has scheduled an SRB to perform PIE/PICA processing.

IEA0DS1 - Entered disabled, key 0, supervisor state. If there is status to save (PSATOLD"0), the task being preempted can hold the LOCAL lock, a CML lock, or one or more of the CMS locks. The dispatcher assumes that the cross memory environment (control registers 3 and 4) has already been saved.

Callers of this entry point include:

- Exit Prologue (IEAVEEXP), when control has not returned to the issuer of an SVC.
- CALLDISP (type 6 SVC) enters the dispatcher at this entry point.
- SVC FLIH (IEAVESVC) when the local lock is required but not available.
- RTM (recovery termination manager).

IEAVDSPC - Entered disabled, key 0, supervisor state if there is status to save (PSATOLD"0). The task to be preempted may hold the LOCAL or CML lock, or the LOCAL or CML lock and at least one cross memory services lock.

This entry point is called by the following:

- Program check FLIH (IEAVEPC) when a TCB or SRB is suspended for a page fault (I/O required or no frames available), or for a page fix.
- Lock manager (IEAVELK) when suspending a task that unconditionally requested a local lock that was unavailable.

The only difference between IEAODS1 and IEAVDSPC is that IEAODS1 also does a store processor timer (STPT) instruction.

IEAPDS7 - Entered disabled, key 0, supervisor state, no locks held.

This entry point is called by the following:

- I/O FLIH
- External FLIH

IEAPDS7A - Entered disabled, key 0, supervisor state, interrupted task holds local lock or local and at least one cross memory services lock. This entry point is called by the following:

- I/O FLIH
- External FLIH

IEAPDS7B - Entered disabled, key 0, supervisor state, no locks held, interrupt occurred in wait state.

This entry point is called by the following:

- I/O FLIH
- External FLIH

IEAPDS7C - Entered disabled, key 0, supervisor state, interrupted task holds CML lock or CML lock and at least one cross memory services lock.

This entry point is called by the following:

- I/O FLIH
- External FLIH

IEAVDSTC - Entered disabled, key 0, supervisor state, no locks held.

This entry point is called by:

- Transfer Control (IEAVETCL), when requested by an SRB to give control directly to a specific TCB.

IEAPDSRT - Entered enabled or disabled, any key, supervisor state, no locks held.

This entry point is the termination return address for all SRBs. If a lock is held, the dispatcher issues an 066 abend with reason code X '04'.

DSJSTCSR - Job step timing subroutine. Calculates and accumulates job step timing.

This entry point is called by the following:

- Timer SLIH. The timer SLIH calls this subroutine before it gives control to SRM.
- Stimer service routine to accumulate job step time prior to enqueuing or dequeuing a task-type TQE.
- EXIT (IEAVEOR) calls this routine when the last RB of the task is exiting.
- SVC FLIH (IEAVESVC) when a type 6 SVC exits and gives control to an SRB.

Dispatchable Units and Sequence of Dispatching

This section describes the unique dispatchable units of work and the queues where they are located. The dispatchable units are described below and are listed according to the priority in which the dispatching queue is searched. In general, this is also the priority with which units of work are dispatched. (However, there are exceptions to the dispatching priority: for example, a ready TCB might be dispatched before a local SRB if the SRB has affinity to another processor.)

1. Special Exit

   A special exit is made known to the dispatcher by a unique flag setting in the LCCADSF1 (LCCA+X'21C') field. The LCCADSF1 bits and the exits they indicate are:

Bit	Exit
LCCAACR	ACR
LCCAVCPU	Vary CPU
LCCATIMR	Timer Recovery

   The dispatcher enters these exits via a branch.

2. Global SRBs

   SVTGSMQ is the header for the global SRB staging queue. If it is not zero, it points to the global SRB (see Figure 5-1). Requestors use the SCHEDULE macro to compare and swap global SRBs onto either this queue or onto SVTGSPL. If both queues are empty, the SRB is placed directly onto the GSPL; otherwise it is placed on the GSMQ. The dispatcher moves the SRBs from the SVTGSMQ with the compare and swap (CS) instruction, placing them on the SVTGSPL from which they will be dispatched.

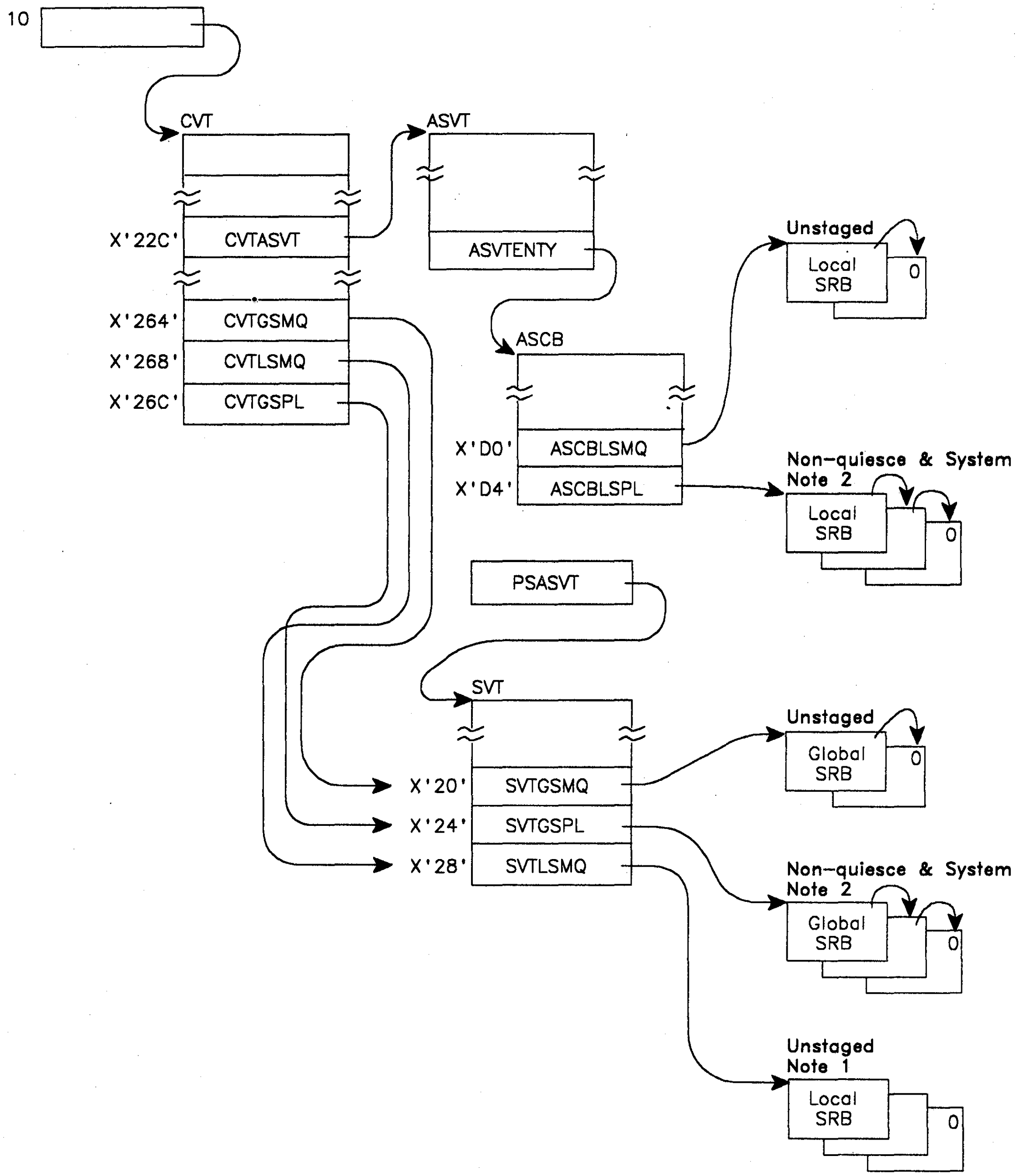
   SVTGSPL is the global SRB dispatching queue. The dispatcher removes the SRB from the GSPL queue, updates the PSAAOLD with the SRBASCB address, uses the CMSET service to establish primary and secondary addressability to the address space, and dispatches the SRB. PSAANEW is not updated.

3. Local SRBs

   ASCBLSMQ is the header for the local SRB staging queue. If it is not zero, it points to a local SRB (see Figure 5-1). Requestors use the SCHEDULE macro to compare and swap local SRBs onto this queue, or onto ASCBLSPL. If both queues are empty, the SRB is placed directly on the LSPL; otherwise it is placed on the LSMQ. The dispatcher moves the SRBs from the ASCBLSMQ with the compare and swap (CS) instruction, placing them on the ASCBLSPL from which they will be dispatched.

   ASCBLSPL is the local SRB dispatching queue. The dispatcher removes the SRB from the LSPL, updates PSAAOLD, uses the CMSET service to establish primary and secondary addressability to this address space, and dispatches the SRB.

   For compatibility, SVTLSMQ is included to support users of the SCHEDULE macro who have not recompiled (see Figure 5-2). Previously, the SCHEDULE macro placed all local SRBs on one common queue rather than on the appropriate address space queue. The dispatcher tests this queue if it cannot find any special exits or global SRBs to dispatch. If the queue is not empty, the dispatcher calls CSECT IEAVESC5 in order to move the SRBs to the appropriate ASCBLSMQ.



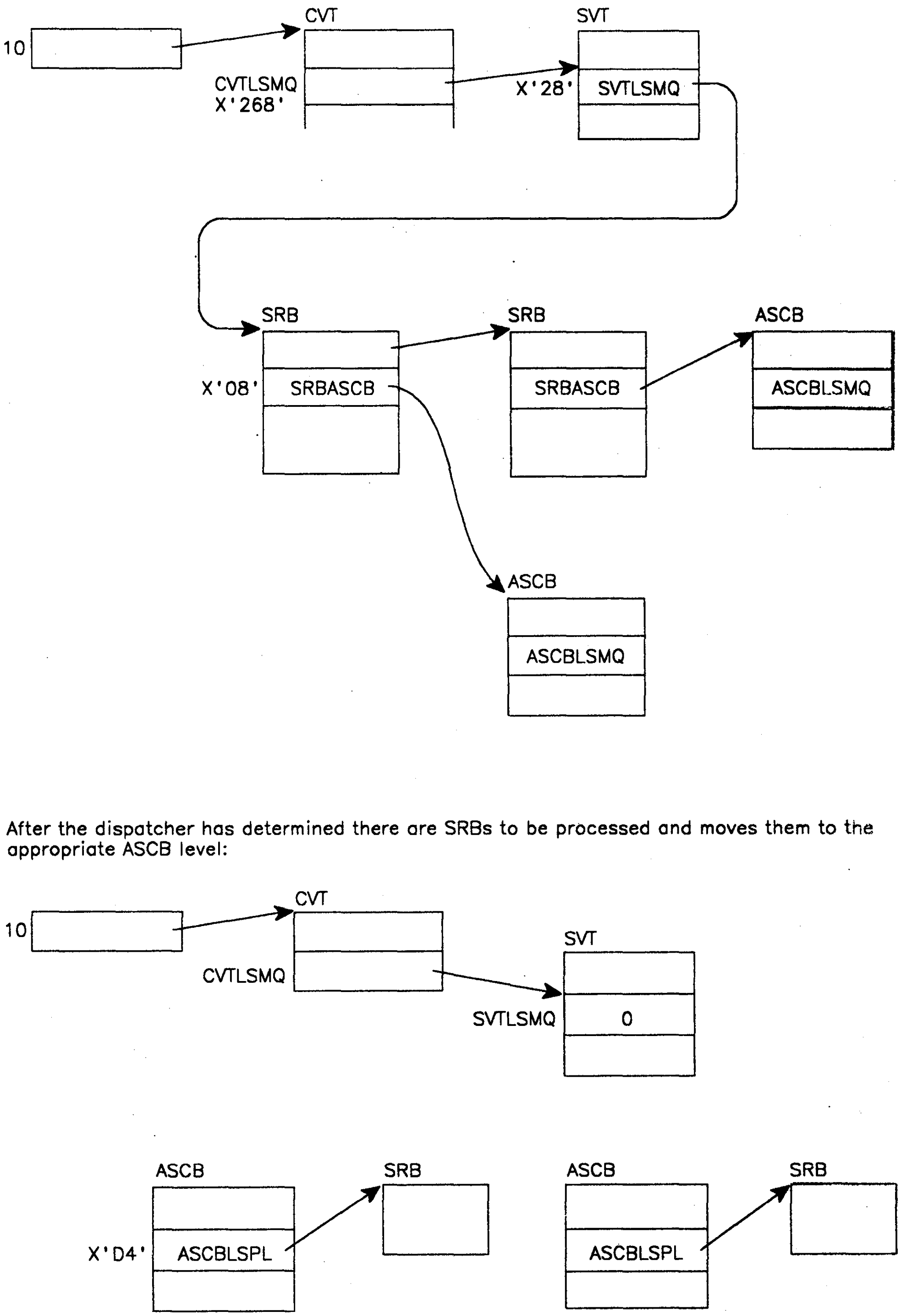
Notes:

1. This queue is maintained for compatability. The SCHEDULE macro places local SRBs on the address space LSMQ.

2. All queues are now single-threaded and one SPL contains system and non-quiescable SRBs.

Figure 5-1. SRB Queue Structure and Control Block Relationships

After a user request to schedule local SRBs:



After the dispatcher has determined there are SRBs to be processed and moves them to the appropriate ASCB level:

Figure 5-2. Local SRB Queue Structure and Control Block Relationships

4. Address Space Dispatcher

   This is not actually a unique dispatchable unit of work, but rather an anchor for the real dispatchable units of work (that is, local SRBs or TCBs). The address space dispatcher is entered to select the next address space in which work will be dispatched. If an address space is dispatchable, the priority of dispatching within the address space is the following:

   a. Local SRBs
   b. Local supervisor (locally locked, interrupted work)
   c. TCBs

   If the dispatcher finds any SRBs on the ASCBLSPL, the top SRB is dequeued and dispatched. If there are no SRBs on the ASCBLSPL queue, the ASCBLSMQ is checked. If it contains any SRBs, the dispatcher stages the entire LSMQ, removing the top SRB and dispatching it. If there are no SRBs, the LOCAL lock is tested for interrupt id, X'FFFFFFFF'. If the interrupt id is in the LOCAL lock, the id is changed to the current CPU ID via compare and swap, and the status (FRRs, GPRs, FRR stack, CPU timer value, PSATOLD, PSATNEW and resume PSW) is restored from the IHSA (interrupt handler save area). The ASCBASXB points to the ASXB: ASXBIHSA (ASXB+X'20') in turn points to the IHSA. Status is saved in the IHSA when a locally-locked task is interrupted and control is switched away from it because there is higher priority work to handle.

   If ASCBLOCK contains a suspended ID (X'7FFFFFFF'), the dispatcher checks the ASCBRCMS field to determine if the suspension is due to a cross memory services lock request. If it is, the dispatching priority of the holder of the cross memory services lock is checked. If the priority is lower than the current ASCB dispatching priority, the dispatcher switches to the cross memory services holder's address space in order to dispatch the holder of the cross memory services lock.

   If ASCBLOCK contains a ready-to-run ID (X'4FFFFFFF'), the dispatcher checks the ASCBLOCI field to determine if the lock is held as a CML lock. If the lock is held as a CML lock and the address space in which the holder of the CML lock resides is lower in priority than the current address space, the dispatcher switches to the lock holder's address space in order to dispatch the lock holder. This is done to prevent a low priority address space from creating a bottleneck in a higher priority address space. If the CML lock holder's address space is a higher priority, then the dispatcher assumes the lock holder was not ready to run because it would already have been dispatched.

   If the LOCAL lock is available (ASCBLOCK=0):

   - and the ASCBS3S bit (ASCB + X'138') indicates that there is work for the stage 3 exit effector to process, the dispatcher will obtain the LOCAL lock and go to the stage 3 exit effector.

   - the dispatcher next determines if the number of ready TCBs not requiring the LOCAL lock (ASCBTCBS), plus the number of TCBs requiring the LOCAL lock (ASCBTCBL), exceeds the number of processors active in the address space. If so, there is work to dispatch and the dispatcher searches for the first dispatchable TCB that is not active on another processor, starting at ASCBTNEW. If the first ready TCB requires the LOCAL lock, the dispatcher will obtain it for that TCB.

If the LOCAL lock is not available, (or the dispatcher fails to obtain it), and the number of ready TCBs not requiring the LOCAL lock (ASCBTCBS) exceeds the number of processors active in the address space, the dispatcher will select the first dispatchable TCB that does not require the LOCAL lock and is not active on another processor. Every search for a dispatchable TCB begins with the ready TCB pointer (ASCBTNEW).

5. Wait Address Space

   The wait address space is dispatched when the dispatcher reaches the bottom of the ASCB ready queue and can find no ready work after a recursive search of the SRB queues and the ready queue.

   When the wait address space is dispatched, the dispatcher uses the CMSET service routine to establish primary and secondary addressability to the master scheduler address space (ASID = 1). PSAANEW is set to the ASCB address of the wait address space and PSAAOLD is set to the ASCB address of the master scheduler's address space.

   Figure 5-3 provides an overview of the processing sequence through the MVS dispatcher.

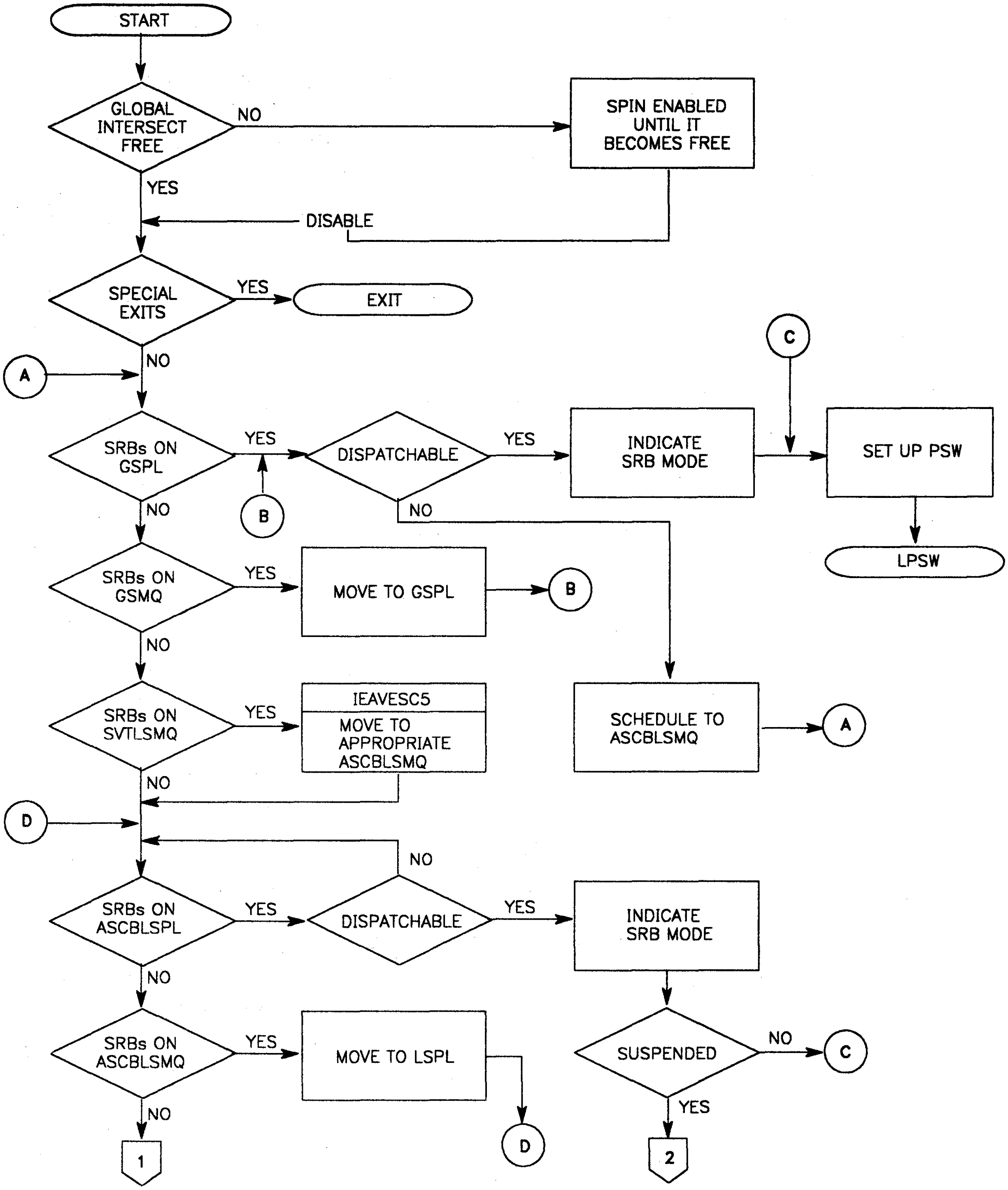


Figure 5-3 (Part 1 of 3). Dispatcher Processing Overview

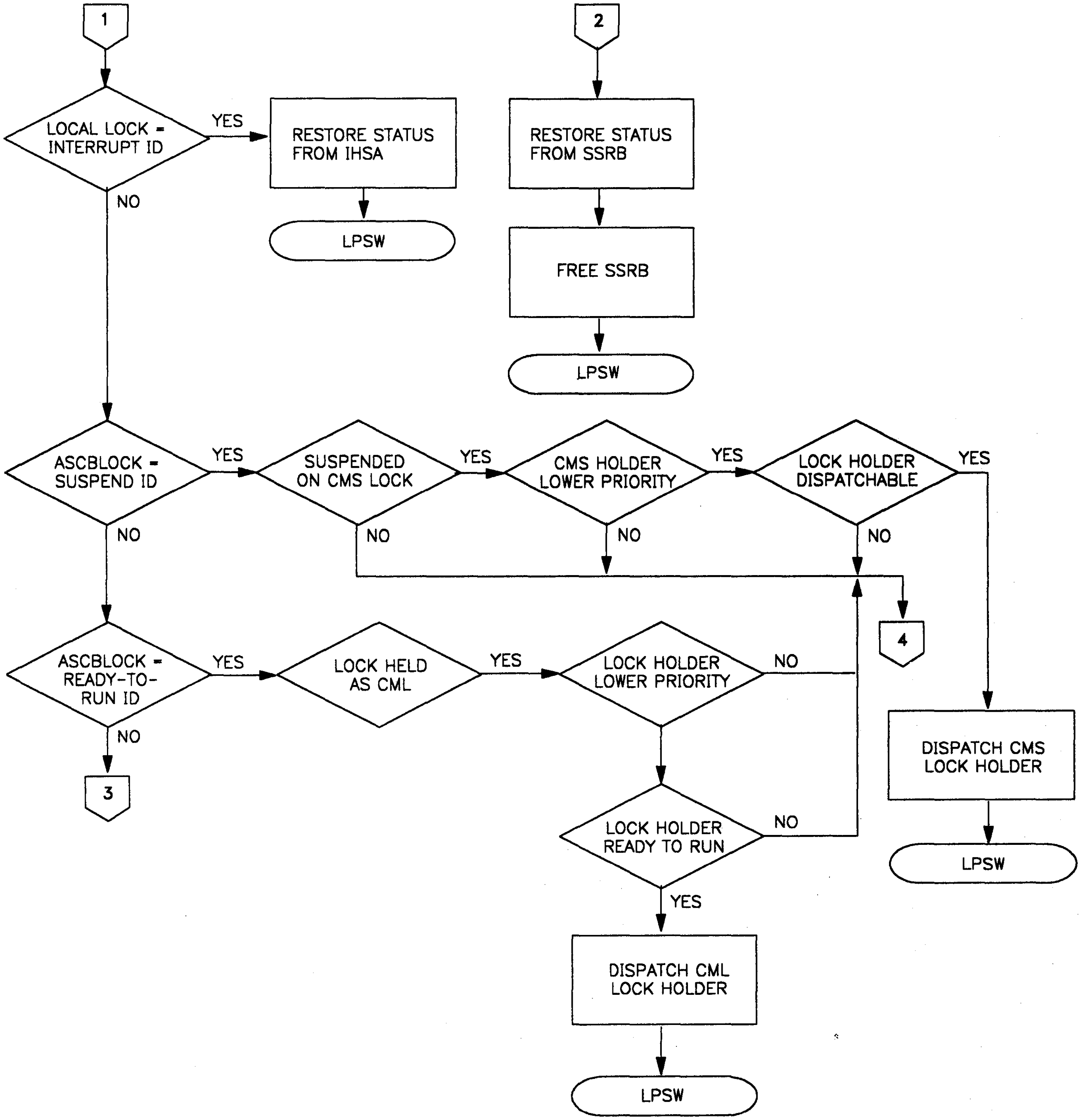


Figure 5-3 (Part 2 of 3). Dispatcher Processing Overview

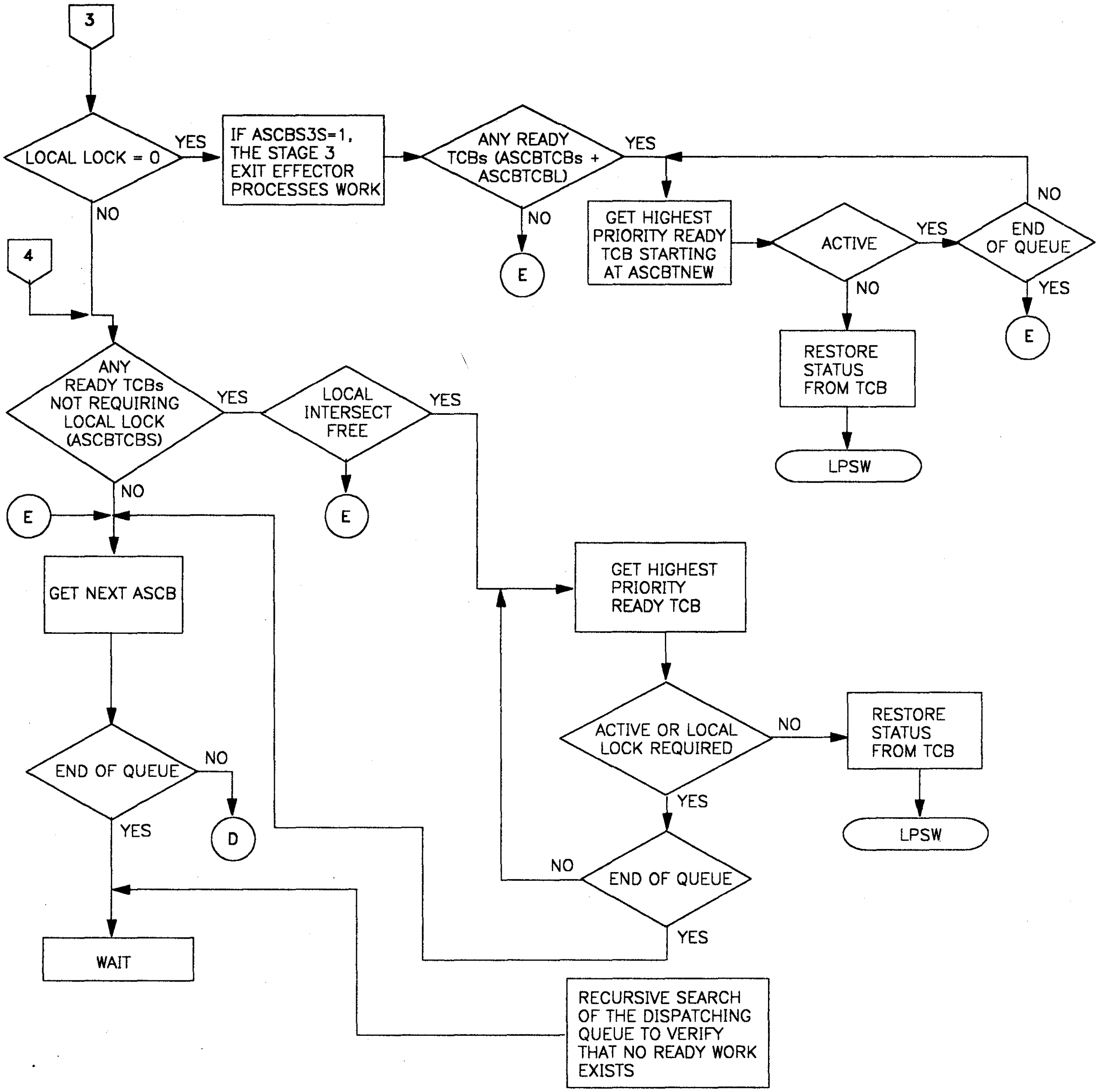

3
LOCAL LOCK = 0
YES
NO
IF ASCBS3S=1, THE STAGE 3 EXIT EFFECTOR PROCESSES WORK
ANY READY TCBs (ASCBTCBs + ASCBTCBL)
YES
NO
E
GET HIGHEST PRIORITY READY TCB STARTING AT ASCBTNEW
ACTIVE
YES
NO
END OF QUEUE
NO
YES
E
RESTORE STATUS FROM TCB
LPSW
4
ANY READY TCBs NOT REQUIRING LOCAL LOCK (ASCBTCBS)
YES
NO
LOCAL INTERSECT FREE
YES
E
E
GET NEXT ASCB
GET HIGHEST PRIORITY READY TCB
END OF QUEUE
NO
D
YES
ACTIVE OR LOCAL LOCK REQUIRED
NO
YES
RESTORE STATUS FROM TCB
LPSW
NO
END OF QUEUE
YES
WAIT
RECURSIVE SEARCH OF THE DISPATCHING QUEUE TO VERIFY THAT NO READY WORK EXISTS

Figure 5-3 (Part 3 of 3). Dispatcher Processing Overview

Dispatchability Tests

The dispatcher conducts the following dispatchability checks:

SRB Tests

Test*	Condition
	Dispatchability flags
1. ASCBDSP1//ASCBSSND	System non-dispatchable and this address space non exempt
2. ASCBDSP1//ASCBFAIL	Address space in failure mode and in the process of being terminated
3. ASCBDSP1//ASCBSSS	System-level SRBs stopped (does not apply to nonquiesceable SRBs)
4. ASCBDSP1//ASCBSNQS	All SRBs stopped
5. SRBCPAFF	Does SRB have affinity to this processor? (PCCACAFM defines the current processor)
6. SRBFLGS//SRBLLREQ	LOCAL lock required at dispatch time
7. SRBFLGS//SSRLLHLD	This SRB holds the LOCAL lock. The ASCBLOCK will be changed from X'4FFFFFFF' to the CPU ID when the SRB is dispatched.
8. SRBASCB	Does SRB point to valid ASCB? (Global SRBs only)

*Format of test description is "field//bit within field."

Address Space and Task Tests

The following address space test criteria must be met before the task dispatcher gets control.

Address Space Tests	Condition
1. ASCBDSP1//ASCBSSND	System non-dispatchable and this address space not exempt.
2. ASCBDSP1//ASCBFAIL	The ASCB is in failure mode and in the process of being terminated. The address space will not be dispatched.
3. ASCBDSP1//ASCBSTND	TCBs not dispatchable. STATUS (IEAVSETS) is stopping SRBs.
4. LOCAL LOCK//ASCBLOCK	
Free (X'00000000')	If any ready work exists in the address space, it can be dispatched.
Other CPUID	If any ready work exists that does not require the LOCAL lock, it can be dispatched.
Interrupt ID (X'FFFFFFFF')	Compare and swap CPUID into LOCAL lock and restore the status (FPRs, GRPs, FRR Stack, CPU timer value, PSATOLD PSATNEW, and resume PSW) from the IHSA.
Suspend ID (X'7FFFFFFF')	If suspension is due to an unsuccessful cross memory services lock request and the holder of the lock has a lower priority, then dispatch the lock holder. Otherwise, if any work exists that does not require the LOCAL lock, that work can be dispatched.

Ready-to-run ID (X'4FFFFFFF')	If the lock is held as a CML lock, then dispatch the lock holder if the lock holder has a lower priority and is dispatchable. Otherwise, if any work exists that does not require the LOCAL lock, that work can be dispatched.
5. ASCBFLG1//ASCBS3S	Interface with Stage 3 exit effector, if the LOCAL lock is available.
6. ASCBTCBS + ASCBTCBL > ASCBCPUS	The LOCAL lock is available. If the total number of ready TCBs (those not requiring the LOCAL lock, plus those requiring it) exceeds the number of processors currently executing in the address space, the address space can be dispatched.
7. ASCBTCBS > ASCBCPUS	The LOCAL lock is not available. There are more ready TCBs (those not requiring the LOCAL lock) than there are processors currently executing in the address space; the address space can be dispatched.

After these seven tests indicate that the dispatcher should dispatch an address space, the following task indicators are tested.

Task Tests	**Condition**
1. TCBFLGS4 + TCBFLGS5	TCB primary non-dispatchability flags must not be set.
2. RBWCF	RB must not be waiting.
3. TCBXSCT1//TCBACTIV	If on, this TCB is active on the other processor (TCBCCPVI), or is suspended holding a local lock.
4. TCBXSCT1//TCBLLREQ	If on, this TCB requires the LOCAL lock before being dispatched.
5. TCBXSCT1//TCBCMLR	If on, this TCB holds a CML lock and is ready to run.
6. TCBAFFN	TCB affinity, if any, must match this processor's (which is located in PCCACAFM).

Miscellaneous Notes About the Dispatcher

1. You can determine the last SRB dispatch by examining the PSW at location X'420' and the last task dispatch by examining the PSW at location X'468'.

   At each dispatch the processor timer is set to a value that will not expire over a 208-day period unless a task has a TQE, in which case it is set to the TQE value.

   For all initial SRBs, the processor timer is also set to the 208-day value. For SSRBs, the processor timer is set to the remaining time value stored in the SSRBCPUT field.

2. The dispatcher sets the following mode indicators before dispatching work.

   a. For a global SRB - LCCADSF2//LCCASRBM, LCCAGSRB, and LCCADSRW

      PSATNEW/PSATOLD = 0's

b. For a local SRB - LCCADSF2//LCCASRBM, and LCCADSRW

PSATNEW/PSATOLD = 0's

c. For a task - LCCADSF2//LCCADSRW

PSATNEW/PSATOLD " 0's TCB address

Dispatcher Recovery Considerations

Dispatcher recovery is designed to record information about the error, reconstruct critical dispatching queues, and to retry to continue normal dispatching functions.

The data that the dispatcher records in the system diagnostic work area (SDWA) is the following:

Fixed Data

SDWAMODN	- IEAVEDS0, dispatcher module name
SDWACSCT	- IEAVEDS0, dispatcher CSECT name
SDWAREXN	- IEAVEDSR, dispatcher recovery routine
SDWACID	- SC1C5, component ID
SDWAMLVL	- module date and level

Variable Data

SDWAVRA - The variable data is written in the key-length-data format. Data items include control block mapping names followed by pairs of field offsets and the contents of the control block at the time of error as follows:

Control Block	**Offset**	**Field Name**	**Description**
IHAPSA	X'224'	PSAAOLD	Current ASCB address.
	X'21C'	PSATOLD	Current TCB address, or zero.
	X'2B0'	PSALOCAL	ASCB address of the CML lock, or zero.
	X'2F8'	PSAHLHI	Locks held indicator.
	X'49C'	PSAMODEW	Word containing PSAMODE.
IHAASCB	X'80'	ASCBLOCK	Local lockword value.
	X'E8'	ASCBLOCI	Address of ASCB holding this ASCB's CML lock.
	X'EC'	ASCBMLH	Address of suspended unit of work holding this ASCB's local lock as a CML lock or LOCAL lock.
	X'B4'	ASCBSRQ	Local dispatcher intersect flags.
	X'13C'	ASCBRCMS	Address of the requested cross memory services lock for which the local lock holder is suspended.
IHALCCA	X'36C'	LCCAFSSJ	SRB journal queue header.
	X'21C'	LCCADSF1	Dispatcher flag bytes.
	X'53C'	LCCAPRMT	Address space promotion indicators.
IHASVT	X'1C'	SVTDSREQ	Global dispatcher intersect flags.

If the dispatcher lock and global intersect can be obtained, the following recovery routines are called by the dispatcher recovery routine:

- IEAVESCR - Scheduler recovery routine; it recovers SRB queues.
- IEAVEQV3 - Verifies, and possibly reconstructs, the ASCB ready queue.
- IEAVEGAS - Verifies each ASCB on the ready queue.

If the LOCAL lock and local intersect can be obtained, the error was not a DAT (dynamic address translation) error; and if the current ASCBSTOR value equaled the CR1 value, then the following recovery routines are invoked by the dispatcher:

- IEAVEEER - Exit effector recovery routine (if the ASCBS3S is on).
- IEAVEQV3 - Verifies, and possibly reconstructs, the TCB queue.
- IEAVETCB - Verifies each TCB on the TCB queue.

Note: The queue verification routine, IEAVEQV3, also records error information in the SDWAVRA about any changes to the queue structure.

By removing elements that have been overlaid (or "clobbered") from the queue, the dispatcher recovery routine attempts to keep the system up at the cost of a particular user, job, address space, etc. There is a certain exposure in this philosophy because the element that has been lost might have owned a critical system resource or might be a critical function in itself (for example, a TCB that represents the user's main application program). If a TCB queue is truncated or found to have invalid data, the address space is terminated. Once the element is lost, there might be no indication that it was a critical resource (a valid control block, for example) or that it owned a critical resource.

Dispatcher Error Conditions

- Abend 075 is issued from CSECT IEAVESC0 when a local SRB is scheduled to an invalid ASCB at the time the SRB is scheduled. The SRB SCHEDULE requester can usually be determined by examining the registers.

- The dispatcher issues the following abends:

Abend Code	Reason Code	Explanation
X'072'	none	A task or SRB is found with CPU affinity to a processor not currently online.
X'22F'	none	There is no usable CPU timer available for tasks with task type TQE, or the processor to which the task has affinity has no operable timer.
X'066'	X'04'	A completed SRB returned to the dispatcher and the SRB holds a lock.
X'066'	X'0C'	An SRB holds the CML lock of an address space that is failing.
X'066'	X'10'	An SRB or SSRB points to an ASCB without a valid ASCB acronym.
X'066'	X'14'	A task holds the CML lock of an address space that is failing.

- Program check interrupts (usually of the page, addressing, or segment exception variety) occur when:

  - PSAANEW is overlaid and the dispatcher attempts to switch address spaces into the value in the PSAANEW
  - PSALCCAV or PSAPCCAV values are overlaid
  - The CVT pointer is overlaid
  - The ASCB ready queue is overlaid
  - The TCB queue or the TCBRBP field is overlaid

SRB/SSRB Pool Manager

The SRB/SSRB pool manager obtains and frees SRBs from the SRB pool and SSRBs (with their associated XSBs) from the SSRB pool. System routines (in key 0, supervisor state) issue the GETSRB, FREESRB, GETSSRB, and FREESSRB macros to request the pool manager services.

This topic describes the following:

- SRB/SSRB pool manager entry points
- SRB/SSRB pool manager recovery considerations
- SRB/SSRB pool manager error conditions

SRB/SSRB Pool Manager Entry Points

The pool manager entry points are:

IEAVSPM1 - entered key 0, supervisor state, enabled for DAT, system mode acceptable to SETFRR (not EUT), no locks required (except the SALLOC might be required for UNCOND and EXPAND type requests.

This entry point is called by the GETSRB macro and obtains an SRB and six-word parameter area from the SRB pool. The SRB is initialized as follows:

- SRB acronym field
- pointer to the parameter area
- FREEMAIN flags, which indicate the origin of the SRB
- other fields and the parameter area cleared.

IEAVSPM2 - entered key 0, supervisor state, enabled for DAT, system mode acceptable to SETFRR (not EUT), no locks required (except the SALLOC lock might be required).

This entry point is called by the GETSSRB macro and obtains an SSRB and its associated XSB from the SSRB pool. The SSRB/XSB are initialized as follows:

- SSRB acronym field
- pointer to a resource management termination routine (RMTR)
- pointer to the SSRB save area
- SSRB pointer to the XSB
- non quiescable and suspended flags set on
- FREEMAIN flags, which indicate the origin of the SSRB/XSB
- XSB acronym field
- other fields cleared.

IEAVSPM3 - entered key 0, supervisor state, enabled for DAT, system mode acceptable to SETFRR (not EUT), no locks required (except the SALLOC lock might be required).

This entry point is called by the FREESRB macro and frees an SRB and its six-word parameter area. If the specified SRB acronym field is not the same as when the SRB was obtained, the program issuing the macro is abended.

IEAVSPM4 - entered key 0, supervisor state, enabled for DAT, system mode acceptable to SETFRR (not EUT), no locks required (except the SALLOC lock might be required).

This entry point is called by the FREESSRB macro and frees an SSRB and its XSB. If the specified SSRB acronym field is not the same as when the SSRB was obtained, the program issuing the macro is abended.

SRB/SSRB Pool Manager Recovery Considerations

When an error occurs, the SRB/SSRB pool manager recovery routine (IEAVSPMR) records information about the error in the SDWA. The queue verifier routine (IEAVEQV1) then uses an SRB/SSRB verification routine in IEAVSPMR to verify that the SRB and SSRB pools are intact. Two tests are used to determine if a given storage area is a valid SRB or SSRB: (1) the storage address must be a valid virtual address, and (2) the acronym field must contain the correct acronym.

If an error is found with a pool, the queue verifier routine attempts to repair the pool, which might include removing invalid SRBs or SSRBs from their pools. Any removed blocks of storage are unavailable for the remainder of the IPL.

The data recorded in the SDWA is:

Fixed data

SDWAMODN	- NUCLEUS, pool manager is nucleus resident.
SDWACSCT	- IEAVESPM, CSECT name.
SDWAREXN	- IEAVESPM, recovery CSECT name.
SDWACID	- SC1C5, component ID.
SDWASC	- descriptive module name.
SDWAMLVL	- module level information.
SDWARRL	- IEAVSPMR, recovery routine label.

Variable data

The variable data in the SDWAVRA is recorded in the key-length-data format.

- FRR parm area - the six-word parameter area passed to IEAVSPMR by the mainline routine is as follows:
  - Mainline SALLOC footprint - indicates if the SALLOC lock was held on entry to the pool manager.
  - FRR SALLOC footprint - indicates if the SALLOC lock was held on entry to the recovery routine.
  - Return address - contents of register 14 on entry to the pool manager (caller's return address).
- ASCBASID - address space ID of the current ASCB.
- PSATOLD - address of the current TCB.
- General register 14 - contents of register 14 on entry to the mainline pool manager (caller's return address).
- Pool problem information - information recorded by the queue verifier routine if problems are found with the SRB or SSRB pools.
- Lock name - SALLOC, indicates that the SALLOC lock was held by the pool manager mainline routine.

SRB/SSRB Pool Manager Error Conditions

If the IEAVSPM3 (FREESRB) or IEAVSPM4 (FREESSRB) routines are called and an error is detected, completion code X'05A' is issued and the caller is abended. Register 2 contains the address of the invalid SRB or SSRB.

Refer to *System Codes* for a description of code X'05A' and specific reason codes in register 15.

Stop/Reset Services

When a unit of work (a current task or SRB) has been dispatched and is executing, the unit of work might need to be suspended. For example, to satisfy a page-in due to a page fault.

System routines (in key 0, supervisor state) use the stop/reset service to suspend and then reset a unit of work. The caller is not required to have addressability to the home address space to suspend a unit of work, and is not required to have addressability to the address space containing the unit of work to reset the unit of work.

This topic describes the following:.

- Stop/reset entry points
- Stop/reset recovery conditions
- Stop/reset error conditions

Stop/Reset Entry Points

The stop/reset entry points are:

IEAVSUSP - entered disabled, key 0, supervisor state, no locks required (except the SALLOC lock might be required).

This entry point is called by the paging supervisor to suspend a current task or SRB because a page fault occurred.

IEAVSUSQ - entered disabled, key 0, supervisor state, no locks required (except the SALLOC lock might be required).

This entry point is called by system routines (other than the paging supervisor) to suspend the current task or SRB.

IEAVRSET - entered disabled, key 0, supervisor state, no locks required (except the SALLOC lock might be required.

This entry point is called by the paging supervisor to reset a task or SRB that was suspended because of a page fault.

IEAVRSTD - entered disabled, key 0, supervisor state, no locks required (except the SALLOC lock might be required).

This entry point is called by system routines (other than the paging supervisor) to reset a task or SRB that was suspended.

Stop/Reset Recovery Considerations

The stop recovery routine (STOPFRR) records information about the error and, depending on the error, either attempts to restore the system and unit of work to a consistent state, or attempts to complete the stop function.

The reset recovery routine (RESETFRR) records information about the error and then attempts to complete the reset function.

The reset STERM, reset schedule, and reset SRB recovery routine (IEAVSCHF) frees the SRB (if one was obtained but not scheduled), clears the stop/reset super bit (PSASTPRT), and releases the LOCAL lock (if the LOCAL lock was obtained by the calling routine).

The data that the stop/reset recovery routines record in the SDWA is:

Fixed data

The fixed data recorded by all of the recovery routines is:

SDWAMODN	- NUCLEUS, stop/reset is nucleus resident.
SDWACSCT	- IEAVESRT, CSECT name.
SDWAREXN	- IEAVESRT, recovery routine.
SDWACID	- SC1C5, component ID.
SDWAMLVL	- module level information.
SDWARRL	- STOPFRR, RESETFRR, or IEAVSCHF, label of the recovery routine.

Variable data

The variable data in the SDWA is recorded in the key-length format. The variable data recorded by the recovery routine is:

For the stop recovery (STOPFRR) and the reset recovery (RESETFRR) routines:

- FRR parm area - the six-word parameter area passed to STOPFRR and RESETFRR by the mainline routine is as follows:
  - General register 13 - caller's register save area address.
  - TCB/SSRB address - address of the TCB or SSRB to be reset.
  - RB address - address of the RB if a TCB is to be reset.
  - Request code - type of reset requested (conditional, unconditional, or page I/O error), or completion code for a termination reset.
  - Flag byte - if X '80', recovery has been entered recursively.
- ASCBASID - address space ID of the current ASCB.
- PSATOLD - address of the current TCB.
- General register 14 - contents of register 14 on entry to the mainline stop routine (caller's return address).

- General registers - for STOPFRR, contents of the original registers, if they were changed by the recovery routine.

- Abend code - for STOPFRR, the original abend code, if it was changed by the recovery routine.

For the reset STERM, reset schedule, and reset SRB recovery routine (IEAVSCHF):

- FRR parm area - the six-word parameter area passed to IEAVSCHF by the mainline routine as follows:

  - SSRB address - address of the SSRB associated with the scheduled SRB. (Note that an SSRB is obtained, made to look like an SRB, and scheduled as an SRB. The remainder of the SSRB is used as a work area by the scheduled routine. The SSRB is restored to an SSRB before it is returned to the SSRB pool.)

  - Flag byte - recovery footprint flags:

    X'80' - stop/reset super bit footprint, indicates the mainline code set the bit.

    X'40' - LOCAL lock footprint, indicates the mainline code had obtained the LOCAL lock and had not released it before the error occurred.

- ASCBASID - address space ID of the current ASCB.

- PSATOLD - address of the current TCB.

- If an SRB was obtained but not scheduled, the following are also present in the SDWAVRA:

  - IHASRB - identifies the following control block.
  - SRB - contents of the unscheduled SRB.
  - SRB parm area header - describes the following six-word parameter area.
  - SRB parm area - contents of the SRB parameter area.

- Lock name - LOCAL, indicates the LOCAL lock was held.

Stop/Reset Error Conditions

The stop/reset services issue the X'059' completion code when an error exists, and abnormally terminates the program requesting the service.

Refer to *System Codes* for a description of code X'059' and specific reason codes in register 15.

SUSPEND/RESUME/TCTL Services

The SUSPEND/RESUME/TCTL services are used to place an unlocked task in a suspended state (SUSPEND), resume an unlocked task from a suspended state (RESUME), and to transfer control from an SRB to an unlocked task (TCTL). These macros can only be issued by key 0, supervisor state routines.

SUSPEND can be issued in any cross memory mode and in task mode; it places the caller in a suspended state. Control is returned to the caller and the task is suspended only when the task incurs an interruption or enters the dispatcher (such as via CALLDISP).

RESUME can be issued in any cross memory mode and in SRB or task mode, with current addressability to the address space of the TCB that is to be resumed.

TCTL can be issued in SRB mode and home mode, with current addressability to the address space of the task to which control is to be transferred.

This topic describes the following:

- SUSPEND/RESUME/TCTL entry points
- RESUME/TCTL recovery considerations
- SUSPEND/RESUME/TCTL error conditions

SUSPEND/RESUME/TCTL Entry Points

The SUSPEND/RESUME/TCTL entry points are:

IEAVSPND - entered enabled or disabled, key 0, supervisor state, task mode, no locks held, any cross memory state.

This entry point is called by the SUSPEND macro and places the current TCB/RB in a suspended state.

IEAVRSUH - entered enabled, key 0, supervisor state, no locks held, SRB or task mode, home mode.

This entry point is called by the RESUME macro to resume a task in the home address space. This entry point performs an unconditional synchronous resume function. The caller must execute enabled and hold no locks unless the LOCAL lock is already held, because this entry point can require the LOCAL lock to serialize the resume function. This is the only entry point where RETURN=N can be specified to indicate that control should be transferred from the calling SRB to the resumed task.

IEAVRSUS - entered enabled, key 0, supervisor state, no locks held, SRB or task mode, any cross memory state, current addressability to the resumed TCB.

This entry point is called by the RESUME macro to resume a task in the address space specified by the input. Current addressability to the task to be resumed must have been established by the caller. This entry point performs an unconditional synchronous resume function. The caller must execute enabled and hold no locks unless the LOCAL lock of the address space of the resumed TCB is already held, because this entry point can require the specified lock to serialize the resume function.

IEAVRSCS - entered enabled or disabled, key 0, supervisor state, SRB or task mode, any cross memory state, locks can be held, current addressability to the resumed TCB.

This entry point is called by the RESUME macro to resume a task in the address space specified by the input. Current addressability to the task to be resumed must have been established by the caller. This entry point performs a conditional synchronous resume function. If serialization to perform the resume function is not available, the function is not performed and the caller receives a nonzero return code.

IEAVRSUA - entered enabled or disabled, key 0, supervisor state, SRB or task mode, any cross memory state, locks can be held, current addressability to the resumed TCB.

This entry point is called by the RESUME macro to resume a task in the address space specified by the input. Current addressability to the task to be resumed must have been established by the caller. This entry point performs an unconditional asynchronous resume function. If serialization to perform the resume function is not available, an SRB is obtained and, asynchronously, an unconditional synchronous function is performed; a nonzero return code is returned to the caller.

IEAVRSCA - entered enabled or disabled, key 0, supervisor state, SRB or task mode, any cross memory state, locks can be held, current addressability to the resumed TCB.

This entry point is called by the RESUME macro to resume a task in the address space specified by the input. Current addressability to the task to be resumed must have been established by the caller. This entry point performs a conditional asynchronous resume function. If serialization to perform the resume function is not available, an SRB is obtained conditionally, and if successful, asynchronously scheduled to perform an unconditional synchronous resume function. Return codes are returned to the caller to indicate whether the SRB could be obtained or not obtained.

IEAVTCTL - entered enabled or disabled, key 0, supervisor state, SRB mode, home mode, no locks held.

This entry point is called by the TCTL macro to transfer control from an SRB to a task in the home address space.

IEAVETCR - entered disabled, key 0, supervisor state, any cross memory state, SRB or task mode.

This entry point is called by RTM and performs recovery processing for the resume and transfer control functions.

RESUME/TCTL Recovery Considerations

RESUME/TCTL processing is protected by an FRR (IEAVETCR) that receives control from RTM when an error occurs. The FRR records debugging information in the SDWA, attempts to restore the system and unit of work to a consistent state, and then percolates to the caller's recovery routine.

The data recorded in the SDWA is:

Fixed data

SDWAMODN - NUCLEUS, nucleus load module.
SDWACSCT - IEAVETCL, mainline microfiche name.
SDWAREXN - IEAVETCL, recovery microfiche name.
SDWACID - SC1C5, component ID.
SDWASC - RESUME ABEND or TCTL ABEND, subfunction in error.
SDWAMLVL - module level information.
SDWARRL - IEAVETCR, recovery routine name.

Variable data

Variable data in the SDWAVRA is recorded in the key-length-data format. A header contains RSLG in key-length-data format followed by the RSLG mapping in key-length-data format as follows:

Offset	RESUME data	TCTL data
X'0'	flag byte: Bit 0=0,RESUME	flag byte: Bit 0=1,TCTL
X'1'	0	SVTDACTV
X'2'	flag byte: Bit 0=1, lock obtained Bit 1=1, SRB obtained	0
X'4'	PSASUPER	PSASUPER
X'8'	PSACSTK	PSACSTK
X'C'	PSATOLD	PSATOLD
X'10'	PSAAOLD	PSAAOLD
X'14'	input TCB address	input TCB address
X'18'	input ASCB address	SVTDSREQ
X'1C'	PSAHLHI	ASCBSRQ
X'20'	home address space ID	-
X'22'	0	-
X'24'	ASCBTCBS	-
X'28'	address of SRB, or 0	-

SUSPEND/RESUME/TCTL Error Conditions

The SUSPEND, RESUME, and TCTL macros issue the X'070' completion code when an error condition exists, and abnormally terminate the program issuing the macro.

Refer to *System Codes* for a description of code X'070' and specific reason codes in register 15.

IOS

The purpose of the I/O supervisor (IOS) is to provide a central facility to control and conduct I/O activity through the operating system. The structure of IOS in MVS is somewhat different than that of previous operating systems. In MVS, IOS "front end processing" is responsible for device control and I/O initiation; IOS "back end processing" is responsible for processing interrupts, providing sense information in error situations, and scheduling the posting of the I/O requestor at completion time. Figure 5-4 provides an overview of I/O front-end and back-end processing. Figure 5-5 shows the major IOS and EXCP control block relationships.

Front-End Processing

The major portion of the I/O process (the queueing of I/O requests and starting them) is contained in CSECT IECIOSCN (microfiche name IECIOSAM), which is called the channel scheduler. The channel scheduler is invoked through an interface provided by the STARTIO macro via a branch entry. The channel scheduler assumes that all channel program translation and page fixing of buffers and CCWs is performed by the caller. The control block interface is the SRB/IOSB combination, which must be non-pageable and commonly addressable from any address space (that is, SQA and fixed CSA). The channel scheduler operates in physically disabled mode. Invokers (called "drivers") of the channel scheduler include EXCP, VSAM block processor, VTAM TPIOS, and PCI fetch; they are identified by the driver ids located in the IOSB+4 (IOSDVRID).

Back-End Processing

When IOS is invoked for an I/O interrupt, processing starts in the I/O first level interrupt handler (FLIH) which branches to an entry point, IECINT, within the channel scheduler. Back-end IOS executes physically disabled in the address space that is active on the processor at the time of the I/O interrupt. IOS then schedules the SRB/IOSB to the address space of the requestor. The module IECVPST (post status) receives control under the SRB and interfaces with the driver's special exits and termination routines (channel end, abnormal end exits). Figure 5-4 shows an overview of the I/O process using EXCP as the I/O driver.

IOS Problem Analysis

Problems in the I/O process can cause three symptoms:

1. Abend codes
2. Loops
3. Wait states

These symptoms are discussed in the following sections.

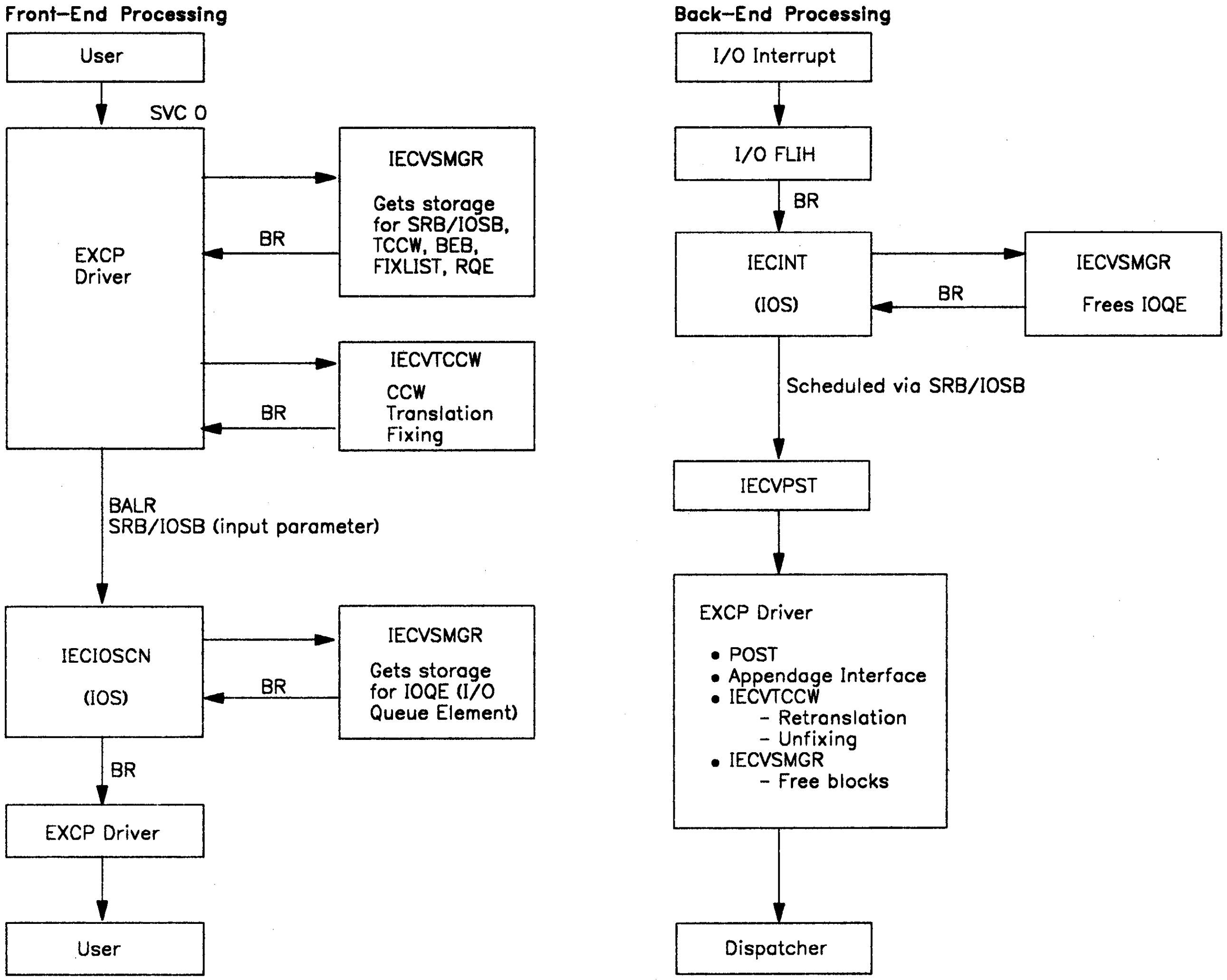


Figure 5-4. I/O Processing Overview

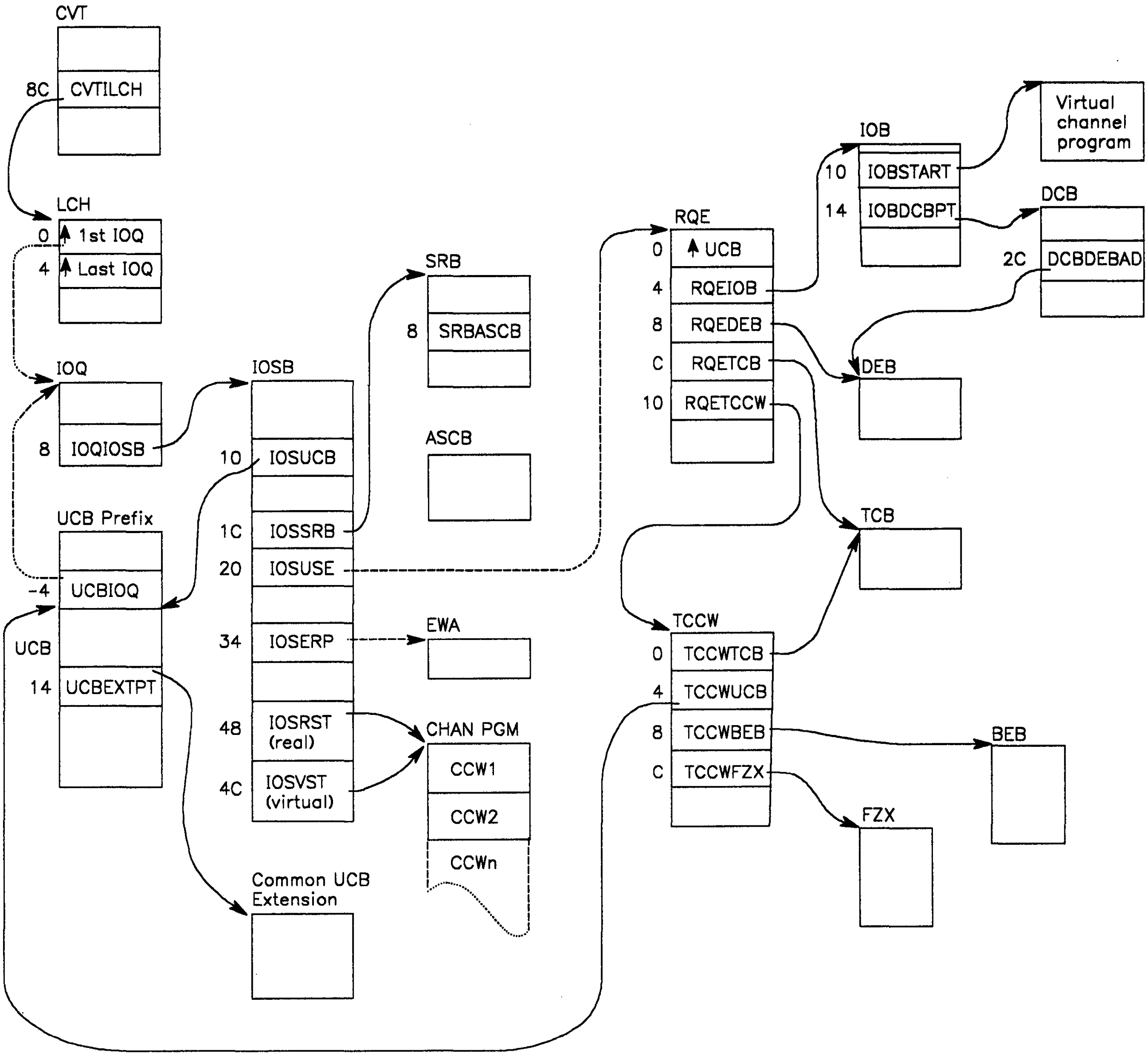


Figure 5-5. Major IOS and EXCP Control Block Relationships

IOS ABEND Codes

IOS abends are generally caused by an invalid control block. The error can be caught by validity checking or it can cause a program check. The recovery routines, generally FRRs, receive control on a program check. For either a validity check or a program check, the error is converted to an abend code. EXCP FRR processing saves the abend code and the relevant status (that is, error PSW, and error registers) at the time of error in the EXCP problem determination area, which is pointed to by the TCB (X'C0'). IOS abend codes are documented in *Message Library: System Codes.* The EXCP problem determination area is documented later in this section in the topic "The EXCP Debugging Area." A list of IOS abend codes and issuing procedures is found in the topic "Wait State Codes."

Note: During abend processing, the EXCP problem determination areas are not freed. When you find the area pointed to by the TCB, scan that area for previously-obtained areas to help with IOS analysis.

Loops

If an invalid control block is passed to IOS and it is not caught by the validity check routines, a loop is often the result. The traditional problem has been caused by a driver that reuses the IOSB before its first use is complete. Consequently, the requestor initializes the block and overlays some field whose use is not complete. On occasion the block is cleared to zeros. The fact that most of the I/O process runs in supervisor state, key 0 means that the PSA can be overlaid. This usually causes a program check loop whenever any type of interrupt is subsequently received by the processor.

At this point, pattern recognition is important to determine whether the storage manager has been involved in the problem. (Pattern recognition is discussed in the "Miscellaneous Debugging Hints" chapter in Section 2.) Try to determine whether 0 has been used as the address of an SRB/IOSB or EWA control block. The first X'A0' bytes of PSA may be affected. The routine responsible for this could be an IOS driver or recovery routine. Look for addresses of exit routines which are pointed to by the IOSB; they give an indication of the driver and potentially some idea of the process. Remember that the hardware stores the current PSW as an old PSW (at locations X'18' - X'40') if any interrupt occurs. Therefore these locations may not look bad.

Often double freeing has occurred some time earlier, which makes the recreation of the erroneous process very difficult. Extensive analysis and piecing are required. Multiple dumps may help provide the pieces necessary to recognize a pattern or common occurrence. Or, a trap might have to be devised.

If there is evidence of a recent error in the I/O process, searching the in-storage LOGREC buffer or SYS1.LOGREC records for an IOS error helps recreate the process. Generally the IOS recovery routines attempt to free control blocks and might inadvertently free one that has just been freed. Try to determine if there is any way that the channel scheduler or I/O driver and its associated exits could have freed blocks before or after recovery processing. In a retry situation, normal termination procedures could have freed a block that was already freed by recovery. Again, traps might be required.

IOS WAIT States

Another problem is an enabled wait state with work remaining for IOS to accomplish. For a list of IOS wait state codes and the procedures that issue them, refer to the topic "Wait State Codes" later in this section. To analyze a wait state, it is necessary to determine the current status of IOS.

To determine current IOS status, scan the UCBs for valid IOQEs in UCBIOQ (UCB-4). The IOQE is valid if UCBPST (UCB+6, bit X'20') is on. The IOQE address is valid only when it is active. Understand that once a block is freed, it is generally reused quickly when a subsequent request for an I/O operation is encountered. Because of this, it is very uncommon to find a significant IOQE pointed to by the UCB prefix once IOS has returned the block. The block usually represents another request. If the UCB pointer in the IOQE does not equal the address of the UCB you started with, the blocks have been reused and the data is invalid.

Additionally the IOQEs can be found in the storage manager areas. These are located by CVT+X'7C' which points to IOCOM+X'24' which points to module IECVSMGR. Label IECVSHDR is an external symbol for the storage pool headers for small blocks (IOQEs). These are followed by the pool headers for medium (RQEs) and large (SRB/IOSBs, BEB, TCCW, ERPWA, fix lists) blocks. The pool headers are 16 bytes long and the last word points to segment headers for 2K bytes (small block) or 4K bytes (medium and large blocks) of storage. The IOQE+5 contains an allocated indicator. If all X'3C' bits are on, the block is allocated and, in the case of IOQEs, represents I/O requests that are started or that have been requested by a driver but have not been started because of a busy or not ready condition (UCBFLA).

After the storage manager (medium and large) blocks are found, notice their 8-byte prefixes, the first halfword of which contains the ASID of the address space to which the block is allocated. Note that the ASID is 0 when the block is not allocated and in special cases such as when unsolicited device ends are not associated with any address space. Scanning these prefixes for an ASID that matches the problem address space can help in finding blocks associated with I/O requests related to that address space. Medium and large blocks that contain a X'17' in the fourth byte of the prefix are not allocated. A value of X'75' for medium blocks, and X'76' for large blocks, indicates that they are currently allocated. (Note that the third byte of this first word of the prefix is unused.)

The IOQE points to the associated IOSBs which contain information about the channel programs and pointers to the requestor's control blocks.

In general, UCBs and associated IOQEs/IOSBs indicate active I/O. Any flag bits set in the UCB+6/7 help identify the status of the requestor. Also, investigate UCB flags indicating the quiesce option, DAVV (direct access volume verification) processing, I/O restart, missing interrupt handler (MIH), or message pending.

Another place to look is the LCHs (logical channel queues). When a STARTIO macro is issued, if both the channel and device are available, IOS attempts to issue the SIO instruction. If any bit in UCBFLA (UCB+6) is on, the device is considered busy. The TCH instruction is used to determine if the channel is busy. If either is busy, the IOQE for the request is queued to the LCH. This queue then indicates all requests that have been accepted for processing but for which either no SIO has yet been issued or an SIO was issued but a non-zero condition code was received. The first LCH is pointed to by CVT+X'8C'. Each LCH is X'20' bytes long. UCBLCI (UCB+X'A') is an index to the LCH for the given UCB. Each LCH is a double-headed, single-threaded queue of IOQEs. The LCH+0 points to the first IOQE and LCH+4 points to the last, or only, IOQE. If LCH+0 is all Fs or 0s the queue is empty, in which case there are no requests for that channel. The IOQEs themselves are linked with IOQELNK (IOQE+0). IOQEIOSB (IOQE+8) points to the IOSB for the request it represents. Note that IOQENQ (IOQE+4, bit X'40') must be on for all IOQEs on the LCH.

General Hints For IOS Problem Analysis

1. *Saveareas.* IOS does not use save areas in the standard manner. When registers are saved, the order is often 0-15 at offset 0 into the save area. If the local lock is obtained (as is generally the case), IECVPST, the first module to execute in the user's address space after an I/O interrupt, uses the local lock save area (ASXBFSLA at ASXB+X'24') to pass the address of the local lock save area to the exit routines. An exception is I/O interrupt processing for a paging pack where an IOS storage manager or ASM area is used. Basic IOS uses the IOS save area (LCCA+X'218' points to the CPU work save area vector table (WSAVTC); WSAVTC+X'18' points to the IOS save area). This save area is also passed to DIE (Disable Interrupt Exit) routines. Also, the TCCW control block contains a save area. EXCP passes the address of the associated TCCW+X'48' (in Register 13) to appendages for use as a save area.

2. EXCP back-end processing does all the interfacing to the traditional appendages. In MVS, appendages are entered in SRB mode, physically enabled, and with register 13 containing the address of a save area.

   It is EXCP's responsibility to map the IOSB to the IOB to maintain compatibility. Also on return from the appendage, EXCP re-maps the IOB to the IOSB.

3. The EWA (ERP work area) can be important in problem analysis. The IOSB+X'34' points to EWA, which contains information, including sense data, passed to the ERPs from IOS as well as work areas and counters for the ERPs. The ERPIB, which is useful for channel errors, is contained in the EWA.

   See the topic "Error Recovery Procedures (ERPs)" later in this section for a description of ERP processing.

   Several problems have been uncovered where ERPs constantly retry an I/O operation that constantly fails. The EWA can contain the number of retries and other control information helpful in determining the reason why. EWAs often contain the retry CCWs.

4. The LCCA of each processor contains an IRT (IOS recovery table). IOS uses various fields in the IRT to checkpoint its progress. The IRT also contains pointers to the active control blocks on whose behalf IOS is processing.

5. Two IOSB flags (IOSEX, IOSERR) are used to control error processing. For a permanent error the general flow is:

   - Abnormal or normal exit initially entered with IOSCOD = 7F, IOSEX = 0 or 1, IOSERR = 0.

   - ERP exit entered with IOSCOD = 7F, IOSEX = 1, IOSERR = 0.

   - SVC F or branch entry back to IECVPST for direct access (DA) ERP:

     with IOSERR = 1, IOSEX = 0 for retry
     with IOSERR = 0, IOSEX = 1 for permanent error

   - Assuming retry, SVC F issues STARTIO.

   - At I/O completion, IECVPST returns control to ERP with IOSERR = 1, IOSEX = 1.

   - ERP returns to IECVPST with IOSERR = 0, IOSEX = 1 to indicate a permanent error.

   - IECVPST enters abnormal exit for second time with IOSCOD = 41, IOSERR = 0, IOSEX = 1.

   - Abnormal exit returns to IECVPST for termination processing.

   In general, the IOSB flag settings are defined as:

IOSERR = 0
IOSEX = 0
	no error or corrected error

IOSERR = 1
IOSEX = 1
	ERP retry in progress

IOSERR = 1
IOSEX = 0
	ERP requesting retry

IOSERR = 0
IOSEX = 1
	permanent error

6. I/O error processing during ACR has caused several problems. The chapter "Miscellaneous Debugging Hints" in Section 2 addresses the ACR processing and potential exposures.

7. Check the trace table for unit check/unit exception interrupts. These interrupts often cause abnormal processing which may contribute to the problem. (For information on "MVS Trace Analysis," see that chapter in Section 2.) The fourth word of the SIO trace entry is the IOSB address associated with the I/O request. The SRB + X'1C' points to the IOSB address associated with the interrupt that caused the post status module (IECVPST) to be scheduled.

8. Check the LOGREC created by IOS modules (the CSECT name in the record will be an IOS module name). Register 2 quite often is the IOSB address associated with a request to be processed at the time of error.

9. Prior to SU64, a channel on processor 0 was distinguished from a channel with the same number on processor 1 by the processor address. This address is contained in such fields as UCBCPU and EWACPU. With channel set switching, the channel set ID is not used to distinguish between two similarly numbered channels. As a result, control block fields contain the channel set ID even though the field name is shown as 'CPU'.

10. By using the GTF CCW trace option, you can trace the CCWs associated with a given SIO or I/O operation along with data to show what was presented to the channels. The EWAs (see hint 3) and the IOSB (see hint 5) can be printed as part of the trace information.

IOS Diagnostic Aids

Both EXCP and IOS diagnostic aids are described in this chapter under separate headings. For detailed information about IOS modules and procedures, refer to *I/O Supervisor Logic.*

Table of EXCP Abend Codes

The following table lists abend codes with the IOS module and symbolic names of the EXCP procedures that issue them. For the meanings of the abend codes, refer to *System Codes.*

Code	Module	Procedure - Name
X'15C'	IECVEXCP	XCP000 - Validity check
X'172'	IECVEXCP	XCP000 - Validity check
X'200'	IECVEXPR	XCPFRR - Functional recovery
X'300'	IECVEXCP	XCP000 - Validity check
X'400'	IECVEXCP	XCP000 - Validity check
X'500'	IECVEXCP	XCP000 - Validity check
X'700'	IECVEXCP IECVEXPR	XCPTER - Termination XCPFRR - Function recovery
X'800'	IECVEXCP IECVEXCP IECVEXCP	XCPPFA - PGFX interface XCPTERM - Termination XCP115 - Translation interface
X'A00'	IECVEXCP IECVEXPR	XCPTERM - Termination XCPFRR - Functional recovery
X'B00'	IECVEXPR	XCPFRR - Functional recovery
X'C22'	IECVEXCP	XCP035 - Get RQE
X'E00'	IECVEXCP	XCPTERM - Termination

EXCP Debugging Area (XDBA)

EXCP's functional recovery procedure, XCPFRR, does not put diagnostic data in the SDUMP buffer. Instead, it gets storage for its own debugging area (the XDBA) and puts diagnostic data there. (Note that an XDBA is not provided for E00 abend codes.)

To locate the debugging area (XDBA) in a SYSABEND, SYSMDUMP, or SYSUDUMP dump, you must:

1. Get the address of the CVT from location X‘4C’ (PSA field FLCCVT2) in the dump.

2. Get the address of the TCB from the first word of the CVT (CVTTCBP).

3. Look X‘C0’ bytes into the TCB (TCBEXCPD) and get the address of the debugging area.

4. If the address of the debugging area is zero then no debugging area is available.

The format and contents of the EXCP debugging area (XDBA) are as follows:

Byte Contents

0 The ABEND code that EXCP issued.

2 A byte that shows where the error occurred. These are the possible bit settings and their meanings:

X‘80’: The error occurred while EXCP was preparing to send an I/O request to IOS.

X‘40’: The error occurred while EXCP was processing an I/O request that IOS was finished with.

X‘21’: The error occurred in a PCI appendage.

X‘11’: The error occurred in a CHE appendage.

X‘09’: The error occurred in an ABE appendage.

X‘05’: The error occurred in an EOE appendage.

X‘03’: The error occurred in a PGFX appendage.

X‘01’: The error occurred in an SIO appendage.

3 Reserved

4 The PSW before RTM was entered. (RTM is entered when a program check occurs or when an ABEND macro is issued.)

C Reserved

E ABEND code at entry to the FRR.

10 The register contents before RTM was entered.

50 Translation exception address.

54 The RQE for the I/O request that was being processed.

7C XDBA chain pointer

The remainder of the debugging area contains up to twelve 160 byte blocks involved with the EXCP request. If these blocks are present, they appear in the following sequence:

EWA
SRB/IOSB
TCCW
IDAL
FIX list
BEB
...

The first 160 bytes following the last block are zero. The SRB and TCCW are valid only if the address of the RQE within these blocks is valid.

Note: For errors that occur in the PCI appendage during disabled interruption exit (DIE) processing, IOCIOSCN provides a SYS1.LOGREC record and an SVC dump. The register contents and PSW at the time of the original error are contained in the SYS1.LOGREC record and the dump. EXCP uses the DIE exit when processing V=R and EXCPVR requests.

SDWA Variable Recording Area

The format and contents of the SDWA variable recording area are as follows:

Byte (offset in SDWA)	**Contents**
194	original ABEND code
196	adjusted ABEND code set by XCPFRR
198	highest lock held word from the PSA
19C	contents of the 6 word FRR parameter area
1B4	contents of the active RQE
1DC	TCCW option byte
1DD	TCCW translation flag byte
1DE	IOSB completion code from the IOSCOD field
1DF	reserved
1E0	ASID of the EXCP request

Output of IOS Recovery Procedures

Functional (FRR) and ESTAE recovery procedures can record their virtual storage environment by two means:

- By issuing an SDUMP macro, which causes the contents of the 4K SDUMP buffer to be written in a SYS1.DUMP data set. (There are ten SYS1.DUMP data sets, SYS1.DUMP00-09.)

- By issuing a SETRP macro, specifying RECORD=YES, which directs RTM to write the SDWA in the SYS1.LOGREC data set.

Some Facts about SYS1.DUMP Dumps

To format a dump for a SYS1.DUMP data set, use the AMDPRDMP service aid. If the dump contains an SDUMP buffer record that was put in the SYS1.DUMP data set by an IOS recovery procedure, each page will be titled "IOS-*module name ERROR*," where *module name* identifies the module to which the recovery belongs.

To locate the SDUMP buffer record, you must:

1. Get the address of the CVT from location X'4C' (PSA field FLCCVT2) in the dump.

2. Look X'24C' bytes into the CVT (CVTSDBF) and get the address of the SDUMP buffer record.

The third halfword of the SDUMP buffer record tells you how much of the 4K bytes contains meaningful data; six bytes of zeros mark the end of the meaningful data.

Some Facts about SYS1.LOGREC Dumps

To get a dump of the SYS1.LOGREC data set, use the IFCEREP1 service aid. IFCEREP1 formats the standard area - the first 404 bytes - of each SDWA into a series of titles, each followed by pertinent data found in the standard area. (For example, under the title *Component/Module/ Name/ID,* you would find the module name *IECIOSCN* if the functional recovery procedure of the basic IOS module wrote in the SDWA.) IFCEREP1 puts the variable area - the last 108 bytes - of each SDWA in a decimal or hexadecimal format, whichever you request.

The remaining topics in this section describe the output - the SDUMP buffer record and SDWA variable areas - of IOS recovery procedures. Before looking at the descriptions for the first time, note these facts:

- Offsets into SDUMP buffer records and SDWA variable areas are given in hexadecimal numbers.

- The formats of data areas listed as part of an SDUMP buffer record or SDWA variable area are shown in the microfiche document *Data Areas,* unless stated otherwise.

Output of the Basic IOS Module (IECIOSCN)

The module's functional recovery procedure, IECFRR, puts one or more of these settings in byte 6 of the SDUMP buffer record:

X'80', indicating that an IRT is in the record, beginning at byte 8.

X'40', if a UCB is in the record.

X'20', if an IOQ is in the record.

X'10', if an IOSB is in the record.

X'08', if a logical channel queue, the header of the "small block" pool, and the first 2048-byte segment of the pool are in the record.

If a UCB lock was held when IECFRR was entered, the UCB associated with that lock appears in the record. If an LCH lock was held when IECFRR was entered, the LCH associated with that lock, the header of the "small block" pool, and the pool's first 2048-byte segment is included. If an I/O request was being processed, its IOQ and IOSB appears. These data areas appear in this order: UCB, IOQ, IOSB, logical channel queue, "small block" pool header, first "small block" pool segment.

Other 4K records and the output of IECFRR follow the SDUMP buffer record in the dump. These records contain the SQA (system queue area), the system's trace tables, and the nucleus.

IECFRR puts the following data in the variable recording area of the SDWA:

At byte 0: X'80', if an IOQ is in the SDWA. X'40', if a UCB is in the SDWA.

At byte 1: the code that was returned when IECfrr issued an SDUMP macro to write in the SYS1.DUMP data set. (X'FF' means nothing was written.)

At byte 4: an IOQ, if an I/O request was being processed when IECFRR was entered.

At byte 10: a UCB (prefix segment included), if a UCB lock was held when IECFRR was entered.

Output of the Build Reserve Table Module (IECVBRSV)

The functional recovery routine (BRSVFRR) of IECVBRSV issues an SDUMP macro requesting an SQA, nucleus, all PSAs and a summary.

BRSVFRR puts the 24-byte FRR parameter area into the SDWA variable recording area.

Output of the DAVV Module (IECVDAVV)

The module's ESTAE recovery procedure, DAVVESTA, puts the following data in the SDUMP buffer record:

At byte 6: the SRB being processed when it was entered.

At byte 32: the IOSB being processed when it was entered.

At byte 9E: the ERP work area used by DAVV. (The work area includes the common segment, EWA, and the direct-access segment, EWD.)

At byte 13E: the UCB (prefix segment included) used in the processing that preceded the error.

DAVVESTA puts the following data in the variable area of the SDWA:

At byte 0: the IOSB being processed when it was entered.

At byte 6C: X'04', a code indicating that DAVVESTA asked RTM to return control instead of continuing termination processing.

Output of the Dynamic Pathing Initialization Module (IECVIOSI)

This module's ESTAE procedure (IOSIRECV) issues the SDUMP macro.

IOSIRECV puts the following data in the variable recording area (SDWAVRA) of the SDWA in a key-length-data format:

- Component ID
- Date and SU or PTF ID
- ENF parameter list (ENFPM) or EVARY parameter list
- Flag field (includes ENF and/or ESTAE return codes)
- ESTAE parameter list
- Path group ID field, if created

Output of the Dynamic Pathing Module (IECVDPTH)

This module's functional recovery procedure (DPTHFRR) issues the SDUMP macro.

DPTHFRR puts the following data in the variable recording area (SDWAVRA) of the SDWA in a key-length-data format:

- Component ID
- Date and SU or PTF ID
- Parameter list (EVARY)
- Footprint and flags fields (FLAGSDP)
- Path group ID field (HOSTIDEN)

If IECVDPTH cannot complete the dynamic pathing request, an OBR-DPA record (type X'3A') is written to SYS1.LOGREC. The device-dependent data in the record contains the following two 12-byte fields:

First field:	1 byte	- Function control byte (contains the requested function)
	11 bytes	- Path group ID for the system
Second field:	1 byte	- Function control byte (contains the status of the path)
	11 bytes	- Path group ID for the path, if one exists

Output of the Force Device Module (IECVFDEV)

This module's functional recovery routine (FDEVFRR) issues the SDUMP macro.

FDEVFRR puts a copy of the FRR work area in the variable recording area (SDWAVRA) of the SDWA in a key-length-data format.

The FRR work area contains:

Word 0: UCB common address.
Word 1: Module base register.
Word 2: Data pointer register.
Word 3: Flags and footprints:
Byte 0: Function indicator flags:
X'80' UCB lock held.
X'40' Device to be boxed.
X'20' Device to be released.
Byte 1: Footprints:
X'80' Initialization complete.
X'40' UCB data stored.
X'20' Halt data transfer complete.
X'10' UCB lock released - Clear reserve complete.
X'08' Dynamic pathing call is complete.
X'04' Simulation of interrupts is complete.
X'02' FRR entered due to an error.
Byte 2: Reserved.
Byte 3: Reserved.
Word 4: Reserved.
Word 5: UCB lockword address.

Output of the Force Channel Offline Module (IECVFCHN)

This module's functional recovery routine (FCHNFRR) issues the SDUMP macro.

FCHNFRR puts the following data in the variable recording area (SDWAVRA) of the SDWA in a key-length-data format:

- A copy of the FRR work area, which contains:

Word 0:
Byte 0: Channel set ID.
Byte 1: Channel number.
Byte 2: Function indicator flags:
X'80' The other processor receives control through EMS SLIH due to RISGNL.
X'40' The error occurred while the other processor was in control.
X'20' Do not retry RISGNL.
X'10' Free any extra reserve table segments that were obtained by the build reserve table routine.
X'08' The IEA019A message was issued.
X'04' The clear channel instruction was issued.
X'02' SALLOC lock held.
Byte 3: Footprint indicator flags:
X'80' Initialization completed.
X'40' Force channel offline subroutine entered.
X'20' Load wait state subroutine entered.
X'10' Free reserve table segments subroutine entered.
Word 1: Module base address.
Word 2: Module workarea address.
Word 3: Module savearea address.
Word 4: FRR retry address.
Word 5: FRR 200-byte workarea pointer.

- The first reserve table segment if any devices were entered in the reserve table segment.

FCHNFRR loads a nonrestartable wait state if reserves have been released without rereservation.

Output of the IOS Restart Support Module (IECVRSTS)

This module's functional recovery routine (RSTSFRR) puts the following data in the variable recording area (SDWAVRA) of the SDWA in a key-length-data format:

- A copy of the FRR work area, which contains:

  Word 0: SRB address.
  Word 1: The SRB and module work area serialization byte address.
  Word 2: FRR retry address.
  Word 3: MIH message block queue pointer.
  Word 4: IECVRSTS caller's return address.
  Word 5:

  - Byte 0: Footprint indicator flags:
    - X'80' Redrive I/O requests segment is entered.
    - X'40' IOSGEN macro is invoked to mark all generated channels on the current processor for restart.
    - X'20' IOSINTRP macro is invoked to simulate a channel available interruption for the current processor.
    - X'10' There is a second processor.
    - X'08' IOSGEN macro is invoked to mark all generated channels on the second processor for restart.
    - X'04' RPSGNL macro is invoked to shoulder tap the second processor.
    - X'02' Scan MIH message queue segment is entered.
    - X'01' Terminate address space segment is entered.

- The contents of the input SRB.
- The DCCB, if the scan MIH message queue segment has received control.
- The DCCBMSGS, if the scan MIH message queue segment has received control.

Output of the Hot I/O Recovery Module (IECVHREC)

This module's functional recovery procedure (HRECFRR) takes an SDUMP but puts no data in the SDUMP buffer.

HRECFRR puts the following data into the variable area of the SDWA:

At byte 0: A copy of the SCD.
At byte 32: A copy of the FRR work area, which contains the following:

- word 0: first base register
- word 1: second base register
- word 2: SRB pointer
- word 3: work area pointer
- word 4: SCD pointer
- word 5, byte 0: Flags
  - X'01' reserved
  - X'02' reserved
  - X'04' FRR recursion indicator
  - X'08' address of work area is valid
  - X'10' reserved devices found, message IEA421E not yet issued
  - X'20' channel can be enabled
  - X'40' channel was reset, re-reserves not complete
  - X'80' SALLOC lock held
- word 5, bytes 1-3: reserved

Output of the I/O Restart Module (IECVIRST)

The functional recovery procedures, (IRSTFRR), of this module issues an SDUMP macro requesting SQA, the nucleus, and the 4K buffer to be dumped. The work area (storage area retrieved via GETMAIN that holds the compiler's automatic data) is copied to the 4K buffer along with each reserve table segment.

IRSTFRR puts the following data in the variable area of the SDWA:

At byte 0: The 24 byte FRR parameter area returned by the SETFRR macro.

At byte 24: Halfword channel mask. Each bit in the halfword channel mask corresponds to a given channel.

(Bit 1 corresponds to channel 1

●
●
●

Bit 16 corresponds to channel 16.)

If a bit is on, the corresponding channel encountered an error.

IECVIRST loads one or more wait states. For each loaded wait state, a system termination record is written to the SYS1.LOGREC data set. **Note:** this record may not appear in the data set since the system may not be able to perform I/O operations before the wait state is loaded. It appears in the SYS1.LOGREC buffer located in the SQA in storage. The mapping macro, IHALRB, maps the system termination record. The variable area within the system termination record contains:

At byte 0: Current registers (0-15)

At byte 64: The 24 byte FRR parameter area returned by the SETFRR macro.

At byte 84: Halfword channel mask. Each bit in the halfword channel mask corresponds to a given channel.

Bit 1 corresponds to channel 1

●
●
●

Bit 16 corresponds to channel 16.)

If bit is on, the corresponding channel encountered an error.

At byte 86: Halfword channel set id.

Output of the Nonresident Halt-I/O Module (IGC0003C)

The module's functional recovery procedure, HALT0900, writes no SDUMP buffer record. If HALT0900 is the first recovery procedure entered by RTM, it writes the following data in the variable area of the SDWA:

At byte 0: the contents of register 0 and 1 when the module was entered to halt a teleprocessing operation.

At byte 8: the IOQ for the teleprocessing operation.

At byte 14: the UCB (prefix segment included) for the teleprocessing device.

Additionally, HALT0900 directs RTM to put trace data, task-related data areas, and all the IECIHIO code in a user dump (SYSABEND, SYSMDUMP, or SYSUDUMP), if such a dump was requested.

Output of the Nonresident Purge Module (IGC0001F)

The module's functional recovery procedure, PURGEFRR, puts data in the SDUMP buffer, but the module's ESTAE recovery procedure, PRGESTAE, writes the contents of the buffer into the SYS1.DUMP data set. PURGEFRR puts the following information into the SDUMP buffer:

At byte 10: the PPL.

At byte 20: the IPIB.

At byte 50: a work area whose contents include a variable number of saved registers, a list of pages to be fixed, and the list forms of macros used by the module.

At byte 140: if the module holds a UCB lock, the UCB (prefix and common segments only) associated with that lock.

At byte 160: if the module holds an LCH lock, the logical channel queue associated with that lock and all the IOQs on the logical channel queue.

PURGEFRR puts the following data into the variable area of the SDWA:

At byte 0: IGC0001F (the module name).
At byte 8: IGC016 (the module's entry point).
At byte 16: PURGEFRR (the recovery procedure's name and entry point).

PRGESTAE puts the same data in its SDWA, except at byte 16, where it writes its own name.

Output of the Post-Status Module (IECVPST)

The module's functional recovery procedure, PSTFRRTY, puts at byte 6 of the SDUMP buffer record the IOSB that was being processed when the error occurred.

PSTFRRTY puts the following data in the variable area of the SDWA:

At byte 0: the IOSB that was being processed when the error occurred.

At byte 6C: IECVPST (the module name).

At byte 73: X'04', a code indicating that PSTFRRTY asked RTM to return control instead of continuing termination processing.

At byte 74: the address of the IOSB.

At byte 78: the address of the FRR work area.

At byte 7C: the contents of the base register.

Output of the Redrive I/O Service Routine (IECVRDIO)

The module's functional recovery procedure, RDIOFRR, puts at byte 6 of the SDUMP buffer record the general work area used for automatic data.

The following is placed in the variable area of the SDWA:

At byte 0: a copy of the 24-byte FRR work area pointed to by SDWAPARM.

Output of the Re-Reserve Service Routine (IECVRRSV)

The module's functional recovery procedure, RRSVFRR, writes no SDUMP buffer record. The following is placed in the variable area of the SDWA:

At byte 0: a copy of the 24-byte FRR work area pointed to by SDWAPARM.

Output of the Resident Halt-I/O Module (IECIHIO)

The module's functional recovery procedure, HIOFRR, puts at byte 6 of the SDUMP buffer record the UCB (prefix segment included) for the device on which a channel program was to be halted. Following the SDUMP buffer record in the dump are other 4K records written by HIOFRR. These contain all the IECIHIO code, the PSA or prefixed save area (the first 4K bytes of low storage), and the system's trace tables.

HIOFRR puts the following data in the variable area of the SDWA:

At byte 0: X'0C', if the error occurred in the shoulder-tapped processor; otherwise, the code that was returned when HIOFRR issued an SDUMP macro to write in the SYS1.DUMP data set. (X'FF' means nothing was written to the SYS1.DUMP data set.)

At byte 1: the UCB (prefix included) for the device on which a channel program halted.

Output of the Special SIO Module (IECVESIO)

This module's functional recovery procedure, ESIOFRR, does not use the SDUMP buffer. However, the following is placed in the variable area of the SDWA.

At byte 0: The 24-byte FRR parameters.

Output of the Storage Manager Module (IECVSMGR)

This module's functional recovery procedure, IECVSMFR, puts at byte 6 of the SDUMP buffer record the pool headers for the "small block," "medium block," and "large block" pools.

Following the SDUMP buffer record in the dump are other 4K records written by IECVSMFR. These contain the SQA (system queue area), the system's trace tables, and the code in the IECVSMGR module.

The output of the system's queue verification routine is in the variable area of the SDWA. IECVSMFR passes the variable area to that routine for use as a QVOD (queue verification output data area). The free queue for small, medium, and large blocks is moved to SDWA.

Output of the Unconditional Reserve Detection Module (IECVURDT)

This module's functional recovery procedure does not use the SDUMP buffer. However, the following is placed in the variable area of the SDWA.

At byte 0: The 24-byte FRR parameters.

Output of the Unconditional Reserve Service Module (IECVURSV)

This module's functional recovery procedure does not use the SDUMP buffer. However, the following is placed in the variable area of the SDWA.

At byte 0: The 24-byte FRR parameters.

Output of the Resume I/O Service Routine (IOSVRSUM)

The module's functional recovery procedure, RSUMFRR, places the following in the VRADATA field of the SDWA:

- The IOSB that initiated the request. (If there is not an IOSB, "IOSB ZERO" is put in VRADATA.)
- The UCB associated with the request. (If there is not a UCB, "UCB ZERO" is put in VRADATA.)

Informative IOSB Fields

An examination of three IOSB fields, IOSDRVID, IOSPROC, and IOSCOD, answers these questions:

1. Did IOS create the IOSB, or did one of its drivers create it? If one of the drivers, which one?
2. If IOS created the IOSB, why did it?
3. If a driver created the IOSB, what does the IOSB show about the status of the I/O request it represents?

The IOSDRVID field answers (1), the IOSPROC field answers (2), and the IOSCOD field answers (3).

The IOSDRVID Field

IOSDRVID is a one-byte field at an offset of four bytes into the IOSB. The possible contents and their meanings are:

Contents	Meaning
X'00'	IOS created the IOSB.
X'01'	The driver wants to be anonymous to IOS because it doesn't want to take part in certain kinds of I/O processing. (For example, the driver might not want to be called to dispose of the IOSB during a purge operation.)
X'02'	EXCP is the driver.
X'03'	ABP (VSAM) is the driver.
X'04'	VTAM is the driver.
X'05'	TCAM is the driver.
X'06'	OLTEP is the driver.
X'07'	Program fetch is the driver.
X'08'	JES3 is the driver.
X'09'	MSC is the driver.
X'0A'	IECVIOPM is the driver.
X'0B'	VPSS is the driver.

X'0C'	CRYPTO is the driver.
X'0E'	ASM is the driver.

The IOSPROC Field

IOSPROC is a one-byte field at an offset of three bytes into the IOSB. The field is used as an index to a branch table in the post status module (IECVPST). The possible contents and what they tell about the IOSB are:

Contents	What They Tell about the IOSB
X'00'	Indicates "normal" non-IOS generated IOSB.
X'04'	The IOSB was created by EDIEINT1 when a PCI interruption occurred without other status information. It was marked X'04' to direct PSTIOSB to enter the driver's PCI appendage.
X'08'	The IOSB was created by EATTENT1 when tests of the CSW, UCB, and attention table indicated that control should be routed to an attention routine. The IOSB was marked X'08' to direct PSTIOSB to branch to the attention routine for the device.
X'0C'	The IOSB was created by LCHPURG to replace a purged IOSB for an I/O operation that must be completed - a sense, reserve, or release operation. After the I/O operation is completed, PSTIOSB sees the the X'0C' and enters FREEBLK, which frees the IOSB.
X'10'	The IOSB was created by EDEVEND1 when called by IECVXDAS because a direct-access device was readied. It was marked X'10' to direct PSTIOSB to enter IECVDAVV via the exit effectors and ERP loader.
X'14'	The IOSB was created by EPOSTIO1 when it determined that a message must be sent to the operator about the availability of the device. The IOSB was marked X'14' to direct PSTIOSB to enter, via the exit effectors and ERP loader, the ERP message writer (IGE0025C).
X'20'	The IOSB was created by EDETECT1 when it determined that unconditional reserve recovery was needed. It was marked X'20' to direct PSTIOSB to enter IECVURDT and IECVDURP.

The IOSCOD Field

IOSCOD is a one-byte field at an offset of five bytes into the IOSB. The possible contents, with explanations of what they mean are:

Contents	Explanation
X'41'	An ERP, the ABE appendage, or the CHE appendage detected an I/O error and determined that is was uncorrectable. (An ERP does not put X'41' in IOSCOD. IGC015 does it if, on receiving control from the ERP, it finds the "exceptional-condition" bit (IOSEX) on, the "retry" bit (IOSERR) off, and X'7F' in IOSCOD.)
X'42'	The EOE appendage detected an extent error and directed the driver to put X'42' in IOSCOD.
X'43'	A paging I/O operation couldn't be started immediately, and the IOSB specified that I/O-request processing be terminated in such a case. ETCH1 compiled by putting X'43' in IOSCOD and scheduling IECVPST. (ABP, on finding X'43', submits a new I/O request to read a duplicate page on another device.)
X'44'	ETCH1 stopped processing the I/O request because its SRB and IOSB were needed to process a hardware error on the device allocated to the request. (ETCH1 does not put X'44' in IOSCOD. IGC015 does it if, on receiving control from the ERP, it finds the "exceptional-condition" bit (IOSEX) on, the "retry" bit (IOSERR) off, and X'7E' in IOSCOD.)

X‘45’ I/O-request processing was terminated abnormally. Reasons for the termination are:

- The IECIOSCN or IECVPST module took a program check.
- The operator pressed the RESTART key while an I/O request was being processed.
- A program check occurred while a nonresident ERP or ERP service module was in control.
- A program check occurred in the NRM/ABM exit processing of module, IECVEXCP.

IECFRR, PSTFRRTY, or the ERP loader's ESTAE procedure (ERPLESTA) was entered by RTM.

X‘48’ The I/O request was purged. The driver's purge procedure put X‘48’ in IOSCOD.

X‘4B’ An I/O error occurred when the tape ERP requested that a volume be repositioned. The ERP put X‘4B’ in IOSCOD.

X‘4C’ In asking for an I/O operation on a specific 2305 exposure, the driver specified an invalid exposure number in the IOSB. IECVXDRS puts X‘4C’ in IOSCOD and scheduled IECVPST.

X‘4D’ The driver guaranteed the availability of a path to the device, but when the start-I/O instruction was issued, the condition code was set to 3 (channel or device not operational). EPOSTIO1 put X‘4D’ in IOSCOD and scheduled IECVPST.

X‘4E’ One of the following occurred:

- The driver guaranteed the availability of a path to the device, but the device was reserved to another path.
- The driver asked IOS to release a device, and in trying, IOS found that at least one other user of the device wanted it to be reserved.

A device dependent SIO module (IECVXDAS, IECVXDRS, IECVXSKS, or IECVXVRS), put X‘4E’ in IOSCOD and scheduled IECVPST.

X‘4F’ The driver guaranteed the availability of a path to the device, but ETCH1 found that the channel set on that path was not configured. ETCH1 puts X‘4F’ in IOSCOD and schedules IECVPST.

X‘51’ ETCH1 determined that the device has been boxed and placed offline, and scheduled IECVPST. (ETCH1 does not put X‘51’ in IOSCOD if the driver is EXCP. The ERP is eventually given control, and if it enters IGC015 with the “exceptional-condition” bit (IOSEX) on, the “retry” bit (IOSERR) off, and X‘74’ in IOSCOD, IGC015 overlays X‘74’ with X‘51’.)

X‘71’ Set by the direct-access ERP when the sense bytes show a data check and the IOSDRVID field shows that the driver is program fetch.

X‘74’ Set by ETCH1 when it determined that a device was boxed and placed offline, and the request was from EXCP. (X‘74’ may be altered by IGC015. See the explanation for X‘51’.)

X‘7E’ Set by ETCH1 when it determined that the SRB and IOSB for the request were needed to process a hardware error on the device allocated to the request. (X‘7E’ may be altered by IGC015. See the explanation for X‘44’.)

X‘7F’ Set the IECHNSCH before the I/O operation was started. If the IOSB has been returned to the driver's termination procedure, X‘7F’ signifies that the I/O operation completed successfully.

Table of IOS Messages

The table below gives the numbers of IOS and ERP messages, identifies the IOS modules that detect a need for the message, and indicates the non-IOS module that issues it.

Message	Issued by	Module Detecting Need for Message
IEA000A	IGE0025C*	EPOSTIO1 DAVERR or an ERP**
IEA000I	IGE0025C	an ERP
IEA001I	IGE0025C	EPOSTIO1
IEA003I	IEAVTRET	CLEARDEV***
IEA004I	IEAVTRET	ACRPROC***
IEA004I	IEAVTRET	UCBACT***
IEA004I	IEAVTRET	LOSTCHAN***
IEA004I	IEAVTRET	IECVRDIO
IEA018A	IEEVLDWT	IECVRSTS
IEA019A	IEEVLDWT	IECVFCHN
IEA026I	IEAVTRET	IECVRRSV
IEA066A	IEEVLDWT	IECVHREC
IEA067A	IEEVLDWT	IECVHREC
IEA068A	IEEVLDWT	IECVHREC
IEA069A	IEEVLDWT	IECVHREC
IEA070A	IEEVLDWT	IECVHREC
IEA071E	IEEVLDWT	IECVHREC
IEA072I	IEEVLDWT	IECVHREC
IEA073A	IEEVLDWT	IECVPST
IEA151W	IGFPTERM	IECVRRSV
IEA151W	IEEVLDWT	IECVRST1
IEA151W	IEEVLDWT	IECVHREC
IEA151W	IEEVLDWT	IECVFCHN
IEA410E	IEAVTRET	LOSTCHAN***
IEA410E	IEAVTRET	IECVIRST
IEA421E	IEAVTRET	IECVIRST
IEA421E	IEEVLDWT	IECVFCHN
IEA427A	IECVDURP	IECVDURP
IEA428I	IECVDURP	IECVDURP
IEA429I	IECVDURP	IECVDURP
IEA438A	IEEVLDWT	IECVIRST
IEA439D	IEEVLDWT	IECVIRST
IEA440A	IEEVLDWT	IECVRSTI
IEA442E	IECLMSGD	an ERP
IEA443I	IECVIOSI	IECVIOSI
IEA444I	IECVDPTH	IECVDPTH
IEA446D	IEEVLDWT	IECVIRST
IEA604A	IECVDAVV	DAVINT
IEA605A	IECVDAVV	DAVINT
IEA606I	IECVDAVV	DAVINT
IEA919I	IEAVTRET	IECVCINT
IEA970I	IEAVTRET	IECVCRHA
IEA971I	IEAVTRET	IECVCRHA
IEA972I	IEAVTRET	IECVCRHD

*This is the ERP message writer.

**IEA000A is issued only if an ERP module finds the "intervention-required" bit on in the sense bytes.

***The message is formatted by another IOS procedure, RECORDIT. It gives control to IEAVTRER, which schedules IEAVTRET to write the message asynchronously.

IOS Wait State Codes

The following table lists wait state codes with the modules that issue them. For the meanings of the codes, refer to *System Codes.*

Code	Issuing Module
X'022'	DAVESTA (ESTAE Recovery)
X'02F'	PSTWAIT
X'041'	ACRPROC (ACR Call Procedure)
X'04C'	IECVIRST
X'04D'	IECVIRST
X'04E'	IECVIRST, IECVHREC, IECVRRSV, IECVRSTI, IECVFCHN
X'066'	IECVHREC
X'067'	IECVHREC
X'068'	IECVHREC
X'069'	IECVHREC
X'06A'	IECVHREC
X'06B'	IECVCINT
X'06D'	IECVRSTS
X'06E'	IECVFCHN
X'06F'	IECVDURP
X'0E6'	IECVFCHN

Table of IOS Return Codes

The following table matches a return code with the names of IOS modules that exit with the code in register 15.

Return		Return		Return	
Code	Module Name	Code	Module Name	Code	Module Name
X'00'	HIOCCH	X'08'	HIOCCH	X'24'	IECVESIO
	IECIHIO		HIOLOP		
	IGC0001G		IECIHIO	X'28'	IECVESIO
	IGC0003C		IGC0003C		
	IGC016		IGC016	X'2C'	IECVESIO
	SMGFREVR		SMGFREVR		
	TCCWR000		TCCWR000	X'30'	IECVESIO
	IECVCINT		IECVCINT		
	IECCONCS		IECVDPTH	X'34'	IECVESIO
	IECVDPTH		IECVESIO		
	IECVESIO		IECHFCHN	X'3C'	IECVESIO
	IECVFCHN		IECVIOSI		IECVDPTH
	IECVIOSI				
	IECVRRSV	X'0C'	IGC0003C		
	IECVURSV		TCCWM000		
	IOSVLEVL		TCCWM100		
	IOSVSUCB		TCCWM300		
			TCCWM400		
X'04'	HIOCCH		IECVDPTH		
	IGC0001G		IECVESIO		
	IGC0003C		IECVFCHN		
	IGC016				
	SMGFREVR	X'10'	IGC0003C		
	TCCWR000		IECVDPTH		
	TCCWU000		IECVESIO		
	TCCWX000				
	IECVCINT	X'14'	IGC0003C		
	IECCONCS		IGC016		
	IECVDPTH		IECVESIO		
	IECVESIO				
	IECVFCHN	X'18'	IGC0003C		
	IECVRRSV		IECVESIO		
	IECVURSV				
	IOSVLEVL	X'20'	IGC0003C		
	IOSVSUCB		IECVESIO		

Error Recovery Procedures (ERPs)

This topic describes ERP routines and helps you determine the module responsible for problem symptoms.

IOS and ERP Processing

Error recovery procedures (ERPs) are scheduled by the IOS post status routine (IECVPST). When IOS receives an interrupt with a unit check, unit exception, incorrect length, program check, chaining check, channel data check, channel control check, or interface control check, the IOSEX bit (in the IOSB) is turned on. If the interrupt shows a unit check, the sense information is read into the ERP work area (EWA).

IOS Sequence

The sequence of IOS post status events is:

1. Inspect the IOSERR (in the IOSB) to determine if error recovery is already in progress, if it is, step 2 is bypassed.

2. Turn on IOSEX (in the IOSB) and issue a BALR to the abnormal end appendage.

3. Upon return from the appendage, or if ERP is already in progress:

   a. For DASD error recovery - issue a BALR to IECVDERP.
   b. For nonDASD error recovery - branch to IEA0EF00 (the exit effector).

4. Upon return from the ERP, IOS takes action to perform the ERP requirements listed in step 3 of the following topic "ERP Sequence."

ERP Sequence

The sequence of ERP events is:

1. The required ERP inspects the status and sense information using an ERP error interpreter table and an IOS routine referred to as the IOS error interpreter (IECVITRP).

2. The IOS error interpreter routine makes a branch vector return to a label within the ERP for a given error.

3. For a given error, the ERP determines the requirement for and initiates one or more of the following actions:

   a. Retry/restart the channel program
   b. Issue console message IEA000I or IEA000A
   c. Log the error
   d. Indicate that the error is permanent
   e. Indicate that the error has been recovered

Identifying ERP Module Names

The name of an ERP routine (for nonDASD devices) must be of the form IGE0xxxx, where xxxx is a positive decimal number. The decimal number xxxx corresponds to the one-byte binary value in the UCBETI. When an error routine is needed, this byte is converted to decimal, unpacked, and substituted for the value xxxx to complete the name. For example, the ERP routine for the IBM 2540 is IGE0001C. The UCBETI for a 2540 contains X'0D'. When this byte is converted to decimal, it becomes a plus 013. When the plus 013 is unpacked, it becomes F0F1C3, which is printed as 01C to complete the name IGE0001C.

How ERP Transfers Control

ERP routines frequently transfer control. They may give control to another load module of the same ERP, to the outboard recorder (OBR), to an IOS statistics update routine (IGE0025D), or to the IOS write-to-operator routine (IGE0025C). The CVT + X'2C' points to the transfer control routine that uses the contents of register 13 to determine which module should receive control. The technique used is the same as described in the previous topic "Identifying ERP Module Names." The contents of register 13 are converted to decimal, unpacked, and placed in the low-order halfword of IGE0xxxx. The following table shows a few examples:

Contents of Register 13	Control Given to Module
00000009	IGE0000I
0000000E	IGE0001D
00000017	IGE0002C
000000FE	IGE0025D
000007D6	IGE0200F

Abnormal End Appendages

Abnormal end appendages are of critical importance to ERP. Within IOS, the BALR issued to the appendages is located immediately before a return vector table. Two important facts are:

- The ability to modify the necessary control blocks allows the appendage to turn off error indicators, or to perform error recovery actions, without ERP being invoked.
- Because IOS gives control to the appendage via a BALR instruction immediately before a return vector table, the appendage can branch back to IOS as follows:
  - Return register + 0 - IOBFLAG1 in the IOB is examined for the IOBERRTN and IOBIOERR bits and the following actions are taken:

IOBERRTN	IOBIOERR	Action
0	0	The user channel program is posted complete.
1	1	The ERP is scheduled.
1	0	Should not occur during error recovery.
0	1	1. If this is the first time the appendage was entered (IOBECB = X'7F'), schedule error recovery if allowed by the DCBIFLG field in the DCB. If error recovery is not allowed, handle as permanent error. 2. If this is the second time the appendage was entered (IOBECB not equal to X'7F'), handle as a permanent error.

  - Return register + 4 - channel program not posted complete.

- Return register + 8 - the request is retried.
- Return register + C - DDR processing required.

The abnormal end appendage should be examined when analyzing possible error recovery problems.

Retry/Restart the Channel Program

For retry, errors are retried from the first CCW. For restart, errors are restarted from the failing CCW. An ERP's decision to retry or to restart a channel program is primarily dependent upon the type of chaining, and secondarily dependent upon the type of error. For channel programs using data chaining or no chaining, the request should be retried beginning with the first CCW. On a request using command chaining, the restart is done from the failing CCW. The IOSB address of the real channel program (IOSRST) is updated with the real address of the CCW at which restart is to begin.

After a retry or restart has been successful, the ending status is presented to IOS, but the ERP is given control again because the IOSERR bit is still on (indicating that error recovery is in control). ERP performs error inspection using the error interpreter table to assure a cleanup of the IOSB. The IOSERR and IOSEX bits (in the IOSB) are turned off, the error count fields are cleared, the abnormal status bits in the CSW are turned off, and return is made to IOS.

Error Interpreter

An ERP module usually contains subroutines to handle various errors for the device types for which the module is responsible. In order to test the two CSW bytes and the first two sense bytes, ERP uses a common IOS routine (IECVITRP) pointed to by CVT+X'44'. This routine uses the ERP module's error interpreter table to determine which subroutine within the calling ERP module should be branched to for handling the error condition detected by IOS. The error interpreter table establishes the priority and sequence in which errors are handled. Each entry in the table represents an error condition, and in the entry there is a label for the subroutine to be branched to when the error condition is detected by the IOS routine. The label names vary from module to module, but the technique used is consistent throughout ERP.

Example of an Error Interpreter Table

The following table shows an example of an error interpreter table.

```
DC X'1D',AL1(CCC-*+1)      Channel control check
DC X'1E',AL1(ICC-*+1)      Interface control check
DC X'08',AL1(PERM-*+1)     Permanent error
DC X'03',AL1(EQUP-*+1)     Equipment check
DC X'2F',AL1(ENDI-*+1)     End of test
```

In this example, if ERP is entered with a status of channel control check, the entry shows that a branch is taken to the label CCC. If ERP is entered with sense bytes indicating only a permanent error, then a branch is taken to label PERM. The table shows that the IOS routine checks for four error conditions, and if none of the conditions is satisfied, then a branch is taken to ENDI.

The IOS routine tests the table from the top, and the first error condition detected results in a branch to the label within the entry. Because ERP handles only one error condition at a time, if two or more error conditions are indicated, only a branch to the first label is taken. For example, if both the interface control check and equipment check are indicated, then a branch is taken only to the label ICC.

Note that the UCB, IOSB, and EWA are required for ERP to inspect the status and sense information.

ERP Messages and Logging

ERP causes an error message to appear on the console or an OBR or MDR record to be written to SYS1.LOGREC based on the IOSMSG and IOSLOG bits in the IOSB. The following table shows the action taken for the possible settings of the bits.

Bit Setting		
IOSMSG	**IOSLOG**	**Action**
0	0	1. No message, no SYS1.LOGREC entry
1	0	2. Console message, no SYS1.LOGREC entry
0	1	3. No message, SYS1.LOGREC entry
1	1	4. Console message, SYS1.LOGREC entry

- Action 1: Certain permanent errors do not require logging or error messages, such as no record found.

- Action 2: Certain errors require only an error message, such as intervention required.

- Action 3: Certain errors are logged but messages are not issued. For example, if a DASD equipment check is recovered, the error is logged, but a message is not issued because the recovery was successful.

- Action 4: Some errors are logged and an error message issued to the operator. For example, if a DASD equipment check is not recovered, an OBR type record is logged and message IEA000I is issued.

ERP transfers control to IOS module IGE0025C for messages and logging. If logging is required, IGE0025C transfers control to the outboard recorder (OBR).

Intercept Conditions

The conditions that cause entry to an ERP occur at SIO time (indicated by a condition code of one), or at channel end time. However, if an error condition occurs at device end time after channel end has been handled, there is no IOSB because it was freed with channel end posting. In this case, IOS saves the two CSW status bytes and complete sense information, and sets the intercept flag (UCBITF) in the UCB. When the next request for this device is processed, IOS detects the UCBITF flag and moves the CSW and sense data to the new IOSB. IOS sets X‘7E’ in the completion code field of the IOSB and passes control to ERP via the abnormal end appendage route.

Some intercept conditions are recoverable (such as the 1403 device end with the channel 9 or 12 sense, or device end with intervention required sense for any device). If the intercept condition is recoverable, ERP changes the code X‘7E’ in the completion code field of the IOSB to X'7F'. However, most intercept conditions cannot be recovered. In this case, ERP marks the IOSB in error, turns off the ERP in-control bit in the IOSB, and returns to IOS which changes the X'7E' to X'44'.

Unit Check on Sense Command

IOS handles a unit check on the sense command as an equipment check. To do this, IOS simulates an equipment check by setting the equipment check in sense byte zero of the IOSB, and X'FE' in sense byte one. The ERP can then distinguish a unit check on a sense command from an ordinary equipment check.

Compound Errors

A compound error is one that occurs when a previous error has been successfully retried. In most cases, this condition is handled by normal flow through the ERP. However, if the compound error is either a unit exception or wrong length indication in the CSW, special processing must take place to guarantee that the channel end appendage is entered before the ERP tests the condition. This processing is needed because IOS does not enter the channel end appendage if ERP is in control. Therefore, on conditions of unit exception and wrong length indication, ERP turns off the exception (IOSEX) and error (IOSERR) bits in the IOSB and returns to IOS. IOS then enters the channel end appendage, and later, the ERP is scheduled because the CSW in the IOSB still contains the error status.

Diagnostic Approach

The ERP diagnostic approach has two major objectives:

1. To determine the recovery action that is being performed by the ERP module.
2. To determine what recovery action should be performed, according to the manuals that provide the component description for the devices.

The previous topics have explained how to perform the first objective. This topic explains the use of IBM documentation to perform the second objective.

If the manuals mentioned in this topic do not show that error recovery action is required, or if the referenced "priority" figures do not show the priority in which a given error is to be handled, the problem may not be suitable for an APAR. The component description manuals show the required/suggested error recovery actions, and these actions are the specifications for the ERP software. If there are no specifications for a given error condition, or if the specifications seem incorrect, then the hardware CE should assume responsibility for any necessary changes.

For brevity, only those manuals for the devices that require the greatest PSR, field support, and APAR activity are shown.

DASD ERP

The error recovery actions for DASD are documented in the following manuals in the topic "Error Condition Table" or "Recovery Action Table."

Order Number	IBM Device Type
GA26-3599 -	2314 Direct Access Storage Facility
GA26-1589 -	2305 Fixed Head Storage and 2835 Storage Control
GA26-1615 -	3330 Disk Storage
GA26-1619 -	3340 Direct Access Storage Facility and 3344 Direct Access Storage
GA26-1638 -	3350 Direct Access Storage

The manuals indicate a required action number for each valid combination of error status and sense information, whether the error condition should be logged, and the action to be taken for each action number. The description includes the CCWs that must be prefixed to the channel program being retried or restarted.

DASD ERP is totally contained in the ERP module IECVDERP. A BALR is issued by IECVPST to IECVDERP for DASD ERP. If console messages or logging are required, control is given to IGE0025C via IOS.

Tape ERP

The error recovery actions for tape devices are documented in the following manuals in the topic "Error Recovery Procedures."

Order Number	IBM Device Type
GA32-0020 -	3803 Tape Control Model 1 and 3420 Magnetic Tape Unit Models 3, 5, and 7.
GA32-0021 -	3803 Tape Control Model 2 and 3420 Magnetic Tape Unit Models 4, 6, and 8.
GA26-1647 -	3803 Tape Control Model 3 and 3420 Magnetic Tape Unit Models 3 and 5.
GA32-0022 -	3410 Magnetic Tape Unit and 3411 Magnetic Tape Unit and Tape Control.

The manuals indicate a required action number for each valid combination of error status and sense information, and the action to be performed for each action number. The action description includes a verbal description of CCWs to be used for error recovery, such as "Set the correct mode (if seven track) and reposition the tape." Also, the priority assignment given to the valid combinations of error status and sense information is shown in the manuals in the topic "Status and Sense Indicator (Bits) Checking Sequence."

Printer ERP

For the IBM 3211 Printer, the manual GA24-3543 describes the error recovery actions to be performed in the topic "Suggested Error Recovery Procedures." The priority assignment for handling valid error status and sense combinations is shown in the figure "Error Recovery Priority Sequence."

For the IBM 3800 Printing Subsystem, the manual GA26-1635 describes error recovery actions in the chapter "Error Detection, Recovery, and Recording." The figure "3800 Error Conditions and Suggested Recovery Actions" describes various status and sense error indicators, the possible cause of the error conditions, and what the ERP should do to recover the error. Within the same chapter is a topic for permanent errors with the required content of operator messages, and a topic for error logging that specifies the types of SYS1.LOGREC entries (CCH, SDR, OBR, and MDR). Also, the figure "3800 Error Recording Actions" specifies which error conditions require SYS1.LOGREC recording and the type of SYS1.LOGREC entry to be created for specific errors.

ERP Traps

The previous topic "Error Interpreter" explains how to determine the assigned label for various error routines within an ERP module. These labels are excellent places from which to branch to the module trap area for traps. However, extra care should be used because the module location to which an error interpreter table causes a branch may be used by more than one entry. It is possible that more than one table entry can contain the same name, and that different label names can reside at the same module displacement if defined as DS. It is important, therefore, that code placed in the patch area test the reason for receiving control. If the patch area verifies the expected reason for receiving control (by testing the status and sense information), then a program check or one-instruction loop is an excellent trap/documentation technique.

MVS systems run with ERP enabled, in supervisor state, and under the RCT TCB. DASD ERP runs in SRB mode. Checking the ERP program is effective when an 0C3 abend is caused (use an EXECUTE instruction to execute itself), and a SLIP command is used to catch the abend before the system FRR has freed the IOSB and EWA control blocks.

An abend (such as a program check) in an ERP module causes the address space to remain unusable until a re-IPL is performed.

When possible, a one-instruction loop in the ERP patch area should be followed by a stand-alone dump for documentation of the ERP's action and failure.

A GTF trace or CCW trace is usually required to debug ERP problems. When possible, use a full trace; but for intermittent problems, it may be useful to modify the interrupt handler so that the trace occurs only for the desired I/O or selected SVC operations.

Diagnostic Approach Summary

In summary, debugging ERP problems consists of:

1. Determining which ERP module and subroutine receives control for error recovery (if the abnormal end appendage and DCBIFLG field permit ERP to execute).

2. Inspecting the routine to determine if it conforms to the specifications for error recovery provided in the component description of the device.

3. Assigning responsibility for the problem to the CE if ERP operates according to the specifications, or document the problem using GTF, dumps, and traps within ERP to assure adequate APAR documentation.

Program Manager

The program manager controls the various means by which programs are located, brought into storage, and subsequently given control for execution. This chapter describes the program manager and includes the following topics:

- Functional Description
- Basic Functional Flow
- 806 ABEND
- APF Authorization
- Module Subpools
- Fetch/Program Manager Work Area (FETWK)
- RB Extended Save Area (RBEXSAVE)
- CDE Pool Control

Functional Description

The program manager's primary functions are to create and maintain control block queues necessary to fetch load modules into virtual storage and delete load modules from virtual storage, and to transfer control between load modules during program execution.

Load modules fetched into virtual storage reside in one of the link pack areas (fixed, modified, or pageable) or in a job step's job pack area. External communication to the program manager during program execution is accomplished by means of the system macros: LINK, XCTL, LOAD, DELETE, SYNCH, and IDENTIFY.

Program Manager Organization

The program manager consists of six modules, IEAVLK00, IEAVLK01, IEAVLK02, IEAVLK03, IEAVID00, and IEAVNP05. Their major functions are described Figure 5-6.

Program Manager Control Blocks

The major program manager control blocks and work areas are the contents directory entry (CDE), the link pack directory entry (LPDE), the load list element (LLE), the fetch work area (FETWK), the extent list (XL), the program request block (PRB), and the DE (directory entry) save area. These control blocks and work areas are described in Figure 5-7.

Program Manager Queues

The job pack queue (JPQ), active link pack area queue (ALPAQ), the load list (LL), and the SVRB suspend queue (SSQ) are the four basic queues maintained by the program manager. These queues which are summarized in Figure 5-8, are described below:

JPQ - Each job step has a job pack queue. The queue consists of CDEs (built in subpool 255) representing modules (or minor entry points of modules) explicitly requested by one or more tasks in the step via the LINK, LOAD, XCTL, or IDENTIFY macros, but not available in one of the link pack areas. This queue is empty at the initiation of the job step.

ALPAQ - The active link pack area is a system queue consisting of CDEs (built in subpool 245) representing modules (and minor entry points of modules) that are in the:

- Modified link pack area
- Fixed link pack area
- Pageable link pack area and listed in the IEALOD00 member of SYS1.PARMLIB
- Pageable link pack area, with no CDE already on the queue, and currently in use

This queue initially consists of CDEs representing modules belonging to the first three categories listed above.

LL - The load list is a task-oriented queue consisting of LLEs representing load modules the task has accessed via the LOAD macro. Each LLE points to the CDE on the JPQ or the ALPAQ representing the module LOADed. The load list for each task is initially empty.

SSQ - The SVRB suspend queue is a CDE-headed queue consisting of program manager SVRBs representing requests within the job step for that particular module. The request could not be immediately satisfied either because the module is currently being fetched into virtual storage by a previous request or because the module is serially reusable and currently in use.

PLPAD - The pageable link pack area directory is a table of LPDEs built at system initialization time. Each LPDE in the table represents a module (or an alias entry point of a module) in the pageable link pack area. Once built, the table is static and read-only.

Module Name	CSECT Name	Residence	Major External Entry Points	Primary Function
IEAVLK00	IEAVLK00	Nucleus	IGC006 IGC007 IGC008 IGC009 IGC012 IEAQCS01 IEAQCDSR IEAVVMSR	Contains the entry points for LINK (IGC006, XCTL (IGC007), LOAD (IGC008), DELETE (IGC009), and SYNCH (IGC012). Along with module IEAVLK01, it services each of these functions.
IEAVLK01	IEAVLK01	Nucleus	None	Along with module IEAVLK00, it services the LINK, XCTL, and LOAD function.
IEAVLK02	IEAVLK02	Nucleus	IEAPPGMA IEAPPGMX	Cleans up resources in case of an ABEND (IEAPPGMA) and whenever a PRB exits (IEAPPGMX).
IEAVLK03	IEAVLK03	Nucleus	FRRPGMMG FRRPGMX	Program Manager Functional Recovery Routines.
IEAVID00	IGC041	Pageable LPA	IGC041	Service routine and FRR for IDENTIFY.
IEAVNP05	IEAVNP05	Exists only during system initialization	IEAVNP05	Creates the modified, fixed, and pageable link pack areas. Creates the initial active link pack area queue. Creates the pageable link pack area directory.

Figure 5-6. Program Manager Modules

Area	Mapping Macro	Size (Bytes)	Subpool Number	Usage	Created by	Deleted by
CDE	IHACDE	32	245 or 255	A CDE represents a copy of a module in virtual storage (called a major CDE). Minor CDEs are also used to represent minor entry points.	Program Manager (LINK,XCTL, LOAD, or IDENTIFY)	Program Manager
CDE Pool Control Area	None	16	245	Controls access to pools of CDEs used for active LPA modules.	System Initialization	Not deleted
LPDE	IHALPDE	40	252	Similar to CDE, except are used for the load modules in the pageable link pack area.	System Initialization	Not deleted
LLE	IHALLE	12	255	An LLE is used to control LOAD and DELETE references to a module.	Program Manager (LOAD)	Program Manager
FETWK	IHAFETWK	1540	253	Used as a work area and communication area by FETCH and program manager.	Program Manager when a FETCH is necessary	Program Manager
XL	IHAXTLST	16	255	Contains the load point and size of a load module fetched into virtual storage.	FETCH, or Program Manager, or OS/LOADER	Program Manager
PRB	IHARB	136	253	Controls the execution of a load module.	Program Manager	Exit Processing
DE Save Area	None	64	255	Used to save the user-supplied BLDL entry while program manager is processing.	Program Manager or ATTACH Processor	Program Manager

Figure 5-7. Program Manager Control Blocks and Work Areas

Queue	Serialization	Type of Queue	Elements	Queue Header	When Element is Enqueued	When Element is Dequeued
JPQ	local lock	push down Note 1	CDE	Field TCBJPQ in the job step TCB	When a module is fetched into virtual storage or an IDENTIFY is used to define an embedded entry point	When the module is no longer needed - that is, its use count goes to zero.
ALPAQ	general cross memory services lock (CMS)	push down Note 1	CDE	field IEAQLPAQ pointed to by CVTQLPAQ	At system initialization and thereafter whenever a pageable LPA module is activated	When an activated pageable LPA module is no longer needed. CDEs enqueued during system initialization are never dequeued.
LL	local lock	push down	LLE	field TCBLLS	Whenever a module, not previously loaded by the task, is loaded	Whenever the load count goes to zero or when the task terminates.
SSQ	local lock	in order of TCB priority	SVRB Note 2	field CDRRBP	Whenever a request within the job step cannot be satisfied because a module is being fetched or it is reusable, but in use	When the module becomes available.

Notes:

1. *Queues is push down except when minor CDEs are enqueued. Minor CDEs are always queued following the associated major CDE.*

2. *If the suspend queue is for a serially-reusable module that is in use, the PRB using the module will be queued between the CDE and the first SVRB.*

Figure 5-8. Program Manager Queues

Queue Validation

JPQ and ALPAQ

The program manager functional recovery routine (module IEAVLK03) calls the queue verifier (module IEAVEQV0) to verify and repair CDE elements on the JPQ and on the ALPAQ. Queue verifier validity checks the parameter list, then verifies and corrects the queue structure, removing elements with bad data. Errors encountered are recorded as 16-byte entries in the queue verification output data (QVOD) area. For program management, the QVOD is mapped in the SDWA variable recording area starting at SDWAVRA+X'E'. On exit to the caller, register 15 contains return information as follows:

- Byte 0, bit 0 = 0 indicates that any error encountered has been recorded in the QVOD area.
- Byte 0, bit 0 = 1 indicates that there were more errors than could be recorded in the QVOD area. Errors were detected but not recorded.
- Byte 1 = count of errors recorded.
- Byte 2 = count of errors detected.
- Byte 3 contains one of the following return codes:

  0 - No errors detected

  4 - Elements with bad data were removed from the queue. Their addresses were recorded in the QVOD area.

  8 - Damage to the queue structure - the queue is operational but an undetermined number of elements may have been lost.

  24 - Invalid input parameters - the queue verifier has not performed its function.

Load List

The load list is validated by the program manager FRR itself, beginning at the queue's header in the TCB (TCBLLS). When an invalid LLE is encountered the queue is truncated.

System Initialization

During system initialization the program manager RIM (resource initialization module), IEAVNP05, is called to create the modified LPA, the fixed LPA, the pageable LPA, the initial ALPAQ, and the PLPAD. IEAVNP05 fetches modules into the link pack areas from SYS1.LINKLIB, SYS1.LPALIB, SYS1.SVCLIB, and user libraries. It is driven by SYS1.PARMLIB members IEAFIXxx, IEALPAxx, IEALOD00, IEAPAK00, LNKLSTxx, and by the CLPA system parameter. (See Figure 5-9.)

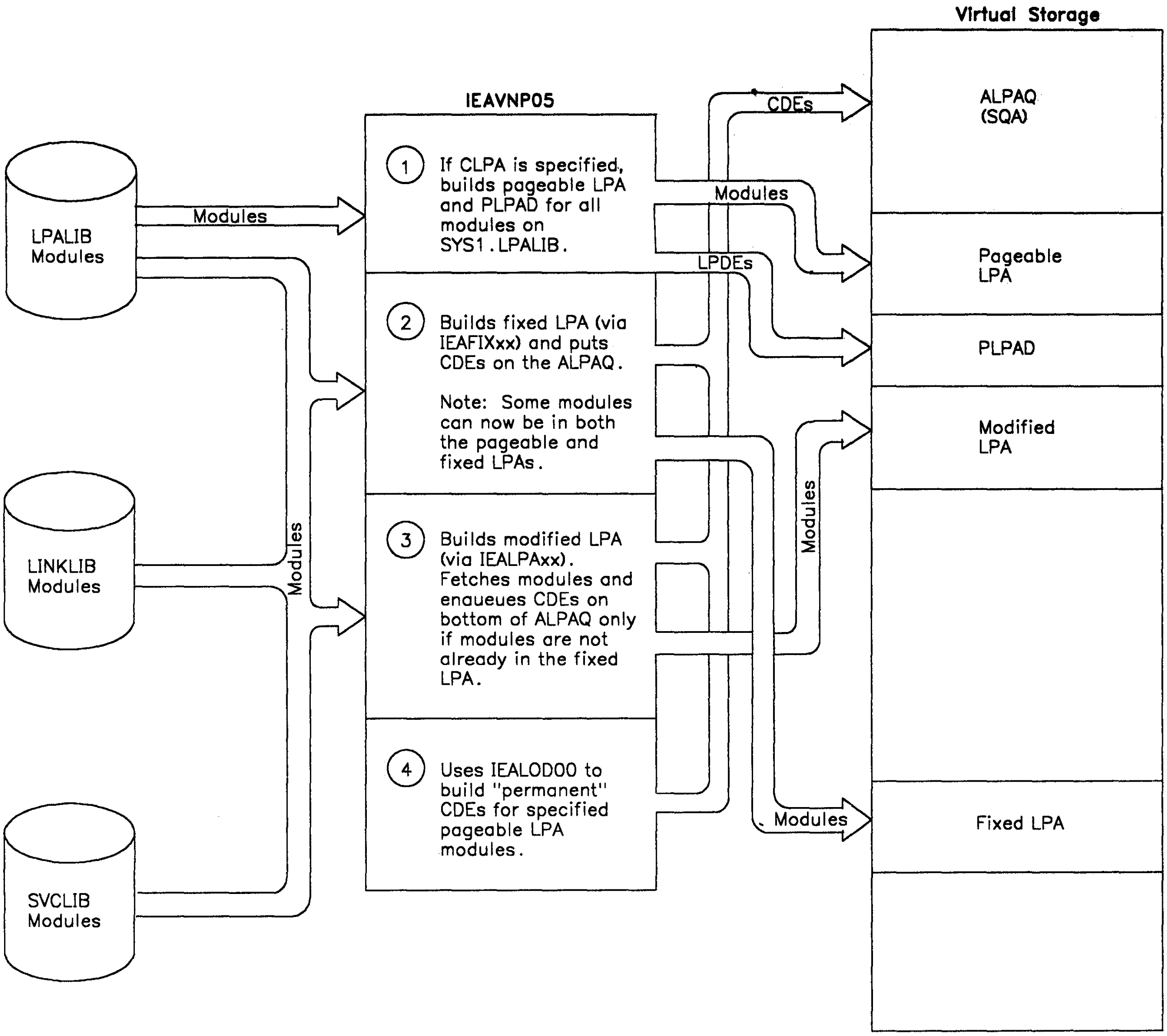

Virtual Storage
IEAVNP05
1
If CLPA is specified, builds pageable LPA and PLPAD for all modules on SYS1.LPALIB.
2
Builds fixed LPA (via IEAFIXxx) and puts CDEs on the ALPAQ.
Note: Some modules can now be in both the pageable and fixed LPAs.
3
Builds modified LPA (via IEALPAxx). Fetches modules and enaueues CDEs on bottom of ALPAQ only if modules are not already in the fixed LPA.
4
Uses IEALOD00 to build "permanent" CDEs for specified pageable LPA modules.
LPALIB Modules
LINKLIB Modules
SVCLIB Modules
Modules
Modules
CDEs
Modules
LPDEs
Modules
Modules
ALPAQ (SQA)
Pageable LPA
PLPAD
Modified LPA
Fixed LPA

Figure 5-9. IEAVNP05 Initialization

Basic Functional Flow

The following section describes the program manager's major functions.

LINK

Module IEAVLK00 is called by the SVC FLIH when the LINK SVC (SVC 6) is issued. Its first function is to locate a usable copy of the requested load module. An abend occurs if it cannot. If a usable copy is found, but not in virtual storage, module IEAVLK01 brings it in.

If a usable copy is found already in virtual storage, the requestor can use the module immediately or may be suspended until it becomes available. In the latter case, a module can be unavailable if either:

- A previous request to fetch it into storage is being processed (indicated by bit CDNIC being on in the CDE) or
- It is a serially reusable module currently in use (indicated by the CDREN bit being off, the CDSER bit being on, and a non-zero CDRRBP field).

If a usable copy of the requested module is not immediately available, the requestor's program manager SVRB is put into a wait state and enqueued on the SVRB suspend queue (SSQ). The SVRB is dequeued and posted out of its wait when the desired module becomes available. For "not in storage" suspends, module IEAVLK01 posts all SVRBs queued on a CDE's SSQ when it successfully completes a module fetch. Each of these SVRBs then restarts the LINK request essentially from the beginning at entry point IEAQCS02 in module IEAVLK00. For the serially reusable case, module IEAVLK02 posts the top SVRB on a CDE's SSQ when the PRB that was using the module represented by the CDE exits. In this case, execution resumes in module IEAVLK00 at entry point IEAQCS03. The logic at this entry point assumes the requested module is in storage and immediately available.

Once a module becomes available to a request, the module-use count in the CDE is increased by one. This use count is decreased by one when the current requestor no longer needs the module.

Next, LINK processing gets storage for a PRB out of subpool 253. The PRB is initialized (including setting the RBOPSW to point to the entry point of the requested module) and enqueued on the current TCB's RB queue. It is enqueued after the program manager SVRB, but before the linking module's RB. The program manager then exits, thus causing the requested load module to gain control next. (See Figure 5-10.)

PRB Field	How initialized by IEAVLK00 for LINK (and ATTACH)
RBPREFIX	zero
RBSIZE	13 double words
RBSTAB1	zero
RBSTAB2	from PM SVRB except RBATTN=0
RBCDFLGS	zero
RBCDE1	@ requested CDE (may be a minor) Note 1
RBOPSW-LH	from caller's RB (or AABC0D00) Note 2
RBOPSW-RH	module entry point from CDENTPT
RBPGMQ	from PM SVRB
RBWCF	from PM SVRB
RBLINKB	@ caller PRB (or @ TCB if ATTACH)
RBGRSAVE	from PM SVRB

Notes:

1. *RBCDE1 will point to the CDE containing the requested load module name. This may be a minor CDE. CDRRBP of the major CDE however will point to the new PRB. Field CDRRBP in a minor CDE has no meaning.*

2. *If ATTACH, RBOPSW (left half) is set to AABC0D00 where,*

AA	*= from current PM PSW*
B	*= from TCB protect key (TCBPKF)*
C	*= X'C' if TCBFSM = 1; X'D', otherwise*
D	*= from PICA if there is one, else 0*

Figure 5-10. New PRB Initialization - LINK

ATTACH

When the ATTACH service routine completes the initialization of the requested daughter TCB, it gives control to LINK in order to establish the first PRB for the daughter TCB. ATTACH simulates the SVC FLIH by creating a program manager SVRB under the daughter TCB and then causing the daughter to branch enter module IEAVLK00 at entry point IEAQCS01. Processing is essentially the same as for LINK except for APF considerations which are explained later.

XCTL

Module IEAVLK00 gets control from the SVC FLIH at entry point IGC007 when the XCTL SVC (SVC 7) is issued. With XCTL, unlike LINK, the first function of module IEAVLK00 is to establish the new RB. The method used depends on the type of caller, as follows:

- If the caller is an SVRB, the caller's SVRB is reused for the new module. It remains in the TCB RB queue in the same position as it was when IEAVLK00 got control.

- If the caller is an IRB, storage is obtained from subpool 255 for a new PRB. The new PRB is then enqueued on the TCB RB queue between the IRB and the program manager SVRB.

- If the caller is a PRB, storage is obtained for a new PRB from subpool 255 and then it is enqueued upon the TCB RB queue following the program manager SVRB. The caller's PRB is then put on top of the queue. The program manager then issues the EXIT SVC (SVC 3) forcing the caller's PRB, since it now is on top of the queue, through exit processing. This results in the storage for the caller's old module being freed before the new module is obtained. The program manager then resumes execution at entry point IEAQCS02 in module IEAVLK00.

Figure 5-11 shows how the new PRB (SVRB in the case where the caller is an SVRB) is initialized for an XCTL. Figure 5-12 shows how the new RB is enqueued in the TCB RB queue before the program manager locates the new load module.

The next function in the XCTL process is to locate the desired module. If the caller is an SVRB, the module is searched for via the ALPAQ; if it is not found, it is searched for via the PLPAD. If it is not found by either the ALPAQ or the PLPAD, an 806 abend is generated. If the load module is found, final initialization in the RB is completed and the program manager exits. The following exceptions to normal processing occur when an SVRB issues an XCTL macro (they are made for performance reasons):

- Only the ALPAQ and PLPAD are searched.

- If the CDE on the ALPAQ is found usable, the use count is *not* increased.

- If an LPDE in the PLPAD is found usable, no CDE is built or enqueued on the ALPAQ. Furthermore, the RBCDE1 field is made to point to the LPDE rather than a CDE.

If the caller is not an SVRB, the requested load module is located as it is in LINK. Once found, initialization is completed on the already existing PRB and return is made to the caller.

	How Initialized by IEAVLK00 for XCTL		
RB Field	**Caller is a PRB**	**Caller is a IRB**	**Caller is an SVRB**
RBPREFIX	zero	zero	left as is
RBSIZE	from caller PRB	17 double words	left as is
RBSTAB1	from caller PRB	zero	left as is except RBTRSVRB, Note 1
RBSTAB2	from caller PRB	from caller IRB except RBFDYN=1	left as is
RBCDFLGS	zero	zero	left as is
RBCDE1	@ requested CDE	@ requested CDE	@ CDE or @ LPDE
RBOPSW-LH	from caller PRB	from caller IRB	left as is
RBOPSW-RH	@ module entry point	@ module entry point	@ module entry point
RBPGMQ	zero	zero	left as is
RBWCF	from caller PRB	from caller IRB	left as is
RBLINKB	@ caller's caller RB	@ calling IRB	left as is
RBGRSAVE	from caller PRB	from caller IRB	left as is

Note:

1. Bit RBTRSVRB indicates (for a dump routine) the location of the load module. It will be set to 0 if the module was located via a CDE on the ALPAQ. It will be set to 1 if the module was located in the pageable LPA.

Figure 5-11. New RB Initialization - XCTL

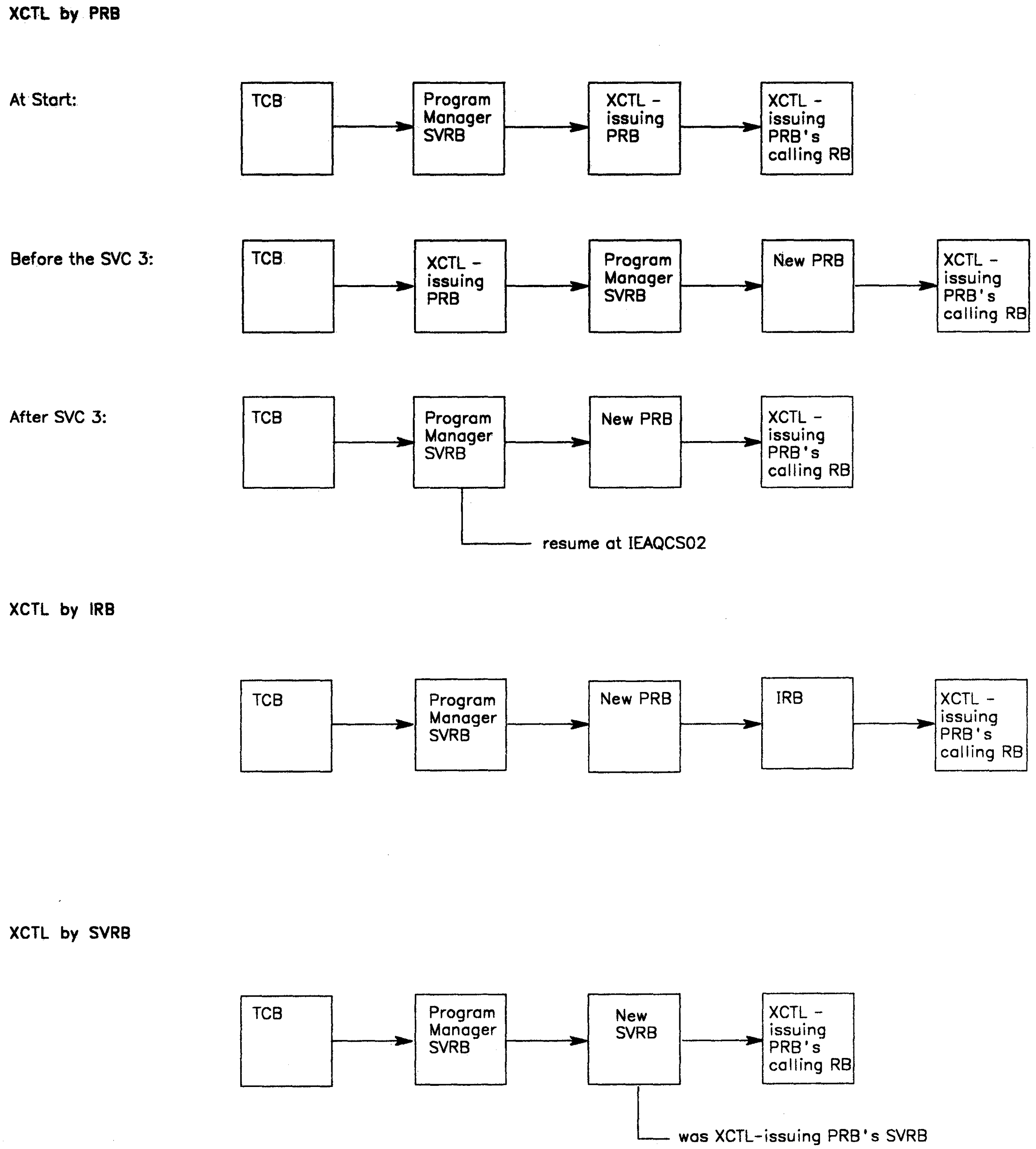


Figure 5-12. XCTL RB Manipulation

LOAD

Module IEAVLK00 is called by the SVC FLIH at entry point IGC008 when the LOAD SVC (SVC 8) is issued. For a LOAD request, the TCB's load list is first searched for an LLE representing a usable copy of the requested module. If found, the LLE total responsibility count is increased by one. In addition, if the caller is in supervisor state and/or key 0-7, the system responsibility count is updated. A separate system count is maintained to prevent a non-system user from deleting a module loaded by a system routine.

If the load list does not yield a usable copy of the requested module, the module is located and CDEs are manipulated as explained earlier for LINK. The final step for LINK processing is the building of the PRB. For LOAD, however, no PRB is built; instead, an LLE is built and enqueued at the *top* of the TCB's load list queue. This LLE points to the CDE (whether it be on the JPQ or the ALPAQ) of the requested module. The total responsibility count is initialized to one, and the system responsibility count to zero or, if a system request, to one.

DELETE

Module IEAVLK00 is called by the SVC FLIH at entry point IGC009 when the DELETE SVC (SVC 9) is issued. Since the module to be deleted must have been previously loaded by the same task, IEAVLK00 searches the TCB's load list queue for the module. If it is not found, the program manager exits with a return code of 4.

If the module is found, the total responsibility count in the LLE is decreased by one. The system responsibility count is also decreased by one if the DELETE was issued by a system program. Finally, the use count in the CDE is decreased by one.

The LLE is dequeued and freed if the total responsibility count goes to zero. If the use count in the CDE also goes to zero, routine CDHKEEP in module IEAVLK02 is called. This routine frees the CDE and all its minor CDEs, the associated extent list, and the module itself. Once control is returned to IEAVLK00, the program manager exits.

Exit Resource Manager

Module IEAVLK02 is called by the exit prologue at entry point IEAPPGMX whenever a PRB exits. The purpose is to clean up the program resources that were being used by the PRB. First, the program manager decreases by one the use count in the CDE being used by the PRB.

If the module is serially reusable, and there are SVRBs suspended on the CDE's SSQ, the top SVRB is posted so it can begin using the module.

If the CDE's use count goes to zero, then the CDE, all its minor CDEs, the extent list, and the module itself are freed. When the module is freed (by subroutine CDHKEEP) it is freed from:

- Subpool 0, if bit CDSPZ is 1
- Subpool 251, if bit CDSPZ is 0 and bit CDJPA is 1
- Subpool 252, if bit CDSPZ is 0 and bit CDJPA is 0

(See the discussion of "Module Subpools" later in this chapter.)

If the exiting PRB is the last in the TCB's RB queue, IEAVLK02 also does end-of-task clean up. This consists of cleaning up and freeing all LLEs remaining on the TCB's load list queue.

SYNCH

Module IEAVLK00 is called by SVC FLIH at entry point IGC012 when the SYNCH SVC (SVC 12) is issued. SYNCH essentially uses the tail end of LINK processing to build and enqueue a PRB for the user exit. No module searching, CDEs, LLEs, etc. are involved.

IDENTIFY

Module IEAVID00 is called by the SVC FLIH at entry point IGC041 when the IDENTIFY SVC (SVC 41) is issued.

IDENTIFY builds a minor CDE for the requested name and entry point. The CDE is enqueued on the JPQ or ALPAQ following the major CDE that represents the module containing the entry point. One exception to this is if the requestor is not authorized (not supervisor state, not in a system key, and not executing in an APF-authorized step) and the embedded entry point is in a module that is CDE-authorized and from an APF-authorized library. In this case, for integrity reasons, a major CDE for the embedded entry point is built and enqueued on the JPQ. Since the CDE is initialized to represent the module as *not* coming from an authorized library, no authorized user is allowed to use this user-defined entry point.

Module IEAVID00 also accommodates OS/LOADER with special processing. When OS/LOADER issues the IDENTIFY SVC, it has loaded a module into subpool 0, built an extent list, and now wants to be represented by a *major* CDE and extent list build and enqueued on the JPQ. This request is called a "major request" and is indicated when Register 0 contains 0 upon entry to IEAVID00. Register 1 contains a pointer to the module name and extent list.

Figure 5-13 illustrates CDE initialization by IDENTIFY.

	Normal Request		
CDE Field	**Normal**	**Non-authorized requestor and module from an APF-authorized library**	**Major Request**
CDCHAIN	(behind major)	(top of JPQ)	(top of JPQ)
CDRRBP	zero	zero	zero
CDNAME	as per input	as per input	as per input
CDENTPT	as per input	as per input	as per input
CDXLMJP	@ major CDE	zero	@ XL (at end of CDE)
CDUSE	zero	zero	zero
CDNIP	as in major CDE	0	0
CDNIC	0	0	0
CDREN	1	as in major CDE	0
CDSER	1	as in major CDE	0
CDNFN	0	0	0
CDMIN	1	0	0
CDJPA	0	1	0
CDNLR	1	as in major CDE	1
CDSPZ	0	0	1
CDXLE	0	0	1
CDRLC	0	0	0
CDOLY	0	1	0
CDSYSLIB	0	0	0
CDAUTH	as in major CDE	0	0

Figure 5-13. CDE Initialization by IDENTIFY

ABEND Resource Manager

Module IEAVLK02 is called by RTM at entry point IEAPPGMA under two circumstances: when a TCB is going to abnormally terminate; and when a program manager SVRB is going to be forced through exit processing because of a recovery retry.

When IEAVLK02 is called, its function is to clean up CDE SVRB suspend queues. If the current TCB has any program manager SVRB on an SVRB suspend queue for any CDE on the JPQ, the SVRB is dequeued. Furthermore, when a TCB is going to abnormally terminate, if any CDE on the JPQ has the CDNIC bit on and a program manager SVRB on the abending TCB's RB queue is responsible for fetching the module into virtual storage, all other SVRBs waiting for the module are posted and the CDE is dequeued and freed.

806 Abend

If the program manager cannot locate a load module in response to a LINK, ATTACH, XCTL, or LOAD request, it issues an 806 abend. Two key areas in the program manager should be understood if an unexpected 806 abend occurs or if the program manager uses a copy of a module that was not anticipated. These are (1) the module search sequence or search order and (2) the criteria used in determining whether or not a module already in virtual storage is useable.

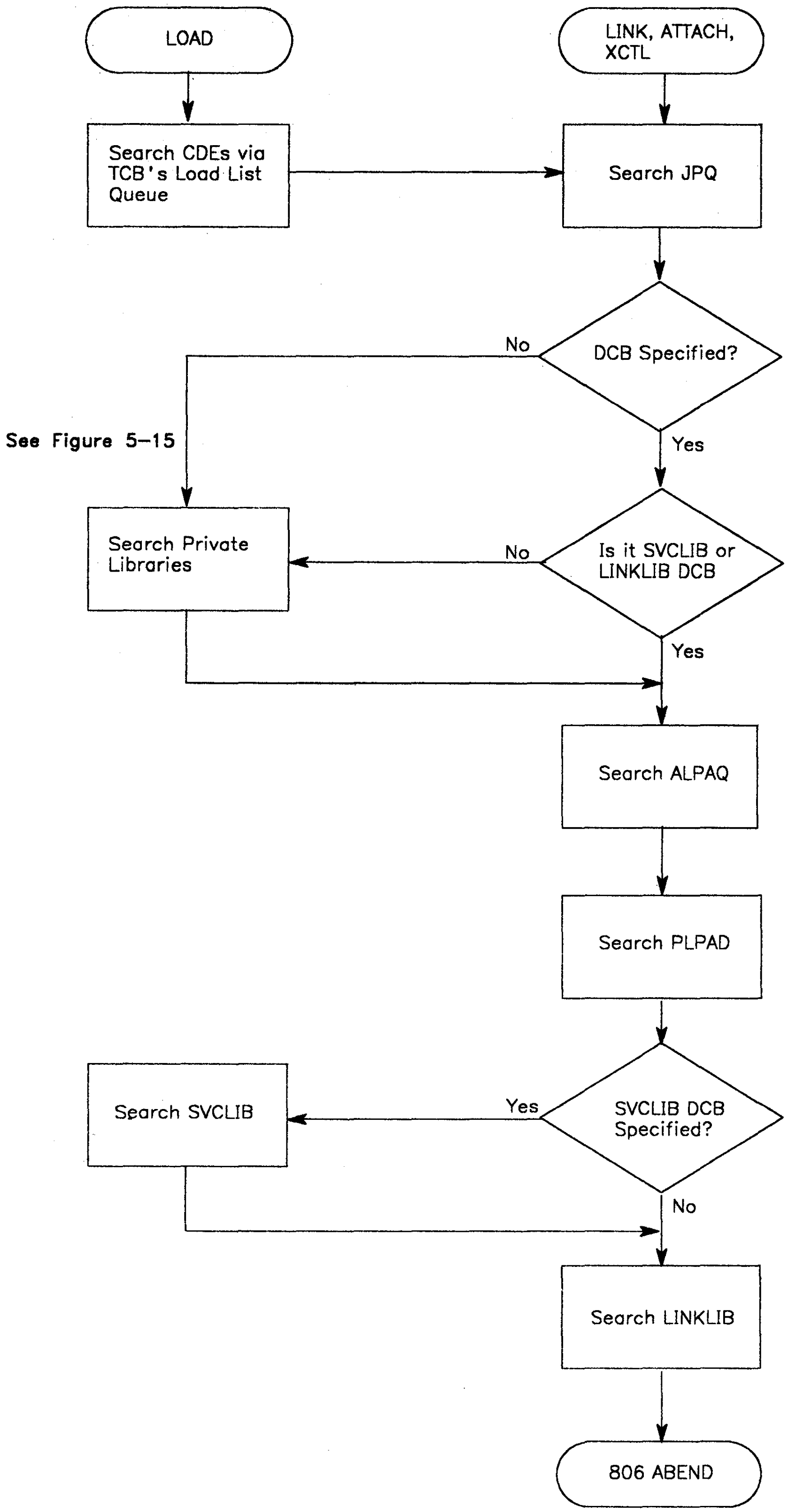
1. Search Sequence

   For a LOAD request, CDEs located on the task's load list queue are first searched for a useable module. If this search fails, the search sequence for LOAD is then the same as it is for LINK, ATTACH, and XCTL.

The search sequence for LINK, ATTACH, XCTL, and LOAD (if the LLE scan is unsuccessful) is shown in Figure 5-14 and Figure 5-15.

2. Usability Criteria

   When searching for a module, the program manager looks for a CDE already enqueued on the JPQ or ALPAQ with a CDNAME the same as that of the requested name. If a matching name is found and the CDE is on the ALPAQ, the module is immediately available to the requestor because all these CDEs represent modules that are reentrant and from APF-authorized libraries. If the CDE is on the JPQ, however, certain tests have to be made to determine if the module represented by the CDE can be used by the requestor. The routine CDALLOC (CDE Allocation) performs this testing. The CDE with the matching name is the input to CDALLOC. Output is a return code indicating the usability of the associated module. Figure 5-16 describes tests and actions taken by CDALLOC.



Notes:

1. For XCTL, if caller is an SVRB, only the ALPAQ and PLPAD are searched.

2. For ATTACH, if the TASKLIB=LINKLIB DCB is specified, no library other than LINKLIB is searched for the requested module and an abend 806-4 might occur.

3. For a LOAD request with the explicit load option (ADDR keyword), only the library indicated by the user-supplied DCB is searched.

4. If the limited library search option is requested (LSEARCH=YES parameter), only the ibrary indicated by the user-supplied DCB is searched. A search of SYS1.LINKLIB involves a preliminary search of the link pack area.

Order of CDEs on ALPAQ

First: Modules activated from the pageable LPA -- newest modules first

Second: Modules in IEALOD00 -- in inverse order of specification in list

Third: Modules in fix lists in inverse order of specification in lists. Lists also in inverse order of their specification

Fourth: Modules in MLPA lists -- in inverse order of specification in lists. Lists also in inverse order of their specification

Figure 5-14. Module Search Sequence for LINK, ATTACH, XCTL and LOAD

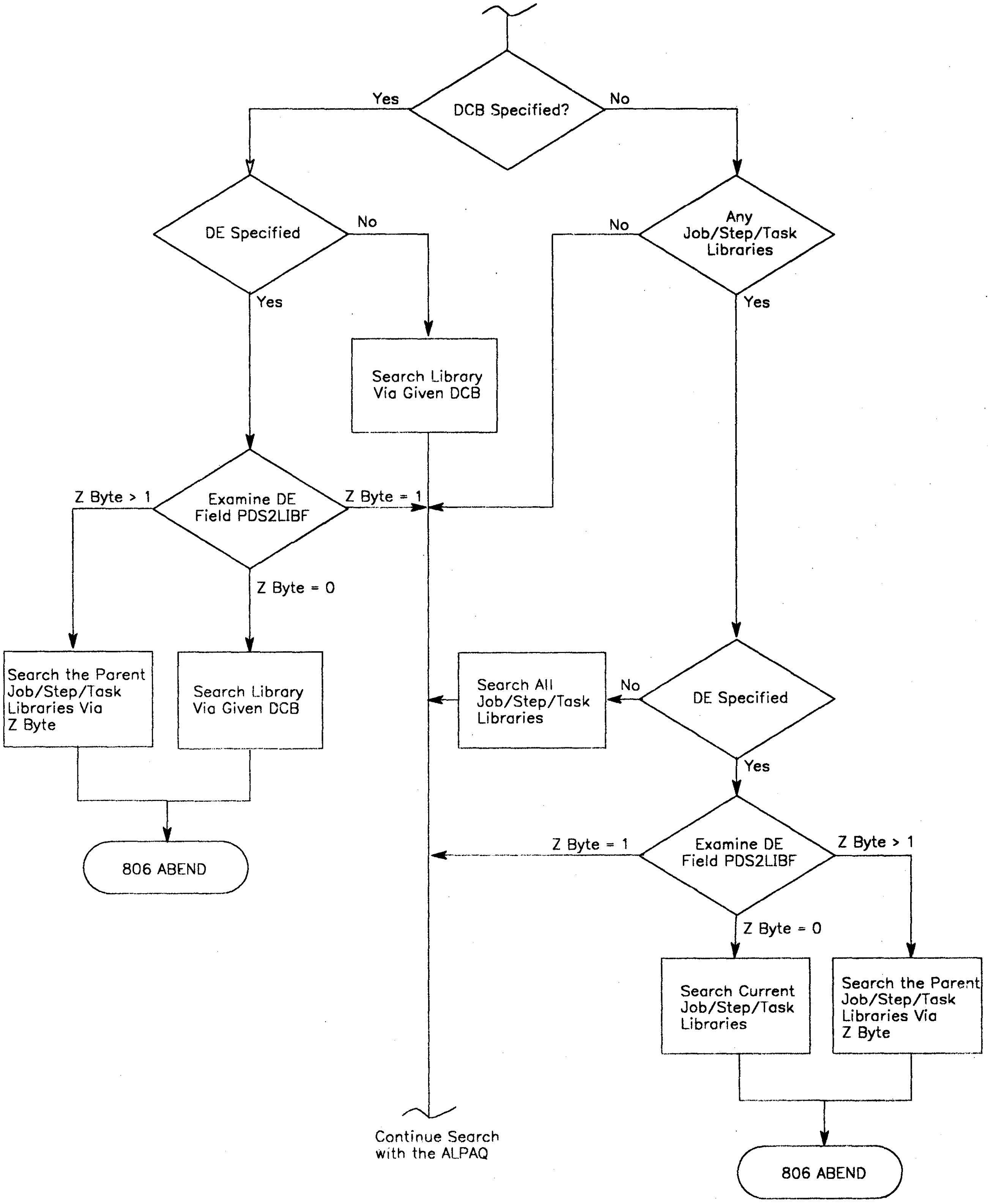

DCB Specified?
Yes
No
DE Specified
No
Yes
Any Job/Step/Task Libraries
No
Yes
Search Library Via Given DCB
Z Byte > 1
Examine DE Field PDS2LIBF
Z Byte = 1
Z Byte = 0
Search the Parent Job/Step/Task Libraries Via Z Byte
Search Library Via Given DCB
806 ABEND
Search All Job/Step/Task Libraries
No
DE Specified
Yes
Z Byte = 1
Examine DE Field PDS2LIBF
Z Byte > 1
Z Byte = 0
Search Current Job/Step/Task Libraries
Search the Parent Job/Step/Task Libraries Via Z Byte
Continue Search with the ALPAQ
806 ABEND

Figure 5-15. Module Search Sequence of Private Libraries

Type of Request		Module Condition via the Input CDE					CDALLOC Return Code
Requestor is Authorized*		Module from non APF-authorized library					8
		From APF-authorized library = same as non-authorized request					-
Requestor is Non-authorized	LOAD	Module being fetched (CDNIC=1)					16
		Module in Storage (CDNIC=0)	No Load Restrictions (CDNLR=1)	Reentrant or serially reuseable			4
				Non-reusable	Fetched by Program Manager		0
					Loaded by OS/LOADER	USECT=0	4
						USECT>0	0
			Load Only	Reentrant or serially reuseable			4
				Non-reuseable			0
	LINK ATTACH XCTL	Module being fetched (CDNIC=1)					16
		Module in Storage (CDNIC=0)	No Load Restrictions (CDNLR=1)	Reentrant (CDREN=1)			4
				Serially Reuseable	In use (CDRRBP”0)		12
					Not in use (CDRRBP=0)		4
				Non-reuseable	Used (CDNFN=1)		8
					Never used (CDNFN=0)		4
			Load only (CDNLR=0)				406 ABEND

0: Module not available via JPQ
4: Module is immediately available
8: Module not available - continue JPQ search
12: Module not immediately available - suspend requestor until module is no longer in use
16: Module not immediately available - suspend requestor until fetch is complete

*In supervisor state, in system key, or as part of an APF-authorized step

Figure 5-16. CDE Allocation

APF Authorization

The program manager performs two APF-related functions. The first function determines whether or not a job step is APF-authorized when the job step TCB is attached. The second function prevents any authorized program from accessing, via LINK, ATTACH, XCTL or LOAD, a module that is not from an APF-authorized library.

1. Establishing APF-Authorization

   An APF-authorized job step is executing if bit JSBAUTH is on in the JSCB. This bit is turned on by the program manager if the following conditions exist when LINK is called by ATTACH:

   - It must be a job step ATTACH. The program manager considers it a job step ATTACH if field TCBJSTCB in the attached TCB points to itself and if there is a JSCB for the step indicated by a non-zero TCBJSCB field.

   - The load module being attached must have been link edited with an APF authorization code of 1. This is indicated to the program manager when bit PDSAPF is on in the module's directory entry.

- The load module being attached must be from an APF-authorized library. This is determined by FETCH and indicated to the program manager by bit WKAUTH being on in the FETWK.

In summary, a job step is APF-authorized if the first program executed in the step is both from an APF-authorized library and link edited with an APF authorization code of one.

2. 306 ABEND

An authorized program is one that is executing in supervisor state, or with a system protect key (0-7), or as part of an APF-authorized job step. An authorized program must LINK to, ATTACH, LOAD and XCTL to modules exclusively from APF-authorized libraries. The program manager issues an abend code of 306 if the only usable copy of a module requested by an authorized program is on a non-APF-authorized library.

When a load module is fetched into virtual storage, FETCH indicates to the program manager via the FETWK bit, WKAUTH, whether it is (bit on) or is not (bit off) from an APF-authorized library. If the requested module is already in virtual storage, the program manager determines whether or not it is from an APF-authorized library by examining the CDE bit, CDSYSLIB. If it is on, the module can be used by an authorized program.

Bit CDSYSLIB = 1 if the associated module is from an APF-authorized library except in the following cases:

- The bit = 0 if the module is reentrant but is still fetched into subpool 251 because of TSO TEST considerations (see the following discussion on "Module Subpools").

- The bit = 0 when IDENTIFY creates a major CDE because a non-authorized user indicates an embedded entry point in a module from an APF-authorized library.

Module Subpools

All modules represented by CDEs on the ALPAQ are loaded into the pageable LPA, the fixed LPA, and the modified LPA. These modules are never freed.

Modules represented by CDEs on the JPQ however, are freed by the program manager and can occupy storage in subpool 0, subpool 251, and subpool 252.

Modules loaded by the OS/LOADER always use subpool 0. This is indicated by bit CDSPZ being set to one.

When bit CDSPZ is zero, modules fetched by the program manager use subpool 251 if bit CDJPA is set on or subpool 252 if bit CDJPA is set off.

Reentrant modules from APF-authorized libraries are always fetched into subpool 252 if the requestor is a system program (a program in supervisor state or with a system key). Reentrant modules from APF-authorized libraries requested by non-system programs are also fetched into subpool 252 provided TSO test (TCB bit TCBTCP = 0) is not running. All other modules are fetched into subpool 251.

FETCH/Program Manager Work Area (FETWK)

Module IEAVLK01 obtains the FETWK (mapped by DSECT IHAFETWK) from subpool 253 when a load module is to be fetched into virtual storage. A pointer to the FETWK is placed in RBCSWORK (at RB+X'74') of the program manager SVRB. FETWK is logically divided into three areas. The first area, up to but not including field WKADDR, is used exclusively by FETCH as a work area. The second area, up to but not including WKPREFX, is used as a work area by the program manager. Field WKIOADDR and bits WKAUTH and WKSYSREQ in this area are also addressed by FETCH, as follows:

- WKIOADDR is always set to zero by the program manager. This instructs FETCH to do the GETMAIN for the load module.
- WKAUTH is set to one by FETCH to tell the program manager when a load module is from an APF-authorized library.
- WKSYSREQ is set to one by the program manager to tell FETCH that the requesting program is in supervisor state and/or system key. FETCH uses this indication to bypass DEB checking.

The third area of the FETWK, starting with WRPREFX, is the BLDL area. This area contains the directory entry used by FETCH to obtain the requested module. The directory entry is placed there by the program manager either by moving a caller-supplied directory entry into the area or by issuing a BLDL.

RB Extended Save Area (RBEXSAVE)

The 48-byte extended save area (RBEXSAVE at RB+X'60') of the program manager SVRB is used by the program manager as a work area. This area, along with the FETWK, is a key area in analyzing program manager problems. RBCSNAME (at RB+X'60') contains the module name requested by the caller, and RBCSDE (at RB+X'68') points to a copy of the caller-supplied directory entry if one was supplied. If EP or EPLOC is specified, then this pointer is zero. Other key areas of the extended save area are RBCSWORK (at RB+X'74'), which points to the FETWK if FETWK was obtained, and bit RBCSSYSR (RB+X'70' = X'40'), which is on if the caller is a system program.

CDE Pool Control

Two pools of CDEs are maintained in SQA (one for minor CDEs and one for major CDEs) from which the program manager obtains the CDEs used to represent modules on the active LPA queue. A sixteen-byte area is also maintained in SQA which is used to control access to the CDE pools. This area is pointed to by CVTCDEQ (CVT+X'3D8'). The format of this control area is as follows:

+0 Pointer to next available major CDE in pool (zero if major pool is empty).
+4 Pointer to next available minor CDE in pool (zero if minor pool is empty).
+8 Pointer to first extent of major pool.
+C Pointer to first extent of minor pool.

Each pool is comprised of one or more extents. Each extent consists of a 4K block of SQA allocated on a 4K boundary. The first four words at the beginning of each extent contain the following information:

+0 "CDEQ" - identifies this 4K page as a CDE pool.
+4 Pointer to next extent of this pool (zero if last or only extent in pool).
+8 Count of CDEs from this extent that are in use.
+C Indicates which pool this extent belongs to (0 = major pool, 4 = minor pool).

The remainder of each extent contains the CDEs which make up the pool. Each CDE in the pool points to the next CDE in the pool using the CDCHAIN field of the CDE. The last CDE in each pool contains a zero in the CDCHAIN field to denote the end of the pool.

CDEs are obtained from the pools by the CELLGET subroutine of IEAVLK01 and returned to the pools by the CELLFREE subroutine of IEAVLK02.

Virtual Fetch

Virtual fetch is a system service that reduces the time and channel contention to locate and bring a load module into storage. This chapter describes virtual fetch and includes the following topics:

- Functional description
- Module organization
- Functional flow
- Control blocks
- Recovery processing
- Debugging hints

For a description of how to use virtual fetch, refer to *OS/VS2 MVS System Programming Library: Job Management.*

Functional Description

Virtual fetch provides the following functions:

Function	**Description**
Initialization	Establishes the virtual fetch service address space, creates a read-only VIO data set containing reformatted load modules, creates a hash table of virtual fetch directory entries (VFDEs) for the modules in the VIO data set, and creates or reactivates the virtual fetch control block (VFCB) in the CSA.
Refresh	Builds a new VIO data set and hash table to reflect any changes made to the load modules and updates the VFCB.
Build	Creates, for the named module, the virtual fetch work area (VFWK) in the user's address space.
Find	Locates, for the named module, the VFDE and copies it into the VFWK in the user's address space.
Get	Brings a copy of the named module from the VIO data set into the user's address space, relocates the address constants, passes control to the named module, receives control back from the module, and returns to the user.

Module Organization

Virtual fetch consists of the following modules:

Module	**Primary Function**
CSVVFCRE alias: CSVVFRSH	Performs the initialization and refresh functions. Contains the cross memory search routine (CSVVFSCH), which obtains the VFDE; the cross memory post routine (CSVVFRSH), which initiates refresh processing; and CSVVFCR1, which reformats load modules to be placed in the VIO data set.
CSVVFGET	Performs the get function. Contains routine CSVVFRM, which is used to free (via FREEMAIN) local module storage.
CSVVFIND	Performs the build and find functions. Contains routine CSVVFORK, which is entered via a non space switch PC instruction and which passes control to the authorized sections in CSVVFIND and CSVVFGET.

CSVVFMEM Cleans up the VFCB when virtual fetch terminates. It also clears the pointers to the virtual fetch data areas in a local address space when the job step task terminates.

CSVVFTCH Obtains a copy of a named module from the VIO data set and relocates the address constants in the user's address space.

The modules that are given control by virtual fetch's get function are effectively nonreusable. Therefore, recursive calls using virtual fetch will fail.

Additional information for the modules is shown in Figure 5-17.

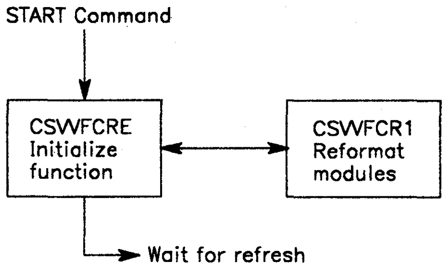
Module Name	**Routines**	**Residence**	**External Entry Points**
CSVVFCRE alias: CSVVFRSH	CSVVFCRE CSVVFCR1 CSVVFSCH CSVVFRSH CSVVFCES-ESTAE CSVVFCFR-FRR	Private (virtual fetch)	CSVVFCRE CSVVFRSH
CSVVFGET	CSVVFGET CSVVFGES-ESTAE CSVVFRR-FRR	Nucleus	CSVVFGET CSVVFRM
CSVVFIND	CSVVFIND CSVVFFES-ESTAE	Nucleus	CSVVFIND CSVVFORK
CSVVFMEM	CSVVFMEM	Nucleus	CSVVFMEM
CSVVFTCH	CSVVFTCH	Nucleus	CSVVFTCH

Figure 5-17. Virtual Fetch Modules

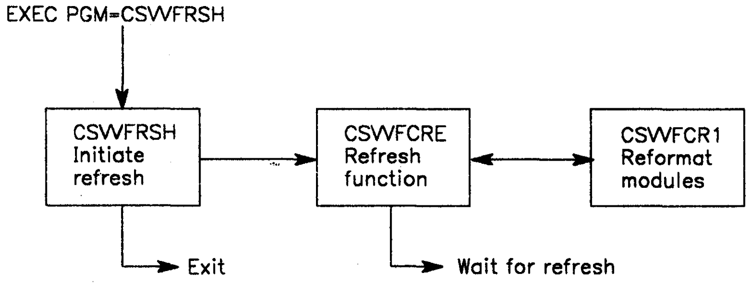
Functional Flow

Flow of control for the virtual fetch modules and routines are shown in the following topics.

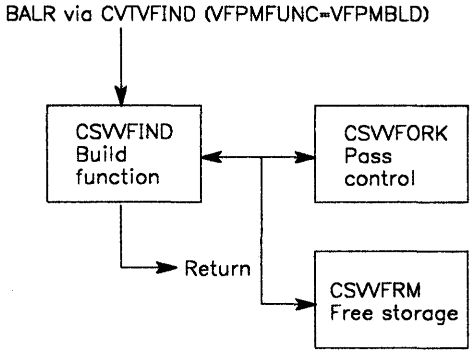
Initialization Function



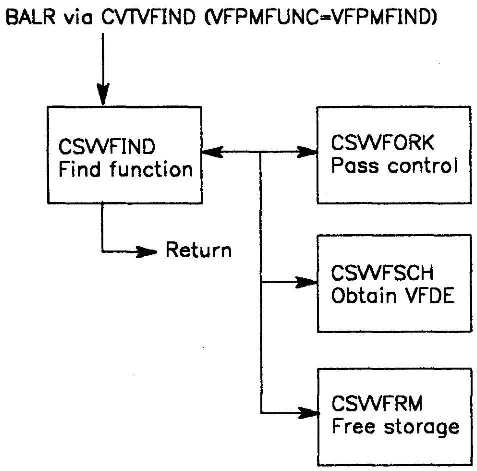
Refresh Function



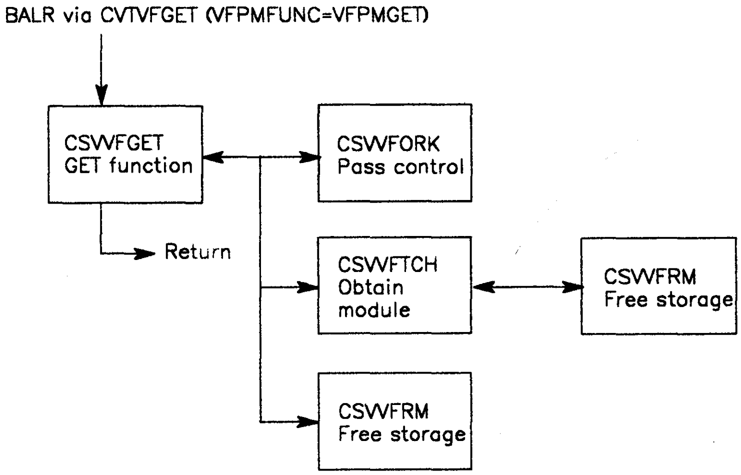
Build Function



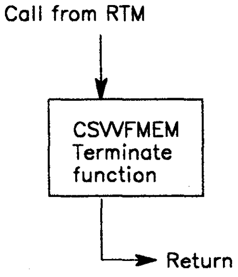
Find Function



Get Function



Termination



Control Blocks

Virtual fetch uses the following control blocks and data areas:

Area	Description
VFCB	Virtual fetch control block - used to access the virtual fetch service address space. The VFCB is located in the CSA and is pointed to by field CVTVFCB in the CVT.
VFHE	Virtual fetch hash entry - contains the virtual fetch directory entries (VFDEs) for the modules managed by virtual fetch. VFHEs reside in hash collision queues in a hash table in the virtual fetch address space. The first elements of the queues are accessed by the module name.
INFODATA	Contains PDS directory entry data. INFODATA format DEs are accumulated in a temporary table (INFOTAB), copied into the hash table, and then converted into VFHEs. INFOTAB is located in the virtual fetch address space.
VFPM	Virtual fetch parameter list - used by the user to request build, find, or get processing. The VFPM is located in the user's address space. For get processing, virtual fetch sets flags in the VFPMRTN field if the named module did not receive control or abended.
VFVT	Virtual fetch vector table - controls the build, find, and get requests in the user's address space. The VFVT is located in the user's address space.
VFWK	Virtual fetch work area - controls the use of a module in a user's address space. The VFWK is located in the user's address space. VFWKs are in push-down queues anchored in a 31-way hash table in the VFVT. The hash table is accessed by module name.
LWK	Local work area - contains dynamic storage for build, find, and get requests. The LWK is located in the user's address space.

Additional information for these control blocks and data areas is shown in Figure 5-18.

Area	Mapping Macro	Size in Bytes	Subpool Number	Created By	Deleted By
VFCB	IHAVFCB	32	241 (CSA)	CSVVFCRE	CSVVFCRE (System VFCB not deleted)
VFHE	IHAVFDE	64	230 (PVT)	CSVVFCRE	CSVVFCRE
INFO-DATA	IHAVFINF	64	0 (PVT)	CSVVFCRE	CSVVFCRE
VFPM	IHAVFPM	88	User key	User	User
VFVT	IHAVFVT	144	254 (LSQA)	CSVVFIND	Job step task termination
VFWK	IHAVFWK	112	254 (LSQA)	CSVVFIND	Job step task termination
LWK	Declares in CSVVFIND and CSVVFGET	276	230 (PVT)	CSVVFIND and CSVVFGET	CSVVFIND and CSVVFGET

Figure 5-18. Virtual Fetch Control Blocks

Recovery Processing

This topic discusses virtual fetch recovery processing. Also see Appendix C for diagnostic information for the SVC dumps issued by virtual fetch modules.

Error During Initialization Processing

When entered, the ESTAE routine CSVVFCES (in CSVVFCRE) attempts to clean up and retry the initialization or refresh request. If retry is not performed, CSVVFCES requests an SVC dump and writes a software record to SYS1.LOGREC. (The dump and LOGREC record are not requested when the abend code is 222, 322, or 522.) VSVVFCES percolates the error under the following conditions:

- There is no SDWA.
- The SDWA indicates that a retry is not permitted.
- A previous retry attempt was unsuccessful.
- The error occurs during initialization before the VFCB is activated.
- The error occurs while the VFCB is being updated.
- The error occurs while waiting for a refresh request.

If an error occurs while making VIO requests, CSVVFCRE issues abend code C0D to enter its ESTAE routine. Under some conditions, CSVVFCRE will terminate with a return code. See *SPL: Job Management* for a description of these return codes.

If an error occurs during CSVVFCES processing, the FRR routine CSVVFCFR (in CSVVFCRE) requests an SVC dump and writes a software record to SYS1.LOGREC.

Debugging Hint - During initialization processing, if an abend 878 occurs or message CSV119I or CSV113I is issued, increasing the region size for virtual fetch might correct the problem.

Errors During Build, Find, and Get Processing

The build and find recovery routine (CSVVFFES) and the get recovery routines (CSVVFGES, ESTAE; CSVVFRR, FRR) clean up and attempt to retry the request. CSVVFFES requests percolation of the error when the SDWA indicates that only clean up is permitted, if the abend code is X'x22', or if the previous retry attempt was unsuccessful. CSVVFGES requests percolation of the error under the same conditions as for CSVVFFES, but in addition, CSVVFGES requests percolation for abends that occur in the requested program. CSVVFRR only requests percolation when the SDWA prohibits retry requests or when the previous retry attempt was unsuccessful.

Except in the case of an X'x22' abend, the recovery routines request an SVC dump and write a software record to SYS1.LOGREC. If retry succeeds, CSVVFIND returns to its caller with a return code X'20', and CSVVFGET returns to its caller with error information in field VFPMRTN of the VFPM. If the caller's VFPM is not in storage that the caller can store into, a protection or translation exception results and the VFPM is not altered.

Debugging Hints

The following conditions can occur during build, find, or get processing:

- If an abend occurs in CSVVFSCH (cross memory search routine), the virtual fetch service address space is flagged as unusable. Recovery routine CSVVFFES (in CSVVFIND) turns off flag VFCBUILT, issues message CSV118E, and returns a code of 20 to the caller. In this case, the virtual fetch service address space should be cancelled, and then virtual fetch should be restarted.

- If an abend occurs while virtual fetch is searching the VFWK collision queues, recovery routine CSVVFFES (in CSVVFIND) or CSVVFGES (in CSVVFGET) turns off flag VFVTVFUP. Further requests for virtual fetch processing in this user's address space will result in invalid return codes until the user's job step task terminates.

- If an abend occurs while virtual fetch is searching or updating the job pack queue, the VFWK in the user's address space is flagged as unusable. Recovery routine CSVVFFES (in CSVVFIND) or CSVVFRR (in CSVVFGET) turns on flag VFWKBADV in the VFWK containing the CDE being queued. The associated module is not obtainable via virtual fetch until the job step task terminates. If the abend occurred while removing the CDE from the job pack queue after the requested program completed processing, CSVVFGET returns to the caller with no flags on in VFPMRTN.

- If either an abend occurred while assigning or page-loading a page for a module, or IEAVAMSI returns a nonzero return code to virtual fetch, the associated module is marked unobtainable in the VIO data set. CSVVFTCH turns on flag VFWKBADM. (The flag is turned off only after an ASSIGN with PAGELOAD has successfully completed processing or if a cancel abend occurs during ASSIGN processing.) The module cannot be obtained via virtual fetch until the virtual fetch service address space is refreshed, or cancelled and restarted.

- If a requested program abends, the program is not available via virtual fetch until a find request is invoked for the program under the TCB under which it abended. This ensures that any dumps resulting from the abend are able to include the storage of the requested module.

- If a GETMAIN macro fails during get processing, CSVVFTCH attempts to obtain storage by issuing FREEMAIN macros for the storage of inactive modules, and then retries the GETMAIN macro. Subsequent get processing will issue GETMAIN macros for any modules whose storage was freed as they are requested. CSVVFGET turns flag VFPMRESH on if CSVVFTCH cannot obtain storage. When the find function is invoked and if CSVVFIND cannot obtain storage, CSVVFIND returns a code of 16 to the user. If the find function obtains storage and the get function is reinvoked, and the get function fails again with VFPMRESH on, then find processing should not be reinvoked unless storage can be freed by the user.

- A module is not obtainable from the current generation of virtual fetch in a user's address space if (1) a page is obtained by CSVVFTCH for a module, and (2) the page is released by CSVVFGET after the module completed execution, and (3) the page is referenced before the next call to IEAVAMSI by CSVVFTCH. This is because IEAVAMSI is not able to complete an ASSIGN and PAGELOAD for the module. CSVVFTCH leaves flag VFWKBADM on for the module.

- For get processing, flag VFPMRESH in the user's VFPM is turned on if any of the following conditions is encountered:

  - The virtual fetch service address space is not active.

  - The VFVT or the required VFWK for the module is not usable.

  - A find request for a module was not issued after the module had abended.

  - A build request and a find request were not issued for a module before a get request was issued for the module.

  - Virtual fetch found the job pack queue in error. There are probably bad pointers in the queue. Recovery routine CSVVFGES or CSVVFRR flags the VFWK as unusable.

  - CSVVFTCH encounters an abend during ASSIGN with PAGELOAD processing. CSVVFGES assumes the module cannot be obtained from the VIO data set and leaves flag VFWKBADM on.

  - Get processing was requested after find processing had indicated that the VFDE was not in the current hash table.

  - The virtual fetch service address space was refreshed or reinitialized after the previous find request for the module.

  - Get processing received a nonzero return code from IEAVAMSI. CSVVFGET assumes the module cannot be obtained from the VIO data set and leaves flag VFWKBADM on.

  - CSVVFRM abended when called by CSVVFGES. CSVVFRR assumes the VFWK is invalid and marks it unusable.

VSM

Virtual storage management (VSM) is responsible for allocating virtual storage, keeping track of what is allocated and, for certain subpools, recording to whom it is allocated. These services are performed both for the system and problem programs. (See Figure 5-19.)

The following are the five basic functions that VSM performs:

Function	Module Performing Function	Comments
Nucleus initialization (NIP)	IEAVNPO8	IPL parameters are: SQA=, CSA=, REAL=, VRREGN=
Address space initialization	IEAVGCAS	Called by memory create
Step initialization/termination	IEAVPRT0	GETPART/FREEPART
Virtual storage allocation	IEAVGM00	GETMAIN/FREEMAIN
Cell pool management	IEAVGTCL IEAVFRCL IEAVBLDP IEAVDELP	GETCELL (get cell) FREECELL (free cell) BLDCPOOL (build cell pool) DELCPOOL (delete cell pool)

The nucleus initialization program (NIP) is not discussed in this book. The remaining VSM functions, and the related FRRs (functional recovery routines), are described in the following topics:

- Address Space Initialization
- Step Initialization/Termination
- Virtual Storage Allocation
- VSM Cell Pool Management
- Miscellaneous Debugging Hints

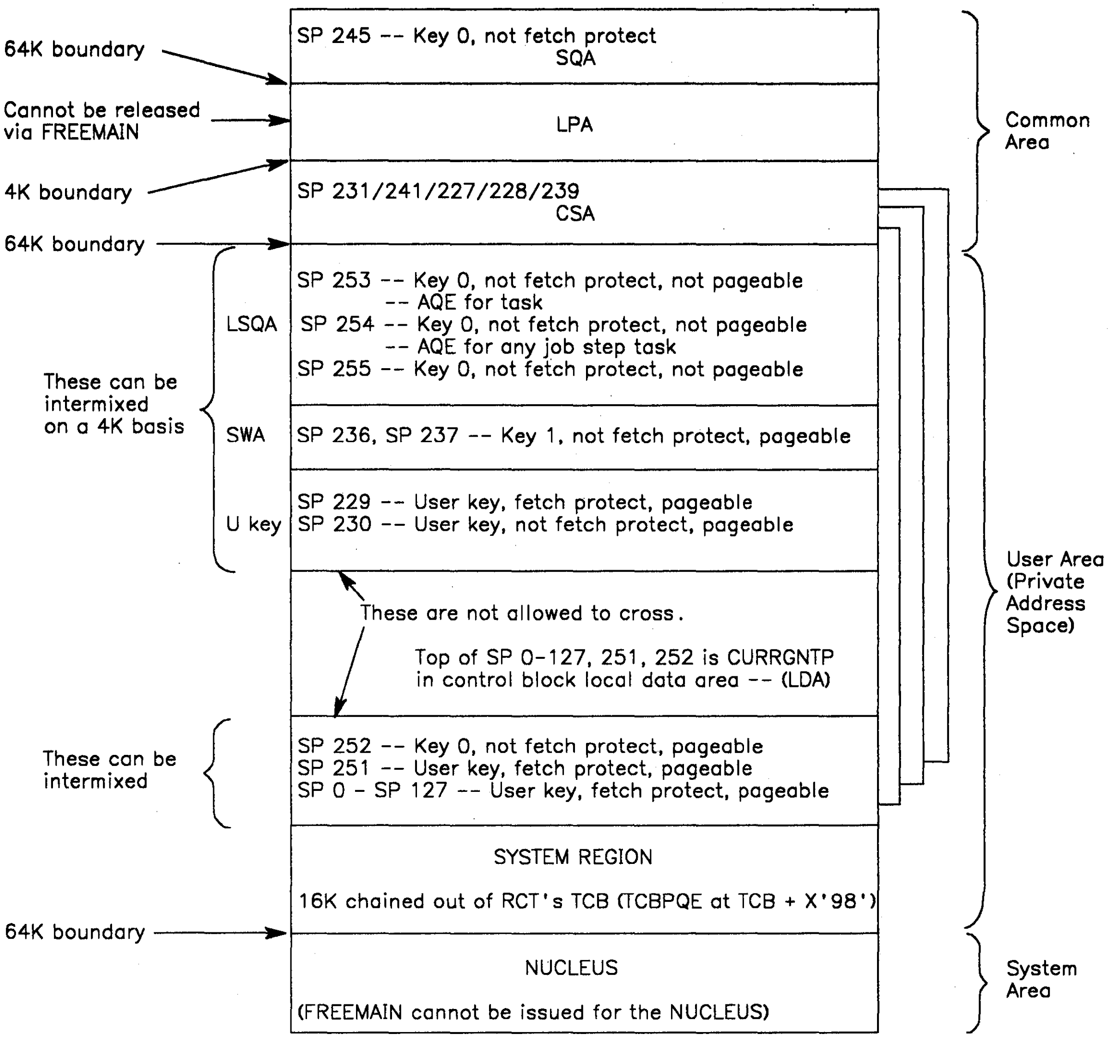

64K boundary
Cannot be released via FREEMAIN
4K boundary
64K boundary
SP 245 -- Key 0, not fetch protect
SQA
LPA
SP 231/241/227/228/239
CSA
Common Area
These can be intermixed on a 4K basis
LSQA
SP 253 -- Key 0, not fetch protect, not pageable
-- AQE for task
SP 254 -- Key 0, not fetch protect, not pageable
-- AQE for any job step task
SP 255 -- Key 0, not fetch protect, not pageable
SWA
SP 236, SP 237 -- Key 1, not fetch protect, pageable
U key
SP 229 -- User key, fetch protect, pageable
SP 230 -- User key, not fetch protect, pageable
These are not allowed to cross.
Top of SP 0-127, 251, 252 is CURRGNTP in control block local data area -- (LDA)
These can be intermixed
SP 252 -- Key 0, not fetch protect, pageable
SP 251 -- User key, fetch protect, pageable
SP 0 - SP 127 -- User key, fetch protect, pageable
User Area (Private Address Space)
SYSTEM REGION
16K chained out of RCT's TCB (TCBPQE at TCB + X'98')
64K boundary
NUCLEUS
(FREEMAIN cannot be issued for the NUCLEUS)
System Area

Figure 5-19. Virtual Storage Management's View of MVS Storage

Address Space Initialization

The create address space module (IEAVGCAS) initializes the VSM address space. IEAVGCAS creates the local data area (LDA). The LDA is the key anchor block and VSM save area for space allocation within an address space. IEAVGCAS builds all the blocks labeled "C" in Figure 5-20. IEAVGCAS also builds the initial allocation of 16-byte LSQA elements for VSM's local cell pool. GETMAIN then allocates VSM's internal control blocks from this pool.

IEAVGCAS also contains VSM's task termination and address space termination resource managers. The task termination routine frees all non-shared space anchored in the TCB. (See Figure 5-20.) The address space termination routine frees any WAIT or POST elements of a V=R (virtual=real) job that are associated with the terminating address space and are chained out of VSM's GDA (global data area). These V=R WAIT/POST elements exist only when a job is waiting for V=R address space.

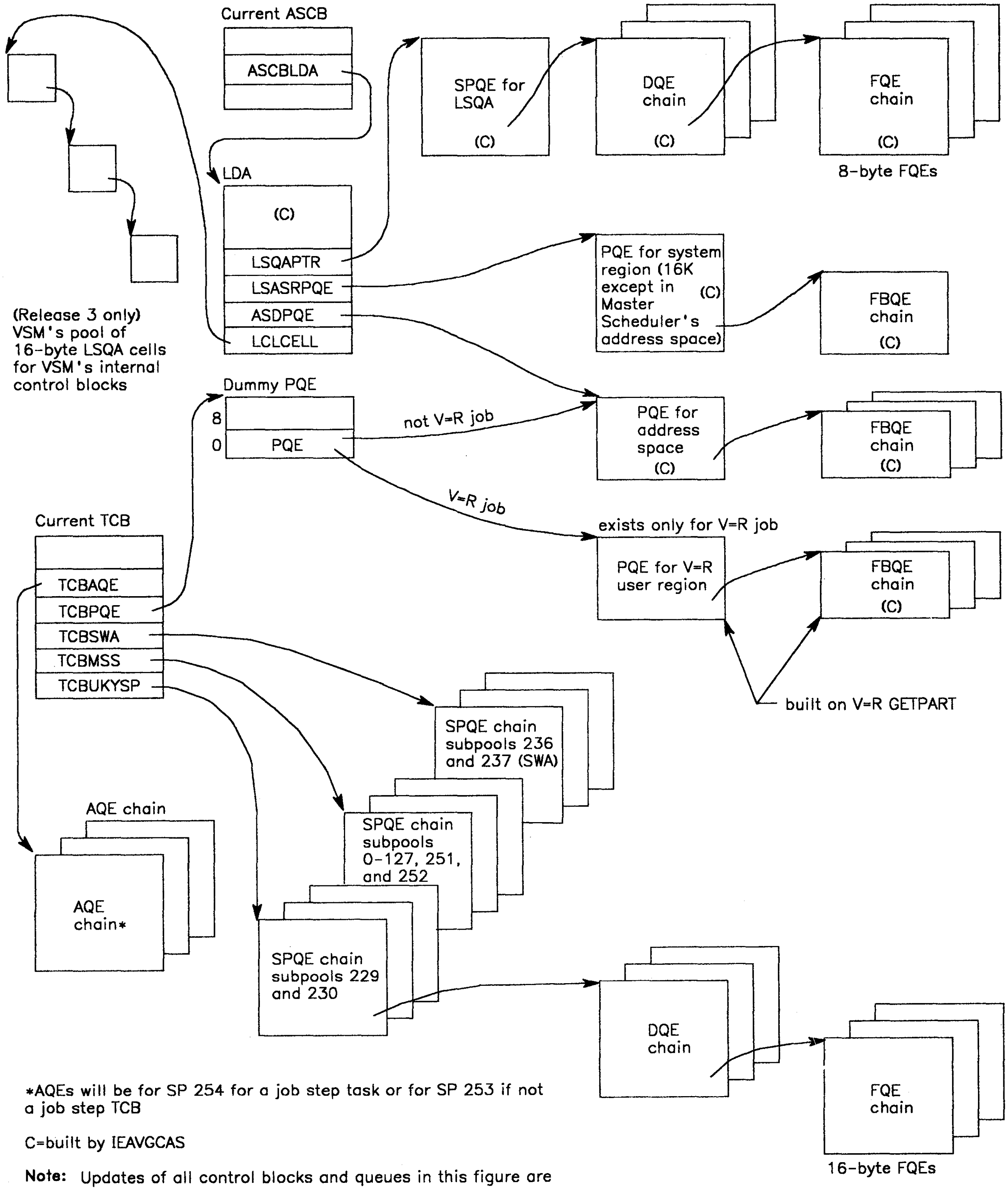
IEAVGCAS's functional recovery routine (FRR) is IEAVCARR. IEAVCARR is divided into the following three sections, corresponding to those of IEAVGCAS.

1. Entry IEAVCARR, which protects the create address space portion of IEAVGCAS.

2. Entry IEAVTTRR, which protects the task termination portion of IEAVGCAS.

3. Entry IEAVFARR, which protects the address space termination portion of IEAVGCAS.

IEAVCARR does not place data in the variable recording area of the SDWA (STAE diagnostic work area). It does invoke SDUMP and either retries at the beginning of the function that detects the error or continues with termination.



*AQEs will be for SP 254 for a job step task or for SP 253 if not a job step TCB

C=built by IEAVGCAS

Note: Updates of all control blocks and queues in this figure are serialized by the local lock.

Figure 5-20. Virtual Storage Management's Control Block Usage

Step Initialization/Termination (IEAVPRT0 - GETPART/FREEPART)

IEAVPRT0 is invoked by IEAVGM00 (GETMAIN/FREEMAIN) via a branch entry as a result of a GETMAIN/FREEMAIN request from an initiator for subpools 242 (V=R) or 247 (V=V). IEAVPRT0 does not return to IEAVGM00; it returns directly to the initiator.

IEAVPRT0 performs four functions, as follows:

1. For a V=V GETPART request:

   - Sets TCBPQE to point to the dummy address space partition queue element (PQE) that was created at address space initialization time.

   - Calls the initiator exit routine IEALIMIT in order to establish the LDALIMIT which is the value used by GETMAIN as an upper limit for problem program subpool GETMAIN's requests. The *OS/VS2 System Programming Library: Supervisor* contains a discussion on LDALIMIT, REGION=, and variable GETMAIN requests.

2. For a V=R GETPART request:

   - Performs IEALIMIT processing as described above.

   - Attempts to obtain V=R space by interrogating V=R FBQEs chained from the GDA.

     - If unsuccessful:

       - Creates V=R wait element
       - Chains the V=R wait element from the GDA
       - Indicates the V=R wait element is waiting for space

     - If successful:

       - Interfaces with RSM's (real storage manager) IEAVEQR to obtain real frames

       - Builds V=R dummy PQE, V=R PQE, and V=R FBQEs, and updates TCBQE

3. For a V=V FREEPART request:

   - Frees all task-related space by calling FREEMAIN, and freeing one subpool at a time.

   - Frees SPQEs and task-related subpools.

   - Sets TCBPQE=0.

4. For a V=R FREEPART request:

   - Performs the same functions as for V=V FREEPART.
   - Returns space to V=R FBQEs chained from the GDA.
   - Attempts to satisfy V=R waiting requests by posting the waiting initiator. The waiting initiator reissues the request; IEAVPRT0 will move the WAIT elements to the POST queue anchored in the GDA. This POST element is freed by GETPART when the initiator's new request is received.

IEAVPRT0's FRR, IEAVGPRR, does not use the variable recording area of the SDWA. It attempts a retry back into IEAVPRT0 based on footprints set in the FRR's six-word parameter area. Refer to the IEAVPRT0 code (microfiche) for a detailed description of this FRR parameter area.

Virtual Storage Allocation (GETMAIN/FREEMAIN)

IEAVGM00, IEAVGM03, and IEAVGM04 satisfy all GETMAIN/FREEMAIN requests. The control block structure they use is shown in Figure 5-20 and Figure 5-21. All GETMAIN processing for the private area subpools and all associated control blocks are serialized by the local lock. All common area subpools and associated control blocks are serialized by the SALLOC lock.

A detailed process flow through GETMAIN for a virtual storage allocation request can be found in Appendix A in the GETMAIN/FREEMAIN process flow description.

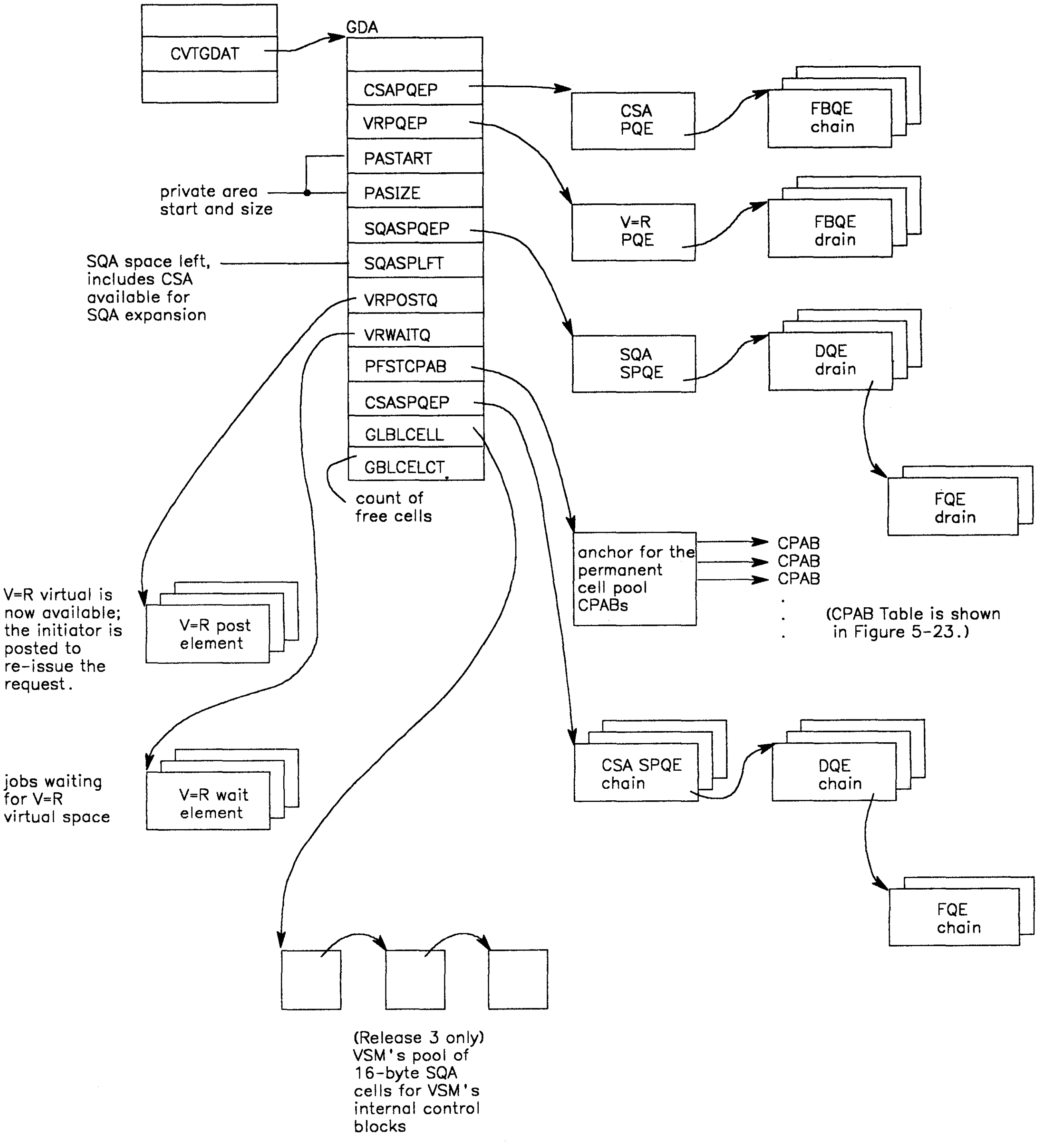
In debugging GETMAIN, the most important information is contained in the original request for virtual storage. This information can be obtained from the trace table, from a parameter list passed by the problem program code, or sometimes from the LDA (local data area).

The LDA cannot always be relied upon to provide information about the request unless the system is stopped immediately. Unless you insert a code or SLIP trap and take a stand-alone dump on error, the LDA is overlaid by another request during the dumping procedure.

If an immediate stop has been obtained upon encountering an error, the most useful information in the LDA is obtained from the flags in the LDARQSTA (LDA+X'10') field. The LDARQSTA contains the current request status. Compare the flags in this field to the request and determine if the two are consistent. Then check through the control block chain, the LDA and GDA chains that are set up when subpools are requested to find out why the abend or error occurred.

LDARQSTA (LDA + X'10')

Offset		
0	1...	Subpool 254 Requester has TCBABGM on
	.1..	Explicit V = V Region reached
	..1.	Variable Request, Pass 2
	...1	SQA or LSQA Expansion
	.... 1...	Deferred Error I/O Flag
	.... .1..	FMAINB or MRELEASR Request
	.... ..1.	GETMAINB Request
	1	Branch Entry
1	1...	4096-byte Request from Subpool 253/254
	.1..	An AQE is needed
	..1.	Fetch Protected Subpool
	...1	Authorized User Key Subpool
	.... 1...	SWA Subpool
	.... .1..	LSQA Subpool
	.... ..1.	CSA Subpool
	1	SQA Subpool
2		SVC Number
3	.... ..1.	Subpool FREEMAIN
	1	Supervisor Mode (if zero)


CVTGDAT
GDA
CSAPQEP
VRPQEP
PASTART
PASIZE
SQASPQEP
SQASPLFT
VRPOSTQ
VRWAITQ
PFSTCPAB
CSASPQEP
GLBLCELL
GBLCELCT
private area start and size
SQA space left, includes CSA available for SQA expansion
count of free cells
CSA PQE
FBQE chain
V=R PQE
FBQE drain
SQA SPQE
DQE drain
FQE drain
anchor for the permanent cell pool CPABs
CPAB
CPAB
CPAB
(CPAB Table is shown in Figure 5-23.)
V=R virtual is now available; the initiator is posted to re-issue the request.
V=R post element
jobs waiting for V=R virtual space
V=R wait element
CSA SPQE chain
DQE chain
FQE chain
(Release 3 only) VSM's pool of 16-byte SQA cells for VSM's internal control blocks

Note: Updates of all the control blocks and queues in this figure, except PFSTCPAB, are serialized by the SALLOC lock. PFSTCPAB is read only after NIP.

Figure 5-21. Virtual Storage Management's Global Data Area (GDA)

GETMAIN's Functional Recovery Routine - IEAVGFRR

Whenever GETMAIN's FRR (IEAVGFRR) finds an error in a queue, an entry is made in the SDWA variable recording area, SDWAVRA (SDWA+X'194') to indicate the error that has been found, its location, and the corrective action taken. Each entry is added to the next available location and the length of the user-supplied data is increased (field SDWAURAL, SDWA+X'193'). The high-order byte (byte 0) of the first word contains FF if the space in the variable recording area was exceeded and data entries were lost. The low order byte (byte 3) of the first word contains a digit indicating the type of error that caused IEAVGFRR to get control. This digit comes from FRRBRNDX (branch index FRR) in the LDA where it is set up by IEAVGM00, IEAVGM03, or IEAVGM04. FRRBRNDX is X'1F' into the GETMAIN/FREEMAIN work area (GMFMWKAR); GMFMWKAR is the portion of the LDA that is used by IEAVGM00 as a work area. It is mapped at the end of this chapter.

The second word of the SDWA variable recording area contains the LDA field LDARQSTA at the time of error. The contents of LDARQSTA are described in the previous topic "Virtual Storage Allocation (IEAVGM00 - GETMAIN/FREEMAIN)."

The next 16 words usually contain the registers (order 0-15) at the time IEAVGM00 was entered. These registers are useful for debugging problems that occur on branch entry requests. Register 14 contains the caller's return address.

The remaining SDWAVRA entries consist of one to three words each. The first word of each entry will have a code in the high-order byte and a data length in the low-order byte. If this length is 0, there is no additional data for this entry. A length of 4 or 8 indicates one or two additional words of data. These data words always contain the address of the affected field or control block. These are all shown in the table in Figure 5-22 Control blocks are checked to determine if they are in the correct subpool, for example, SQA or LSQA; queues are checked for validity.

If an error occurs in IEAVGM00, IEAVGM03, or IEAVGM04, the name of the module in error is indicated in the SDWA. If recovery percolates to the FRR routine IEAVGFRR, the SDWA module name in error is indicated as IEAVGM00, and the CSECT name is indicated as IEAVGM00, IEAVGM01, IEAVGM03, IEAVGM04, or IEASMFGF.

	Data	Data Addresses		
Code	Length	First	Second	Explanation
01	4	PVTLCSA	-	PVTLCSA is incorrect - it will remain unchanged.
02	4	PASTRT	-	PASTRT in GDA is incorrect - it is reset using PVT.
03	4	PASIZE	-	PASIZE in GDA is incorrect - it is reset using PVT.
04	0	-	-	All three sources of CSA start addresses (GDA, PVT, CSA, PQE) disagree - no change will be made.
05	4	PVTHQSA	-	PVTHQSA is incorrect - it will remain unchanged.
06	4	Bad TCB Pointer	-	TCB pointer is not valid - no queue repairing is attempted.
B1	4	SPQE with bad pointer	-	Next SPQE pointer is incorrect - the SPQE queue is truncated (by setting the bad pointer to zero).
B2	4	SQA SPQE	-	SQA SPQE flagword is incorrect - it will remain unchanged.
B3	4	LSQA SPQE	-	LSQA SPQE flagword is incorrect - it will remain unchanged.
B4	4	SPQE	-	Next SPQE pointer = 0, but last SPQE flag is not on - the last SPQE flag is set on.
C1	4	Cell with bad pointer	-	Next cell address is incorrect - the cell pool chain is truncated.
D1	4	SQA SPQE	-	First SQA DQE pointer (in SPQE) is incorrect; it points outside SQA so all of SQA may be lost - it will remain unchanged.
D2	8	DQE with bad pointer	Bad DQE address	DQE pointer is not in SQA or LSQA - DQE queue is truncated (by setting the bad pointer to zero).
D3	4	DQE	-	DQE area address or length is incorrect - this DQE is removed from the queue.
D4	8	Current DQE	Overlapped DQE	Current DQE area overlaps a previous DQE - current DQE is removed from queue.
D5	8	Current DQE	Previous DQE	Circular DQE queue - queue is truncated after previous DQE.
D6	4	DQE	-	First SQA DQE area address or length is incorrect - address and length are corrected.
F1	4	FQE	-	Incorrect type flag in FQE - the flag is corrected.
F2	4	DQE or FBQE with bad pointer	-	Next FQE pointer is incorrect (if SDA or LSQA, then previous FQE length could be too large) - FQE queue is truncated (by setting the bad pointer to zero).
F3	4	FQE	-	Incorrect (too long) length in FQE - FQE queue is truncated.

Figure 5-22. SDWAVRA Error Indicators

VSM Cell Pool Management

VSM's cell pool management consists of the following functions:

Module	Macro	Function
IEAVGTCL	GETCELL	Gets a cell from a preformatted pool of cells
IEAVFRCL	FREECELL	Frees a cell to a preformatted pool of cells
IEAVBLDP	BLDCPOOL	Builds a pool of formatted cells
IEAVDELP	DELCPOOL	Deletes a pool of formatted cells

The key to the VSM cell pool management function is the cell pool anchor block (CPAB). The layout of the cell pools is shown in Figure 5-23. Note that the permanent cell pools have IDs that are negative, while the dynamic cell pools have IDs that are the address of the first CPAB divided by 4 (shift right 2).

The four VSM cell pool management modules are small enough that processing can be determined from studying the CPAB mapping along with the code.

Miscellaneous Debugging Hints

1. Most common problems with GETMAIN/FREEMAIN occur in the interface processing. There is usually a bad TCB or ASCB address or the local lock is not held upon entry. The ASCB is used only to find the LDA which is in the same location in all address spaces except the master scheduler's.

   Note: On a branch entry to GETMAIN, register 7 contains the address of the ASCB; however, on return from the branch entry, register 7 no longer contains this address.

2. A valid GETMAIN/FREEMAIN that is issued for zero bytes is treated as a no-op.

3. Good places for GETMAIN/FREEMAIN traps are:

   - Routine CSPCHK in IEAVGM00.

   - Routines called from branch tables SPBRTBLG and SPBRTBLF in IEAVGM03.

4. GETMAIN makes few queue manipulation errors. If GETMAIN/FREEMAIN rejects a request, it is usually because the caller made an error on the interface; the message IEA700I is issued.

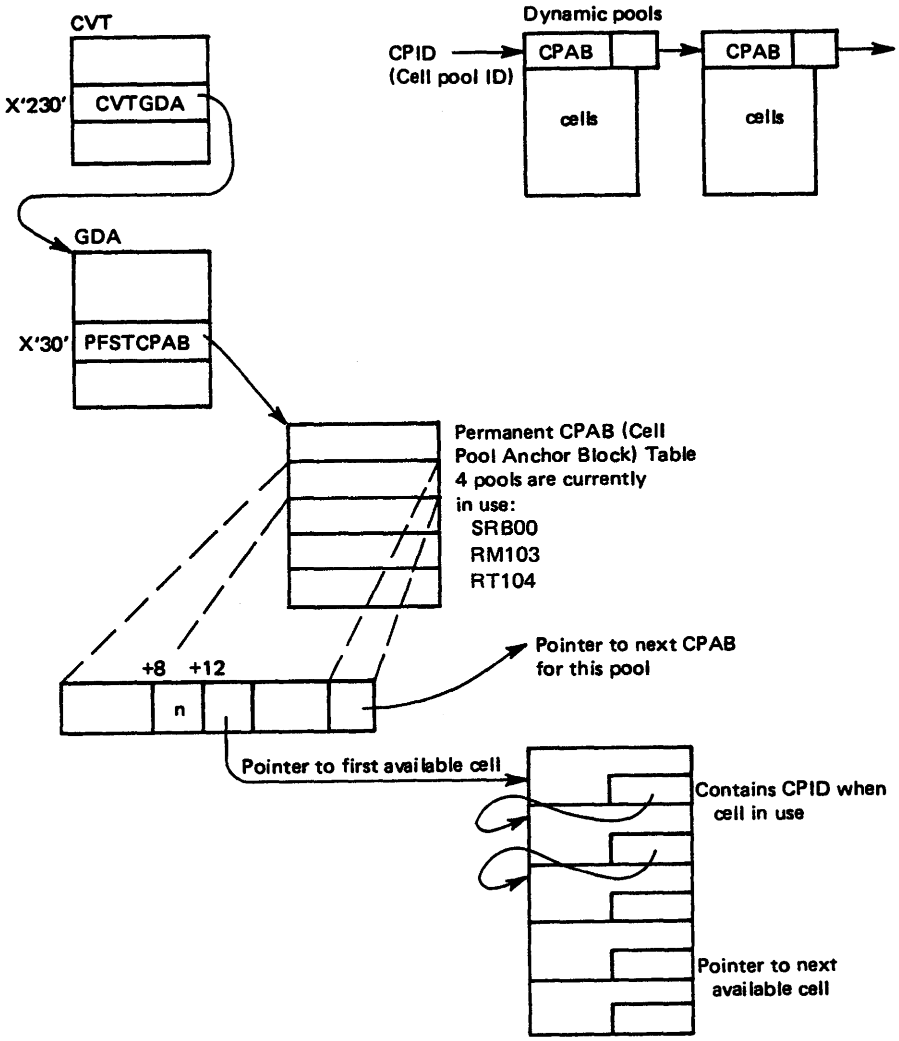


Figure 5-23. VSM Cell Pool Management

5. Subpool queue elements (SPQEs) are not freed even on a subpool FREEMAIN, and multiple keys for a problem program subpool on the same TCB are not allowed. This can result in a problem if a user changes TCBPFK. The following is an example of such a situation:

Set TCB key	TCBPFK=6
	GETMAIN SP 1 Causes SPQE to be built, storage in key 6
	FREEMAIN SP 1 SPQE for SP 1 is not destroyed
Change TCB key	TCBPFK=8
	GETMAIN SP 1 IEAVGM00 uses old SPQE and assigns storage in key 6

6. On branch entry to GETMAIN, registers are saved at field BRANCHSV in the LDA and turns on the BRENTRY flag at offset X'10' in LDA. To be sure these saved registers are for the request in question, it is necessary to stop the system via a trap. (See "Using the SLIP Command" and the "System Stop Routine" topics in the chapter "Additional Data Gathering Techniques" in Section 2.)

7. MVS has added a new register type GETMAIN/FREEMAIN SVC and branch entry. It is SVC 120. The parameters differ from those of SVC 10 as follows:

Register 1 -	Zero for a GETMAIN; address to be freed for FREEMAIN.			
Register 15 (SVC only) -	Bytes 0 and 1: Reserved, set to 0.			
	Byte 2:	Subpool ID		
	Byte 3:	Following flag values:		
		Bits 0-4	Reserved, set to 0	
		Bit 5		
			=0	Double word boundary
			=1	Page boundary
		Bit 6		
			=0	Conditional request
			=1	Unconditional request
		Bit 7		
			=0	GETMAIN
			=1	FREEMAIN

For the branch entry SVC 120, register 15 contains the entry point address and register 3 is used for the parameters. Register 3 is set up the same as register 15 above with one exception: Byte 1 is the protect key for subpools 227-231 and subpool 241. The address that was obtained via GETMAIN is returned in register 1 as in SVC 10.

8. Some GETMAIN failures are recorded in an information list contained in the nucleus. This list is similar to a trace table and is pointed to by the CVTQMSG (CVT+X'10C'). Each entry contains data such as: ABEND code, ASCB address, TCB address, register 14 if GETMAIN was branch entered, and the parameters passed. Refer to the DSECT INFOLIST in module IEAVGM00 for additional information.

GMFMWKAR (IN LDA AT +X'58')

OFFSET
IN HEX (FROM START OF LDA)

OFFSET				
58	ABNDATA (VAR. DATA)			
5C	MSGLEN	FREESW	LOCKFLAG	FRRBRNDX
60	REGSAVE SAVE AREA USED FOR SRM AND RSM (18 FULL WORDS)			
A8	MSAVE SAVE AREA USED FOR MRELEASE (16 FULL WORDS)			
E8	GNOTSAVE SAVE AREA USED FOR GNOTSAT (16 FULL WORDS)			
128	LOCKSAVE (OVERLAPS INTO GFSMFSVE AND MPTRS)			
130	GFSMFSVE/CSPCKSAV (SMF CORE ROUTINES SAVE AREA/ CSPCHK SAVE AREA)			
13C	MPTRS (PREVIOUS AND NEXT PTRS SAVE AREA)			
144	DUMYDQE (DUMMY DQE FOR L/SQA EXPANSION) (4 FULL WORDS)			
154	TEMPDQE (TEMPORARY DQE FOR FMCOMMUN) (4 FULL WORDS)			
164	DUMFBQE (DUMMY FBQE FOR MRCLEASE) (4 FULL WORDS)			
174	SAV911 SAVE AREA FOR REGS 9-11 (BRANCH ENTRY)			
180	LASTSAVE (LAST LIST ENTRY)			
184	MINMAX MAX & MIN LENGTH FOR VARIABLE REQUEST			
18C	LASTLSTA (LAST LIST ENTRY ADDRESS)			
190	LSTINDEX (INDEX FOR LIST REQUEST)			
194	FMARCAS (PTR TO AREAS TO BE FREEMAINED)			

MSGLEN – REASON CODE AND LENGTH OF VAR. DATA

FREESW

X'80'	FREEMAIN IN PROGRESS
X'40'	LENGTH HAS BEEN INCREMENTED
X'20'	ADDRESS HAS BEEN DECREMENTED
X'10'	NOT 1ST DQE (FOR L/SQA)
X'08'	FQE WAS BELOW FREED AREA
X'04'	FURTHER PAGE RELEASE NEEDED

LOCKFLAG

X'02'	SALLOC LOCK OBTAINED
X'01'	SALLOC LOCK ALREADY HELD

FRRBRNDX

X'07'	SUBPOOL FREEMAIN, AQE AREA NOT IN DQE
X'06'	PAGE RELEASE RETURN CODE OF 1
X'05'	SALLOC OBTAIN RETURN CODE NOT 0 OR 4
X'04'	ON L/SQA EXPANSION, GFRECORE FAILED
X'03'	FINDPAGE RETURN CODE NOT 0 OR 4
X'02'	CREATE SEGMENT RETURN CODE>0
X'01'	SALLOC RELEASE RETURN CODE>0
X'00'	UNEXPECTED ERROR, SEE STATUS

OFFSET
IN HEX (FROM START OF LDA)

198	PARMLDA	FRRPARM (FRR PARM AREA ADD)		
19C	CLOPREV (PREVIOUS FQE TO CLOSEST)			
1A0	CLOSEST (CLOSEST INSIZE FQE)			
1A4	LARGESTA (LARGEST AVAILABLE)			
1A8	LARGEST (LARGEST AVAILABLE FBQE)			
1AC	LENSAVE (SAVE AREA FOR LENGTH LIST PTR)			
1B0	SAVESIZE (SIZE OF MULTIPLE OF 4K CORE)			
1B4	ENDADD (END ADDRESS)			
1B8	STRTADD (START ADDRESS)			
1BC	DIFF/SAVEPQE (DIFFERENCE/PQE PTR IN FBQESPCH)			
1C0	FIXSTART (STARTING ADDRESS TO CLEAR)			
1C4	FIXEND (ENDING ADDRESS TO CLEAR)			
1C8	NOTSATSV (LEN PTR USED BY GNOTSAT)			
1CC	NOTSATSI (LDARQSTA SAVE AREA FOR GNOTSAT)			
1D0	SAVESEG (ADDRESS OF MULTIPLE OF 4K CORE)			
1D4	LARSOFAR (LARGEST AVAILABLE IN FBQE)			
1D8	RSTRTADD (ROUNDED START ADDRESS)			
1DC	RENDADD (ROUNDED END ADDRESS)			
1E0	VPFP (FIND PAGE ADDRESS TO BE USED)			
1E4	DQESAVE SAVE DQE PTR AND PREVIOUS DQE PTR			
1EC	FMSAVE (SAVE RETURN REG FOR FREEMAIN)			
1F0	PREVFQSV (SAVE AREA FOR PREVIOUS FQE PTR)			
1F4	FQESAVE (SAVE AREA FOR FQE)			
1F8	SPQESAVE (SAVE AREA FOR SPQE)			
1FC	SVRLB (SAVE AREA FOR RLB)			
200	SVSIZE (SAVE AREA FOR ROUNDED SIZE)			
204	DQESAVE1 SAVE DQE INFO WHEN USING FMAINB			
20C	FMSAVE1 (SAVE RETURN REG IN FMAINB)			
210	FQESAVE1 (SAVE FQE INFO IN FMAINB)			
214	PREVFQS1 (SAVE PREVIOUS FQE IN FMAINB)			
21B	SPQSAVE1 (SAVE SPQE IN FMAINB)			
21C	SVRLB1 (SAVE RLB FOR FMAINB)			
220	SVSIZE1 (SAVE ROUNDED SIZE FOR FMAINB)			
224	SAVSVTSV (SAVE LDARQSTA IN FMAINB)			
228	FQENXTSV (FQE NEXT SAVE AREA)			
22C	OLDFQELN (OLD FQE LENGTH)			
230	NEWFQELN (NEW FQE LENGTH)			
234	RQSTSIZE (SIZE OF ORIGINAL REQUEST)			
238	SEGTEST (END SEG TEST AREA)		CODE1	CLEARSW
23C	GMFMSW	FETCH	OUTSW	CODE2
240	SAVFRESW	SPID	LSPQEKEY	RQSTRKEY
244	SAVSPID	SAVSPID2		

PARMLDA
X'80' GLOBAL REQUEST (GLBRANCH OR MRELEASE ON GLOBAL REQUEST)
X'40' SALLOC LOCK OBTAINED BY GM/FM
X'04' FIRST FLAG BYTE IN FRR PARM
X'00' LDA ADDRESS IN FRR PARM

CODE1 – (SAVE AREA FOR OPTION CODE)
X'C0' LIST INDICATOR (MIXED IF LIST)
X'20' CONDITIONAL INDICATOR
X'10' MASK FOR PAGE BOUNDARY
X'04' SVC 120 PAGE BOUNDARY REQUEST
X'02' SVC 120 UNCONDITIONAL REQUEST
X'01' SVC 120 FREEMAIN REQUEST

CLEARSW – (CLEARSW FOR GFRECORE)
X'01' FQECPB INDICATOR ON IN FQE

GMFMSW – (GM/FM SWITCH FOR MRELEASE)
X'04' FIRST TIME SW FOR MRELEASE
X'02' INDICATES FM FOR FBQE
X'01' INDICATES GM FOR FBQE

FETCH – (KEY AND FETCH PROTECT)
X'08' FETCH PROTECT ON

OUTSW – (SWITCH FOR OUT OF REAL/VIRT.)
X'00' REAL INDICATOR FOR OUTSW
X'FF' VIRT. INDICATOR FOR OUTSW

CODE2 – (SAVE AREA FOR OPTION CODE)

SAVFRESW – (SAVE FREESW IN FMAINB)

SPID – (SPID FOR MRELEASE)

LSPQEKEY – (PROTECT KEY FROM CURRENT SPQE)

RQSTRKEY – (REQUESTER KEY OR KEY = PARM)

SAVSPID – (SAVE SPID FOR FREEMAIN)

SAVSPID2 – (SPID FOR MESSAGES)

Real Storage Manager (RSM)

The real storage manager (RSM) manages the real storage of the system. To do this, it divides all potentially pageable real storage into 4K-byte frames. Within RSM, the page frame table entry (PFTE) describes the frame according to type, current use, or its most recent use.

The current or last state of a request for RSM pageable services is described by the page control block (PCB) within RSM: the requestor supplies information about his request and RSM reformats this data into a PCB. As the request is processed, RSM adds other internal RSM information to the PCB.

RSM is a queue-driven component. Both PFTEs and PCBs are queued based on their current state. Simply stated, frames that can be used immediately are queued on the available frame queue; their PFTEs describe the frame's last use. Similarly, free request elements are queued on the FIFO PCB free queue; these PCBs describe the final state of previously processed requests. (This historical nature of PCBs is often useful in problem analysis.) To manipulate these control blocks and manage the queues, RSM has a PFTE manager (IEAVPFTE) and a PCB manager (IEAVPCB). Besides being queued, PFTEs are located in a contiguous table starting at (PVTPFTP) + (PVTFPFN) multiplied by 16) and ending at (PVTPFTP) + (PVTLPFN multiplied by 16). PCBs, however, are obtained (via GETMAIN) in groups and are spread out in SQA. They can be found only by following queue pointers.

Major RSM Control Blocks

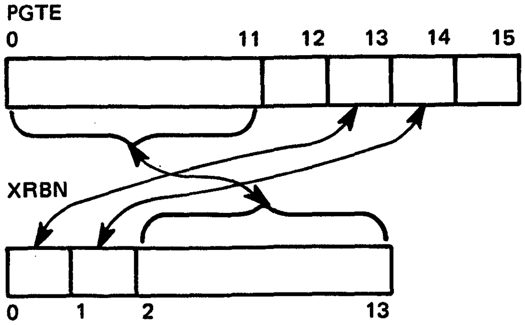
RSM's major control blocks are the PFTE, PCB, page table entry (PGTE), external page table entry (XPTE), paging vector table (PVT), RSM header (RSMHDR), and swap control table (SPCT). An RSM service routine called find page (IEAVFP) locates the PGTE and XPTE control blocks. The table in Figure 5-24 lists the control block functions.

Control Block	Function
PFTE	describes the last use of a frame
PCB	describes the current or last state of a request
PGTE XPTE	describes the current real frame and virtual page relationship for a particular virtual address
PVT	basically these are RSM anchors and work areas
RSMHDR	header information
SPCT	related only to swapping, it describes the RSM requirements necessary to swap in an address space (the swap out process formats the SPCT)

Figure 5-24. Major RSM Control Blocks and Their Functions

Only the leftmost 14 bits of a real address (XRBN) or the leftmost 12 bits of a virtual address (VBN) are needed to describe a specific real frame or virtual page (a modulo 4K-bytes real and virtual addressing scheme). Also, note the following:

- PGTEs contain XRBN values in a rearranged format: bits 0 and 1 of the 14-bit XRBN are in bits 13 and 14 of the 16-bit PGTE, and bits 2 through 13 of the XRBN are in bits 0 through 11 of the PGTE.



- The contents of PVTPFTP plus XRBN0 is the address of the PFTE for the frame whose real address is XRBN000 through XRBNFFF.

Of all the RSM control blocks, the most critical are the PCB, PFTE and SPCT. The important fields in each block are described below. Figure 5-25 shows the relationship among the blocks.

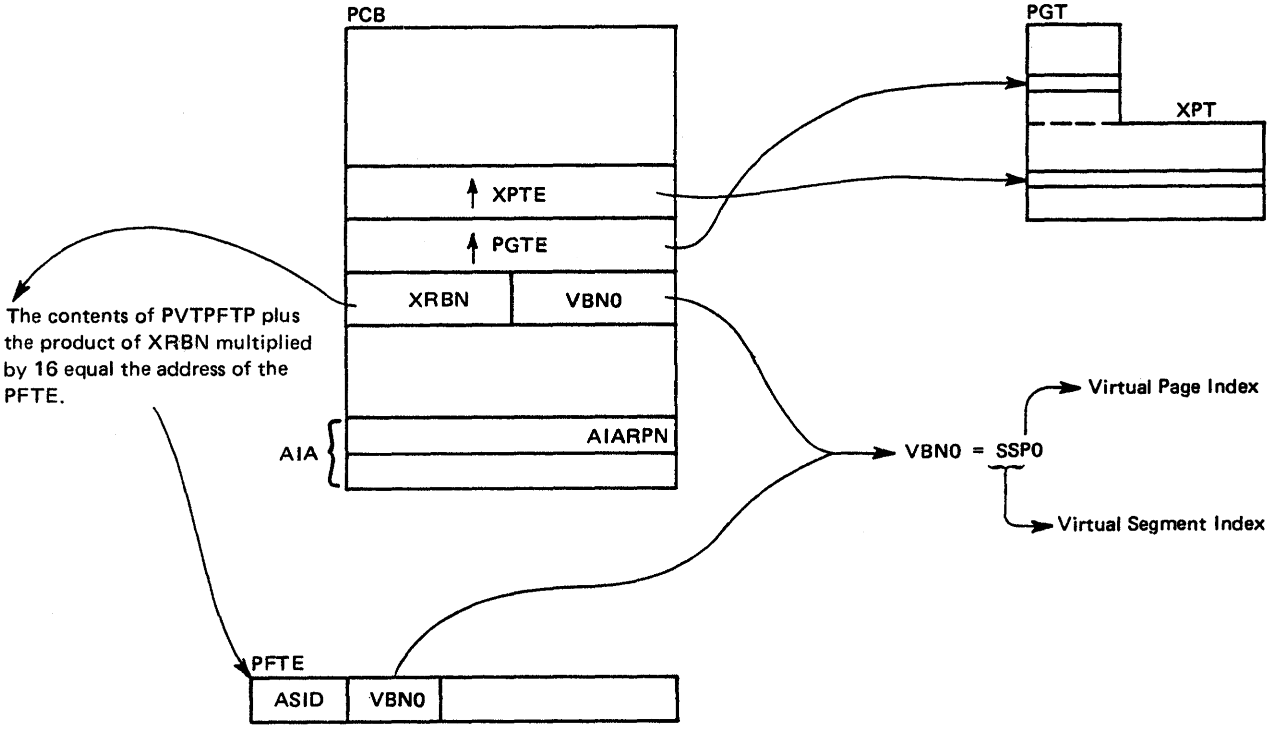


Figure 5-25. Relationship of Critical RSM Control Blocks

PCB (Page Control Block)

Important fields in the PCB are:

+0 PCBCQN - indicates the current queue location of this PCB as follows:

X'10' - PCB is not currently in use. It is queued on the PCB free queue anchored in the PVT.

X'18' - PCB is currently waiting for frame allocation to occur. It is queued on the PCB defer queue anchored in the PVT.

X'20' - PCB represents a common area I/O operation. Actual physical I/O may or may not be complete. It is queued on the PCB common-I/O queue anchored in the PVT.

X'88' - PCB represents a private area I/O operation. Actual physical I/O may or may not be complete. It is queued on the PCB local-I/O queue anchored in the RSMHDR for the address space indicated by PCBASCB; ASCBRSM points to the RSMHDR.

X'FF' - PCB is probably in use. The not-queued state means only that the PCB is not on the primary forward/backward chain of the above mentioned major PCB queues. It can be a related PCB, a root PCB, or an associated PCB.

+8 PCBFL1:

PCBSRBMD=X'20' - PCBSRB is the address of a page-fault-suspend SSRB. The use of this address is the only means of locating page-fault-suspended SRBs.

PCBROOT=X'04' - PCBRTCA is the address of a root PCB. Root PCBs are only valid if their PCBCQN field is X'FF'.

+9 PCBRTPA - When the PCBROOT bit is on, this contains the address of a PCB that controls a block page operation.

+X'D' PCBRLPA - The address of a chain of PCBs for the same PCBVBN/PCBRBN. The related chain of PCBs are dequeued PCBs that are chained via the PCBRLPA field (not via PCBFQP/PCBBQP).

+X'10' PCBFL2:

PCBRESET=X'10' - The function indicated by the PCB has been suspended for a page fault because no frames were available or paging I/O had to be completed before redispatching the page faulter. PCBASCB, PCBRTPA, and PCBSRB define the ASCB, TCB, and RB to be RESET when PCBSRBMD is 0. When PCBSRBMD is 1, PCBASCB and PCBSRB define the ASCB and SSRB that will be RESET.

+X'11' PCBXPTA - Is either 0 or the address of the XPTE.

+X'15' PCBPGTA - Is either 0 or the address of the PGTE.

+X'18' PCBXRBN - This value when multiplied by 16 and added to the address in PVTPFTP gives the address of the associated PFTE.

+X'1A' PCBVBN - This field is often zero; when it is zero, the operation has either been NOPed with page I/O still in progress or the I/O is complete and the PCB is only serving a scheduling/tracking function. The operation is considered to be complete when PCBVBN=0; no other paging request should be able to relate to it; that is, it cannot be found via an equal compare on PCBVBN. When PCBVBN is zero, its previous value can be determined from the AIARPN field in the AIA. The AIA is the last 28 bytes of the PCB.

The following information about roots is useful to the problem solver.

- Root PCBs can generally be recognized because most of the PCB is still zero.
- The SPCT points to active roots for SWAP; RSMSPCT in the RSMHDR points to the SPCT.
- V=R waits for region roots are queued from PVTVROOT in the PVT.
- Vary offline roots are queued from PVTOROOT in the PVT.
- PAGE FIX and PAGE LOAD roots can only be found via PCBRTPA of the queued FIX/LOAD PCBs.

For non-root PCBs: PCBCQN, PCBFL1, PCBFL2, and PCBFL3 are the key fields. They describe the current state of the paging request for which the non-root PCB was last used.

See the topic "PCB Trace Facility" later in this section for a description of PCB tracing.

SPCT (Swap Control Table)

The SPCT is mapped in modules IEAVSOUT, IEAVSWIN, IEAVCSEG, and IEAVITAS. Space for the SPCT is obtained via GETMAIN and is initialized in IEAVITAS. As segments are created, IEAVCSEG updates the SPCT. IEAVSOUT initializes the SPCT with the pages that make up the working set (such as, LSQA and fixed pages plus recently referenced pages). IEAVSWIN uses the information IEAVSOUT put in the SPCT in order to start up a previously swapped-out address space. Note that a one-stage swap permits pageable working set pages to be written to the swap data set along with the LSQA pages. This improves response time for swappable address spaces (such as TSO).

The first portion of the SPCT contains the address of the swap root PCB (SPCTSWRT); the number of fixed, LSQA and pageable page entries in this SPCT (SPCTFIX, SPCTLSQA and SPCTS1PE); the number of segment entries and the number of active segment entries (SPCTNSEG and SPCTSSEG); and the working set size (SPCTWSSZ). The flags at offset X'A' are defined as follows:

X'80'	Swap-in in progress
X'40'	Swap-out in progress
X'20'	Paging was purged during swap-out
X'10'	There is at least one fix entry with a fix count greater than 255
X'08'	Page data set used for LSQA
X'04'	Swap-out requested by IEAVEQRP

The next portion of the SPCT (SPCTSWAP) is the SPCT extension and is 56 (decimal) bytes long. It contains a maximum of six fix swap entries or eight LSQA swap entries, or a combination of the two. In a combination, LSQA entries precede all fix entries. LSQA entries are six bytes each and fix entries are eight bytes each. Both entries contain the following flags in the first byte:

X'80' LSID in this entry is valid.

X'40' This is an LSQA or pageable page entry.

X'20' This SPCT entry is for a pageable page that should be backed below 16 Mb, if possible, because it was previously fixed and will probably be fixed again later.

X'10' The page was flagged defer release at swap time.

X'08' This SPCT entry is a pageable page. Note that X'48' indicates a pageable page and X'40' indicates an LSQA page.

X'04' This SPCT entry indicates a pageable page that was changed during the swap-in interval. On swap-in the change bit must be turned on in the protect key because a valid auxiliary copy does not exist.

X'02' This SPCT entry is ignored.

This flag byte is followed by a three-byte LSID and a two-byte VBN. If the entry is for LSQA or pageable page, there is nothing more, but if the entry is a fix entry, the next two bytes contain the fix count. The last portion of the SPCT contains a variable number of six-byte segment entries. The first byte is the segment number and it is followed by the address of the page table. The next two-byte field (SPCTBITM) is a 16-bit map indicating which pages are to be brought in at swap-in time as two-stage pageable working set pages.

PFTE (Page Frame Table Entry)

Important fields in the PFTE are:

PFTIRRG - indicates the format of the first word of the PFTE. This bit is located in PFTFLAG2 at offset X'D' and is a X'10'. If it is on, the first word of the PFTE is mapped as PFTPGID and contains a VIO LGN and RPN. If PFTIRRG is off, the first word of the PFTE is mapped as PFTASID and PFTVBN. An ASID of X'FFFF' indicates a common area page. Note that a VIO LGN can be the same as an address space ASID; address space ASIDs and LGNs are seldom the same but could be.

PFTPCBSI - indicates there is a PCB on an I/O queue for this page; there can be a string of related PCBs for this page. This bit is located in PFTFLAG1 at offset X'C' and is a X'08'. This bit is turned off by the process that validates the page when the I/O completes, or, for output I/O, after the I/O completes but before the PFTE is sent to the free queue. Note that I/O queues sometimes contain several "no-op" PCBs; these appear to point back to a frame and its associated PFTE. When a PCB is made into a "no-op," PFTPCBSI is turned off and the association between that PCB and that frame and its associated PFTE is broken. These "no-op" PCBs are either output PCBs with incomplete I/O or input PCBs with complete I/O.

PFTSASF - indicates whether RSM or PFA (page fault assist) allocated the frame. If this bit is on, the frame was allocated by RSM. If this bit is off, the frame was allocated by PFA.

Page Stealing

Figure 5-26 shows the flow of the page stealing process. The circled numbers in the figure correspond to the notes below.

1 If the frame queue is for a private address space that is not the current address space, IEAVRFR issues CMSET to the private address space.

2 IEAVRFR scans the local frame queue (LFQ) or common frame queue (CFQ); the queue it scans is determined by the parameter list received from SRM.

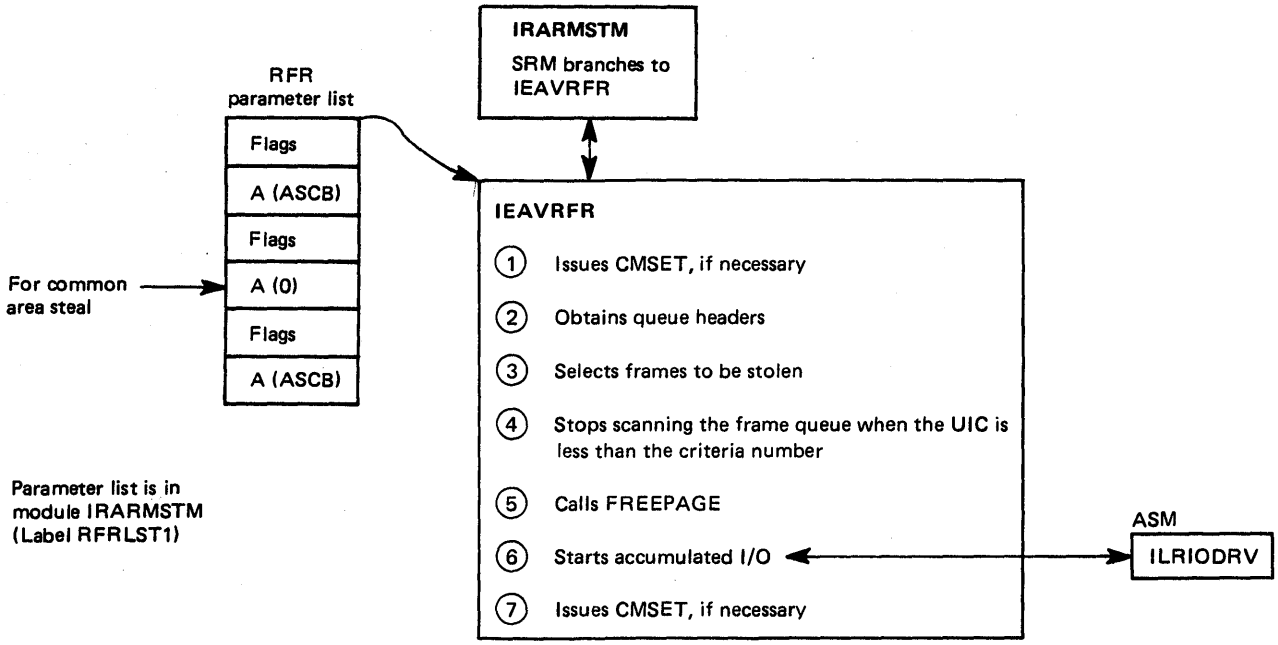
3 IEAVRFR checks the hardware reference bits and then updates the unreferenced interval count (UIC). IEAVRFR orders the LFQ and the CFQ so that the PFTEs with the highest UICs are at the top of the queue. The queues are in descending order, with zero UICs at the bottom.

4 Frames are selected to be stolen based on their UIC and pageability; that is, fixed/LSQA/bad pages, and pages that are V = R allocated cannot be stolen.

5 IEAVRFR calls a common routine, FREEPAGE, to invalidate selected pages and free a frame (if a page is unchanged), or to build a PCB for the page-out process (if the page is changed).

6 When the scanning of the queue is completed, IEAVRFR calls ASM to start output paging if any PCBs have been accumulated.

7 IEAVRFR issues CMSET to return to the original address space, if necessary.



IRARMSTM
SRM branches to IEAVRFR
RFR parameter list
Flags
A (ASCB)
Flags
A (0)
Flags
A (ASCB)
For common area steal
Parameter list is in module IRARMSTM (Label RFRLST1)
IEAVRFR
1 Issues CMSET, if necessary
2 Obtains queue headers
3 Selects frames to be stolen
4 Stops scanning the frame queue when the UIC is less than the criteria number
5 Calls FREEPAGE
6 Starts accumulated I/O
7 Issues CMSET, if necessary
ASM
ILRIODRV

Figure 5-26. Page Stealing Process Flow

Reclaim

Reclaim is an RSM function that revalidates a page/real frame pair that was previously invalidated. IEAVGFA performs the reclaim for the normal case after a page fault on an address space or common area virtual address. IEAVAMSI handles the VIO case.

In the virtual address case, IEAVGFA handles work as follows:

1. PCBVBN is used to locate the PGTE.
2. The PGTE is used to obtain the last-used XRBN value.
3. The XRBN is used to address the PFTE.
4. PFTIRRG is checked to determine if the first word of the PFTE is in PFTPGID or PFTASID/PFTVBN format.
5. If PFTIRRG = 0, PFTVBN is compared to PCBVBN.

6. If the VBNs match and the VBN is in the common area, the reclaim is successful. If the VBN is in the private area and PFTASID matches ASCBASID (which PCBASCB points to), the private area reclaim is successful.

In the VIO case, IEAVAMSI handles work as follows:

1. IEAVAMSI is supplied with both a XRBN and a DSPID.

2. The XRBN is used to address the PFTE.

3. PFTIRRG is checked to determine if PFTPGID is in PGID format.

4. If PFTIRRG = 1, PFTPGID must match the DSPID; if it matches, the reclaim is successful.

When reclaim fails, normal frame allocation paths are followed just as though the page had never been in storage.

Relate

Relate is an RSM function that associates independently-generated page requests (PCBs) for the same virtual address. When the physical action required to satisfy one of these requests (I/O or frame allocation) is completed, all related requests are also satisfied. A PCB-related chain is produced for all cases except the VIO data set. The same modules that do *reclaim,* IEAVGFA and IEAVAMSI, handle the relate process, which only follows after a successful reclaim.

In the virtual address case, IEAVGFA handles work as follows:

1. The relate function is invoked for one of two cases:

   - The reclaim function has successfully completed and PFTPCBSI is on, indicating page I/O is in progress; a PCB I/O queue is searched.

   - The XPTDEFER bit is on, indicating that the previous PCBs have been delayed because frames were not available. The GFA defer queue will be searched to do the relate function.

2. The search argument is PCBVBN in all cases except that of the GFA defer queue; in that case PCBASID and PCBVBN are the search arguments.

3. When the correct queued PCB is located, the current PCB is added to the related PCB chain. For the defer case, the PCB is added to the head of the chain; for the I/O case, the PCB is added to the end of the chain.

In the VIO data set case, IEAVAMSI handles work as follows:

1. The PCB local I/O queue is scanned for a match on PCBRBN because PCBVBN is always set to 0 for move-out PCBs. If PCBRBN matches, PCBVAM must be on.

2. When the correct PCB is found, it is updated with the information the I/O completion portion of RSM needs to place the page of the VIO data set in the new window location (this is not necessarily a new page).

RSM Recovery

RSM recovery consists of a SETFRR at all major entry points to the RSM:

- The issuer of the SETFRR places the address of the FRR in PVTPRCA.
- Each SETFRR returns a six-word parameter list in the recovery communications area (RCA).
- RSM has only one FRR - IEAVRCV.
- The IHARCA macro maps the RCA; this macro can be found in most RSM modules.
- IEAVPSI contains the RCA macro in assembler language format.

Whenever an unexpected error or C0D abend occurs, the RCA is copied into SDWAVRA. The CSECT ID and the module-entered flag in the RCA can be used to determine the path taken through RSM to the point of error. To determine this path, you must understand the RSM flow and know which module issues SETFRR. The following RSM modules issue SETFRR at their main entry point:

IEAVAMSI
IEAVCSEG
IEAVEQR
IEAVIOCP
IEAVITAS
IEAVPIOI

IEAVPIOP
IEAVPIX
IEAVPRSP
IEAVPSI
IEAVRCF
IEAVRCF3
IEAVRCV
IEAVRFR

IEAVSOUT
IEAVSQA
IEAVSWIN
IEAVTERM
IEAVRELS at IEAVRELV (entry point)
IEAVFRSB at IEAVPRSR (entry point)
IEAVSWPP

RSM's FRR does not attempt complex recovery. Its main objective is to record the error and issue an SDUMP. It has some special logic based on where the error occurred, as follows:

Error Occurred In	FRR Action
IEAVEQRI, IEAVRCFI, or IEAVRF3I	Restore registers for return to IEAVPFTE.
IEAVPIX	Attempt to reset page faulter.
IEAVSIRT	"Memterm" swapping in address space.
IEAVSWIN	"Memterm" swapping in address space.
IEAVPIOI	Retry for cleanup or "Memterm."
IEAVINV	Set "GO" indicator and PTLB or retry to PTLB.
IEAVPSI	If error occurred while checking input parameters, set abend of 171.

	If error occurred in IEAVPSI at entry point IEAVPSIX, IEAVPSJY, or IEAVPSIF, ensure that if the SALLOC lock was held, it is released prior to percolation.
Other	If it is a non-zero retry address, retry; otherwise continue with termination.

Recursion is not allowed. The PVT and PFTEs are dumped on the SDUMP.

The following reason codes are put into the RCARCRD field when real storage management issues abend with a code of C0D. All C0D abends are retried at the next sequential instruction.

Real Storage Management ABEND Reason Codes

Code (hex)	Meaning
01	Findpage, translate real to virtual, or the LRA instruction returned an unexpected code for a segment, page, or frame whose existence was implied by some RSM control block or function. Findpage, translate real to virtual, or LRA is assumed to be correct.
02	A GETCELL or FREECELL for the RSM cell pool failed. If FREECELL, the error is ignored; if GETCELL, asynchronous retry is attempted where possible.
03	A FREEMAIN failed for space originally obtained by RSM or VSM using GETMAIN. The error is ignored.
04	The return code from ASM (ILRSWAP, ILRIODRV, or ILRTRPAG) indicates an invalid request. The recovery action taken by RSM varies with the type of request, but the RSM function being performed is usually terminated if ASM resources were being requested, or continued if ASM resources were being returned.
05	A GETMAIN for RSM control block space was unsuccessful. The function for which the space was required is terminated.
06	An attempt was made to release a lock which was not held. RSM tables might be damaged due to the loss of serialization. RSM attempts to continue normal operation.
07	RSM control information indicated a PCB for a page should exist on an I/O active queue or on the defer queue, but searching of the queue(s) failed to find the PCB. It is assumed the control information is in error and no such PCB exists.
08	The existence of a V = R or offline root PCB was implied but no appropriate PCB could be found on the V = R or offline root queue. The error is ignored and indicators are reset.
09	Swap-in's XMPOST error exit was entered, so restore will not run. The target address space is terminated.
0A	An incorrect fix count was detected in a PFTE. The count is adjusted to the expected value.
0B	The interprocessor communications service routine (RISIGNL) could not signal another processor as requested by IEAVINV. The condition is ignored and normal operation continues.
0C	IEAVPIOP has discovered an undefined combination of I/O completion status flags in the AIA after a page-in or page-out. The condition is treated as an I/O error.
0D	IEAVDSEG was requested to destroy a non-existent or common area segment. The request is denied.
0E	A PCB was required but none were available. The routine needing a PCB is terminated.
0F	The attempt to complete processing of a previously deferred FREEMAIN release has failed.

10 An FOE could not be found on the specified TCB's fix ownership list.

11 An internal RSM invocation of the PGOUT function was unsuccessful. The page remains in real storage.

12 A swap (in or out) was requested for an address that already has a swap in progress, or no SPCT exists for the address space to be swapped. The request is denied.

13 Swap-in could not re-establish the address space due to missing or incorrect control information (SPCT or PCBs). The address space is abnormally terminated.

14 An internal invocation of PGFREE failed. The error is ignored.

15 Swap-out has detected an inconsistency in the SPCT fix entries it has created. The error is suppressed and recovery attempted.

16 ASCBCHAP could not enqueue or dequeue an ASCB during a swap-in or swap-out operation. The address space is terminated.

17 Swap-out has detected an error in the allocated frame count (ASCBFMCT) for the address space. If possible, the count is corrected and the swap-out continued; otherwise, the swap-out is suppressed.

18 No SPCT segment entry could be found for a segment whose existence was implied by other RSM control information. The error is ignored and the SPCT update is skipped.

19 An internal RSM function issued a return code which was either invalid or not applicable. System action depends on the nature of the function.

1A Swap-in detected a common area page that was not obtained using GETMAIN among the input working set. The page is not made available to the incoming address space. Some other address space must have freed the page using FREEMAIN while the current one was swapped-out. Probable user error.

1B During an attempt to free the frames backing a V = R region, it has been determined that the virtual space is not backed by real storage, or that the virtual-to-real mapping is not 1 to 1.

1C IEAVPSI attempted to fix the ECB for a page service that will complete asynchronously, but IEAVFXLD returned a code indicating the fix was not accomplished.

1D A PCB marked I/O complete (indicating that it was previously processed by IEAVPIOP) has been passed to IEAVPIOP by ASM.

1E A software error has been found in the AIA passed from ASM to RSM for an I/O request. Possible errors are:

- The AIA contains data inconsistent with previous AIAs.
- The original input chain (to ASM) was invalid.
- The LSID in the XPTE was invalid.
- The LPID in the XPTE was invalid.
- A hardware I/O error occurred on a pageout PCB (this should not occur).

1F An invalid real storage address was returned to IEAVPRSB at entry point IEAVPRSR.

20 SWAP-out has encountered a bad frame. The page residing in that frame will not be paged out. If the page is a private area page, it will be "locked" into storage for the duration of the swap. During this time, V = R allocation and vary offline of storage could be affected. If the page is an LSQA page, a DONTSWAP sysevent is issued.

21 IEAVPFTE detected a discrepancy in the SQA reserve queue count controls. Use of the SQA reserve queue is discontinued until after re-IPL. RSM attempts to continue normal operation.

22 IEAVTERM has found an FOE fix count that is greater than the fix count in the corresponding PFTE. The PFTE fix count is not changed, but the FOE is freed.

23 Entry point IEAVREP1 in module IEAVPCB was passed a nonzero return code from IEAVBLDP, which indicates the CPAB for RSM is bad.

25 During IEAVSOUT processing, the search for a PCB on a related chain failed.

26 During IEAVRCF processing, using input from PFTASID, the LOCASCB failed.

27 In IEAVAMSI, the ASGNLOAD subroutine found fewer UCBs requiring page-ahead PCBs than the mainline routine in IEAVAMSI had counted.

28 In IEAVSOUT, a pageable changed page was valid in storage at the time an SPCT entry was built for it. However, the page was found invalid when an LRA instruction was done while building a PCB for it.

29 An undefined function code was passed to IEAVPRSB at entry point IEAVPRSS.

2A In IEAVPREF, an attempt to issue message "IEA988I PREFERRED AREA EXPANDED. RECONFIGUREABILITY MAYBE IMPAIRED" failed.

2B During page fault assist (PFA) processing, the microcode encountered a program check. Code X'2B' is issued by IEAVPIX upon receiving control from the program check FLIH after PFA processing issued a X'26' program interruption.

RSM Debugging Tips

1. Because the PCB free queue is a FIFO queue, it represents recent history in RSM. Start your search of the PCB free queue with the youngest PCB (PVTFPCBL) and look for the appropriate VBN in the PCBVBN or AIARPN. This approach often reveals what has most recently happened to the page in question.

2. Whenever the system wants to break the logical connection between the PCB and the page, it sets PCBVBN to zero. Therefore, look at AIARPN to determine what VBN the PCB was associated with (AIARPN=PCBVBN/16).

3. The PVT contains several work/save areas that belong to a unique module. These are often useful to determine the last thing a module did.

4. At any time, there should never be more than one input PCB with a given PCBVBN on the I/O-active or GFA-deferred PCB queues. Output PCBs are never in a related position. Although, an output PCB for a nondisconnect VIO moveout might have input PCBs related to it.

5. The XPTVIOLP flag can be confusing. If it is on, XPTXAV must be on. XPTVIOLP=1 indicates that there is an LPID in the XPT, not an LSID.

6. It is sometimes useful to observe the AIANXAIA pointers in PCBs on the PCB free queue. These pointers probably indicate the order in which I/O completed for a group of requests.

7. To help diagnose a C0D abend, the PVTDUMP bit (byte 0, bit 7 of the PVT) can be turned on (using superzap) to cause the RSM FRR to dump the PVT, PFT, SQA, and current LSQA data areas.

8. On a system with page fault assist active, if there is a problem with RSM-related control blocks, the problem might be caused by an earlier problem suffered by the page fault assist microcode. The occurrence of a page fault assist problem is indicated by a program check X'26', which RTM converts into a X'0D9' abend. The page fault assist microcode alters RSM

control blocks, and if unable to complete processing, might leave the control blocks partially updated before generating the program check X'26'.

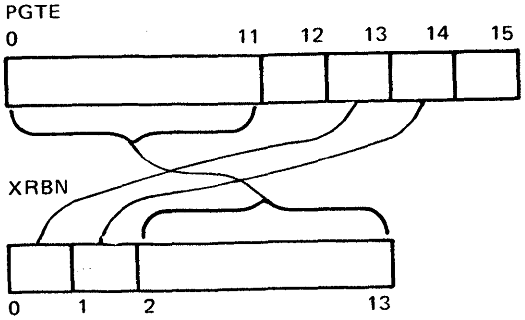
Converting a Virtual Address to a Real Address

A virtual address contains the segment number in the first byte, the page number in the next four bits, and the page displacement in the remaining twelve bits (that is, sspddd - segment, page, displacement). The ASCB for the address space points to the RSMHD. The first word (RSMVSTO) of the RSMHD is the virtual address of the segment table (SGT). Multiply the segment number (ss) by the length of a segment table entry (4) to locate the SGT entry (SGTE). The SGTE contains the real address of the page table (PGT).

A real address consists of a 14-bit real block number (XRBN) followed by a 12-bit page displacement (that is: rrrrddd-XRBN, displacement). The XRBN portion of the real address of the PGT is concatenated with zero (XRBN0) to form an index into the page frame table (PFT). This index is added to the apparent origin of the page frame table (PVTPFTP) in order to obtain the virtual address of the page frame table entry (PFTE). The PFTE identifies the frame that contains the page in which the page table resides.

The second half of the first word of the PFTE is the virtual block number (VBN). The VBN is concatenated with the displacement portion of the real address of the page table to form the virtual address of the page table (VBN||ddd). Multiply the page number (p) of the virtual address being converted by the length of a page table entry (2) to locate the PGT entry (PGTE). The PGTE contains the XRBN portion of the real address that corresponds with the initial virtual address.

Bits 13 and 14 of the PGTE are concatenated in front of bits 0-11 of the PGTE to form XRBN, as follows:



This XRBN is concatenated with the displacement portion of the initial virtual address to obtain the desired real address (RBN||ddd).

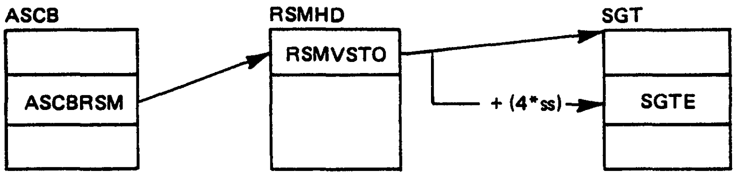
Figure 5-27 shows the relationship of the control blocks used to convert a virtual address to a real address.

Given a virtual address — find the corresponding real address.

Definitions: Virtual address = sspddd = VBNIIddd
- ss — segment number
- p — page number
- ddd — displacement within page
- VBN — virtual block number

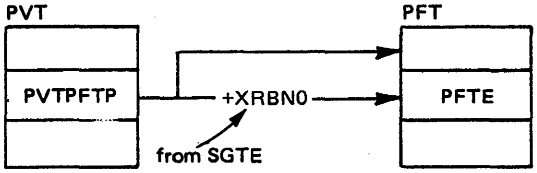
Real address = XRBNIIddd
- XRBN — real block number
- ddd — displacement within page

① Find the real address of the page table (RBNIId'd'd').



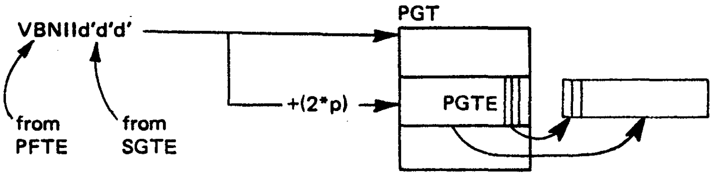
SGTE contains the real address of the page table

② Convert the real address of the page table to a virtual address (VBNIId'd'd').



PFTE contains the VBN portion of the virtual address of the page table.

③ Find the XRBN portion of the real address.



PGTE contains the XRBN portion of the desired real address.

④ Concatenate the displacement portion of the virtual address (ddd) with the real block number (XRBN) to form the real address that corresponds to the given virtual address.

real address = XRBNIIddd

Figure 5-27. Converting Virtual Addresses to Real Addresses

Example: Converting a Virtual Address to a Real Address

This example shows how a virtual address of A9EC0 was converted to a real address. The values used in this example (such as ASCBRSM = FC7380) were obtained from a sample dump.

Given: Virtual address = A9EC0 (sspddd)

ss = 0A (segment number)
p = 9 (page number)
ddd = EC0 (displacement within page)

Step 1: Find the real address of the page table (PGT).

ASCBRSM = FC7380 (address of RSMHD)
RSMVSTO = 89FC00 (address of SGT)

```
     89FC00 (RSMVSTO)
 +       28 (4*ss)
     ------
     89FC28 (address of SGTE)
```

SGTE = F0307F20

Real address of PGT = 307F20

Step 2: Convert the real address of the PGT to a virtual address.

XRBN = 307 and XRBN0 = 3070
d'd'd' = F20

PVTPFTP = 78760

```
     78760 (PVTPFTP)
 +    3070 (XRBN0)
     -----
     7B7D0 (address of PFTE)
```

PFTVBN = 87B0

Virtual address of the PGT = VBN||d′d′d′ = 87BF20

Step 3: Find the XRBN portion of the real address.

```
     87BF20 (virtual address of PGT)
 +       12 (2*p)
     ------
     87BF32 (virtual address of PGTE)
```

PGTE = 3811

XRBN portion = 381

Step 4: Form the real address for the sample.

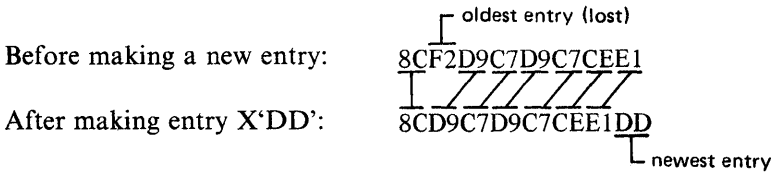
Real address = XRBN||ddd = 381EC0

PCB Trace Facility

To help you debug RSM, the PCB trace facility places trace entries in PCBs. A trace entry shows when a particular PCB was processed. (For example, a trace entry can show that RSM removed a PCB from the PCB defer queue to send it through GFA.) Trace entry information is available in a dump for those PCBs that are currently in use. The free PCB queue provides a record of recently completed RSM processing because the PCB queue is a FIFO queue and a PCB is cleared only when it is reused.

You enable PCB tracing at NIP time by setting the PVTTRXLN field in the PVT to the value (in eight-byte increments) that represents the size of the trace entry extension area you want appended to the end of each PCB. You use the SPZAP service aid to set the PVTTRXLN field, which must be set before the PCB pool is initialized. Once the PCB trace facility is enabled, it cannot be disabled for the life of the IPL.

An example of an eight-byte PCB trace area is:



In the example, X'8C' represents the RCA CSECT ID of the program that requested the PCB from the free PCB queue. The remaining seven bytes show the most recent trace entry points (IDs of the trace points within the RSM module).

See the program listing for module IEAVPCB (PCB manager) for detailed information on using the PCB trace facility. See the mapping macro IHARCA (in module IEAVPCB) for the trace IDs for the various RSM modules.

Auxiliary Storage Manager (ASM)

The auxiliary storage manager (ASM) controls all system direct access storage that is allocated for virtual address space paging and for virtual input/output (VIO) data sets. ASM supports the dynamic paging requirements of the real storage manager (RSM) and the data set storage and retrieval requirements of the virtual block processor (VBP). For MVS paging, ASM has the responsibility of selecting the auxiliary storage location (slot), maintaining the slot/page mapping, and coordinating the slot/frame transfer.

The auxiliary storage manager consists of three sections:

- I/O control
- VIO control
- VIO group operators.

I/O control is the link between RSM and the I/O supervisor for paging and swapping requests. I/O control accepts the paging/swapping requests from RSM, determines the type of I/O to be done and how it can be started, and notifies RSM when the I/O is completed. I/O control records the auxiliary storage locations of all virtual pages. I/O control also communicates with IOS to cause the physical transfer of data between real and auxiliary storage. It allocates auxiliary storage slots, builds paging channel programs, passes them to IOS for execution, and processes I/O completions.

VIO control coordinates all the ASM processing required to support VIO data sets (called logical groups by ASM). Operations on a logical group are classified as group operations and page operations. A group operation is not allowed to process while another group operation or page operation is processing for a logical group. The virtual block processor (VBP) initiates group-related operations and VIO control passes them to the VIO group operators to be processed. RSM initiates page-related operations and I/O control and VIO control jointly process them.

VIO group operations maintain the logical group information that VBP requires. The VBP group operators perform all processing necessary to create, save, restore, and delete a logical group. These operators are invoked only by VIO control as a result of requests from VBP.

Modules (CSECTs) belonging to each section are:

I/O Control		**VIO Control**	**VIO Group Operators**
ILRIODRV	ILRFRSLT	ILRPOS	ILRACT
ILRCPBLD	ILRCMP	ILRGOS	ILRSAV
ILRPAGCM	ILRMSG00	ILRVIOCM	ILRRLG
ILRSWAP		ILRSRBC	ILRTMRLG
ILRSWPDR		ILRJTERM	ILRVSAMI

Component Functional Flow

ASM provides eight functional services. The first five are invoked by the use of the ILRCALL macro, the remaining three via BALR:

- ***ASSIGN LG*** obtains a logical group identifier from ASM and creates a logical group for a VIO data set.
- ***SAVE*** preserves the status of a logical group for recovery at a later time.
- ***ACTIVATE*** places a logical group into active status after it has been saved and the saved status of the group is desired. (Used for step restart of VIO data sets.)
- ***RELEASE LOGICAL GROUP*** deletes an entire logical group; this allows ASM to reuse all slots associated with that logical group (VIO data set).
- ***XM ASSIGN*** requests a logical group identifier from ASM. It creates a multi-memory accessible logical group for a virtual fetch data set.
- ***TRANSFER PAGE*** moves the logical slot identifier (LSID) for a page from an address space to a VIO logical group.
- ***REQUEST I/O*** transfers page-sized blocks between real storage and ASM's auxiliary storage.
- ***REQUEST SWAP I/O*** transfers LSQA and all working set pages between real storage and ASM's auxiliary storage. Page-size blocks are transferred if page data sets are used. Swap-set size (up to 12 pages) blocks are transferred if swap data sets are used.

The following descriptions track three of these services through the component: SAVE, which is similar to assign LG, activate, and release logical group; request I/O; and request swap I/O.

Saving an LG

SAVE requests ASM to write the ASPCT containing the slot numbers (LSIDs) of a VIO data set to SYS1.STGINDEX. ILRGOS receives control from VBP in task mode with an ASM control area (ACA) containing the LGN of the VIO data set as input. ILRGOS builds an ASM control element (ACE), queues it to the logical group entry (LGE) process queue (LGEPROCQ) for that LGN, and calls ILRSAV.

ILRSAV calls ILRVSAMI, which calls VSAM to write the ASPCT to the SYS1.STGINDEX data set. An 'S' symbol is returned by ILRVSAMI. (The 'S' symbol is part of the VSAM key used to save this ASPCT and can be used to uniquely identify the ASPCT for an activate request). ILRSAV puts the 'S' symbol in the ACE and returns to ILRGOS. ILRGOS copies it into the ACA, frees the ACE, and returns to VBP.

Requesting I/O

RSM calls ILRIODRV for I/O requests. An ASM I/O request area (AIA), or string of AIAs describes the request. ILRIODRV determines if the request is for a VIO page and, if it is, calls ILRPOS to process it. Otherwise, ILRIODRV continues to process the request. Each request or group of requests from RSM is for only one type of data request. The request is either all reads or all writes.

For write requests, ILRIODRV frees the previous slot for this page, selects new page data sets and slots for the requests, and calls ILRCPBLD to start I/O.

For read requests, the LSID is obtained from the external page table entry (XPTE) and put into the AIA. The requests are separated into groups based on the paging activity reference table entry (PARTE) they are associated with. Each PARTE represents a paging data set. These groups of requests are given to ILRCPBLD to start the I/O.

ILRCPBLD builds the channel program for the requests and determines how the requests are to be started. If a channel program is currently running on the page data set that these requests are to be read from or written to, the new requests are chained on to the end of the running channel program. If the previous channel program on the page data set for these requests is suspended, IOSVRSUM is called to issue a RESUME I/O to start the requests. If the previous channel program on the page data set for these requests has ended, a START I/O is issued to process the requests.

New requests chained to a running channel program have a program controlled interruption (PCI) set in the first new request. The PCI generated indicates to ASM that the previous channel program (those requests already running when new requests were added) has ended. Channel programs that support SUSPEND/RESUME services also have a PCI set at the end of the channel program to indicate that the requests ahve completed and the page data set is suspended.

When a PCI occurs, IOS calls ASM's PCI exit ILRCMPCI (an entry point in ILRCMP). ILRCMPCI determines that the PCI is one of ASM's channel programs and processes the requests that have completed. The requests are passed directly to RSM (IEAVPIOP) for single page-ins or to ILRPAGCM for all other request types.

For I/O interrupts other than PCI, IOS calls ASM's disabled interruption exit (DIE) routine (ILRCMPDI - an entry point in ILRCMP). ILRCMPDI checks for errors, and if one occurred, returns to IOS indicating that the I/O should be handled by the post status IOS routine and ASM's appendages (ILRCMPAE and ILRCMPNE). If the I/O is successful, ILRCMPDI calls page completion (ILRPAGCM). ILRPAGCM calls VIO completion (ILRVIOCM) if the I/O is for a VIO page. If it is a non-VIO write request, ILRPAGCM takes the LSID that ILRIODRV put into the AIA and puts it in the XPTE for the page in the correct address space. The AIA is then returned to RSM (IEAVPIOP).

Requesting Swap I/O

RSM calls ILRSWAP with a chain of AIAs for either a swap-in or swap-out request. The following discussion traces a swap-out operation.

ILRSWAP separates the nonworking AIAs from the working set AIAs and calls ILRIODRV to process the nonworking set pages as a regular request I/O function. The working set AIAs are queued to the ASM header of the address space (ASHSWAPQ). If there were no nonworking set AIAs, ILRSWAP immediately calls ILRSLSQA to process the working set AIAs. Otherwise, ILRSWAP returns to RSM.

As nonworking AIAs complete, ILRPAGCM is given control (see Requesting I/O). When all nonworking set AIAs have completed, ILRPAGCM calls ILRSLSQA to process the working set AIAs. ILRSLSQA, called by ILRPAGCM or ILRSWAP, passes control to alternate entry points in ILRIODRV to process the working set AIAs if there are no available swap sets. Otherwise, ILRSLSQA assigns swap sets and initializes swap channel control work areas (SCCWs) for all the AIAs queued to ASHSWAPQ. A count of LSQA pages (ASHSWPCT) is incremented for each AIA. The completed SCCWs are chained to the swap activity reference table (SART) entry SCCW queue (SRESCCW). If an SRB is not already scheduled for swap driver (ILRSWPDR), ILRSLSQA schedules one. ILRSWPDR searches each SART entry for a non-zero SCCW queue, chains the SCCWs to an IORB for that data set, and issues a STARTIO macro to initiate I/O processing. Completed I/O is handled by ILRCMPDI as in the "Requesting I/O" function, and ILRPAGCM is called. ILRPAGCM processes working set AIAs by putting the LSID for each page into the SPCT control block for this address space, putting the AIA on the capture queue (ASHCAPQ), and decreasing the swap count (ASHSWPCT) by 1. When the swap count is 0, ILRPAGCM returns all the AIAs on the capture queue to RSM (IEAVSWPC module).

Figure 5-28 shows the relationship among the important ASM control blocks.

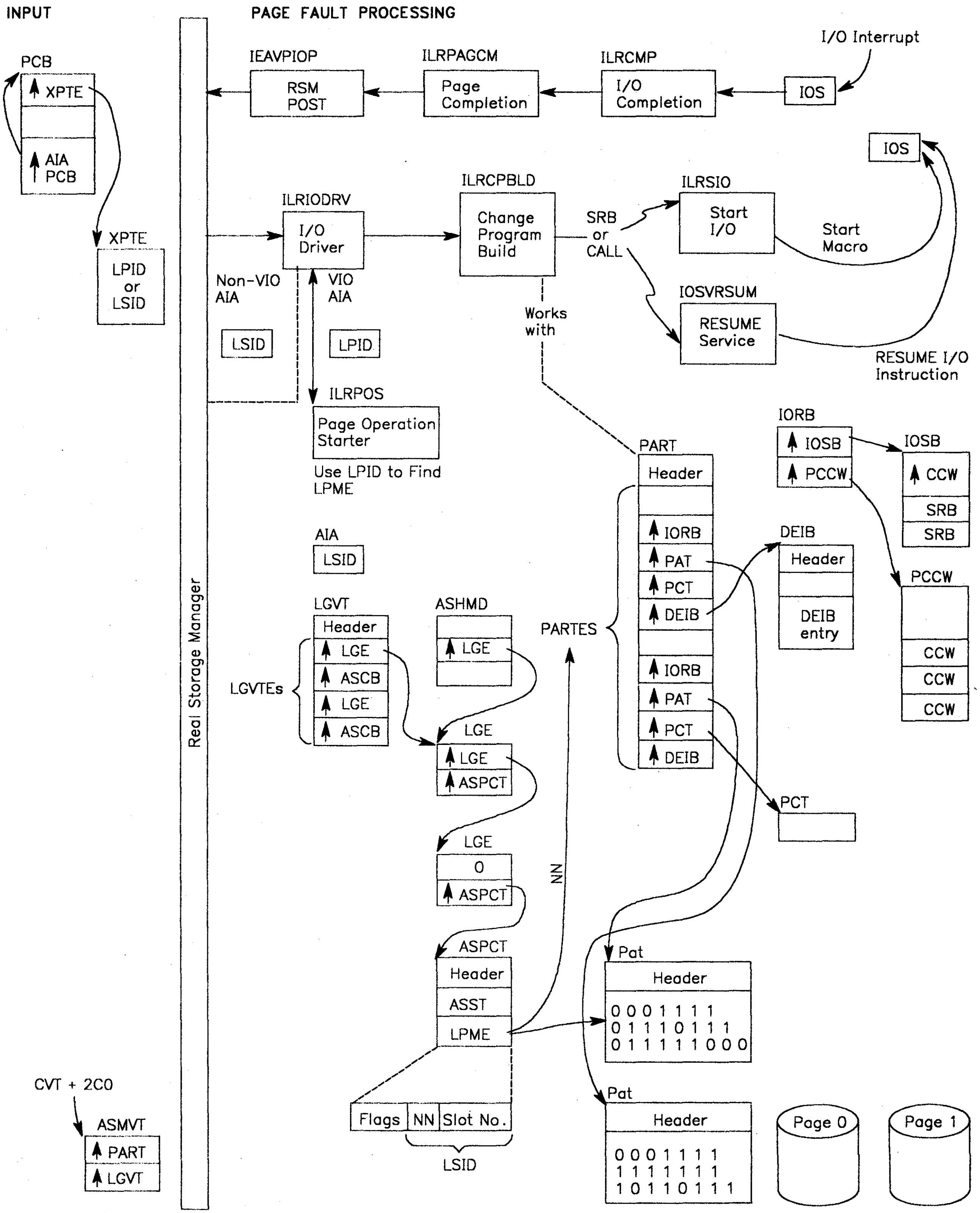


Figure 5-28. Relationship of Important ASM Control Blocks

Component Operating Characteristics

The following topics discuss characteristics of ASM's operating environment.

System Mode

ASM uses the SALLOC lock in most page and swap processing in I/O control modules. I/O control modules interface directly with RSM, the principal user of the SALLOC lock. The SALLOC is held throughout processing, including the calls to the VIO control routines ILRPOS and ILRVIOCM. The local lock is used during assign and release logical group requests processed by ILRGOS and ILRRLG.

An ASM class lock exists for each active address space (lockword in ASMHD). The ASM class lock is used by the VIO control modules. The ILRCMPDI, ILRCMPCI, and ILRCMPNT entries of ILRCMP run in physically-disabled mode because they run under the I/O interrupt handler. (ILRCMPNT also runs under IOS front end processing.) The rest of ASM's modules simply run in task or SRB mode using compare and swap instructions where necessary.

For additional information on locking, refer to the topic "ASM Serialization" later in this section.

Address Space, Task, and SRB Structure

The I/O control modules given control from RSM and IOS run in the address space of the caller. These modules (and entry points) are:

ILRIODRV
ILRSWAP
ILRCMPDI
ILRCMPCI
ILRCMPNT
ILRCPBLD
ILRPAGCM
ILRFRSLT

Note: ILRPAGCM transfers to the correct address space (via CMSET) to update the external page table entry (XPTE) that is in the LSQA.

The remaining I/O control modules and alternate entry points in other modules run in SRB mode in the master scheduler's address space.

VIO group operator modules, as well as ILRGOS (VIO control module), are tasks (locked mode) executing in the address space associated with the VIO requests. ILRTMRLG runs in task mode, but in the master scheduler's address space.

VIO control modules ILRPOS and ILRVIOCM receive control in the address space of the caller. ILRSRBC executes in SRB mode in the address space associated with the VIO requests.

Storage Considerations

ASM maintains three cellpools for its internal control blocks. These cellpools are pushdown stacks and the elements at the top of the cellpools represent the last control blocks used by ASM. The cellpools are for work areas, ACEs, BWKs, and SWKs. These cellpools are expandable. The cellpools are anchored in the ASMVT and the control blocks reside in SQA. The ASMVT is in the nucleus, but most of the other ASM control blocks are in SQA. One exception is the ASPCT, which resides in the LSQA of the associated address space.

Interfaces With Other Components

Five other components interface with ASM:

- RSM with I/O control for page and swap I/O requests, and with VIO control for transfer page requests.
- VBP with VIO control for assign, save, activate, and release logical group requests.
- IOS with I/O control to process I/O requests.
- VSAM with VIO group operators to handle I/O to SYS1.STGINDEX. RSM and VBP call ASM, and ASM calls IOS and VSAM.
- Contents supervisor with VIO control for XM ASSIGN logical group requests.

Register Conventions

ASM modules adhere to the following register conventions when calling other ASM modules. There are some exceptions where certain addresses are not required.

Register: 0 - Parameter register, if required.

1 - Parameter register, if required.

2 - RSMHD or ASMHD address for the current address space or the address space identified by an input parameter in register 0 or 1.

3 - ASMVT address.

4 - Address of ATA or EPATH currently active for recovery tracking.

13 - Address of register save area, if required.

14 - Return address.

15 - Entry point address.

Footprints and Traces

The most useful traces of ASM processing are its control blocks and queues, because they document the movement of requests through ASM.

The processor work/save area vector table (WSAVT), which is pointed to by LCCACPUS, will point to the work/save areas for the last I/O processed on the processor. WSACASMS points to the 1024-byte save/work area used by ASM routines running as SRBs: ILRSWPDR, ILRCMPAE, ILRCMPNE, and ILRCMP. WSACASMD points to another 1024-byte save/work area for disabled ASM routines: ILRIODRV, ILRCPBLD, ILRCMPDI, ILRCMPCI, and ILRCMPNT. WSACASMR points to a 512-byte save/work area used by ILRIOFRR. ILRPAGCM uses the work area of its caller.

ASMVT contains save areas for ASM's other I/O-related modules. ASMBWKPC is a pool of work areas used by VIO-related modules (ILRGOS, ILRACT, ILRSAV, and ILRRLG). Bits in the X'01' byte of the ASMVT indicate whether the IPL was a cold, quick, or warm start.

The LGE process queues (LGEPROCQ) contain AIAs and ACEs in process, or AIAs and ACEs waiting for processing of VIO requests.

Field IORNOP (IORB+X'2C') points to the end of the channel program pointed to by the field IORPCCW. If ASMTMECB (ASMVT+X'A8') is a posted ECB, ILRTMRLG is or was about to process the task portion of a release logical group request.

When an ASM-locked or SRB-mode routine is processing, its functional recovery routine is on the current FRR stack. The first word of the parameter area passed to the FRR contains a one-byte id of the ASM module that established the FRR, followed by three bytes of flags indicating the ASM module or entry point in process at the time of the error. The different ids are discussed in "Recovery Footprints."

When ASM's I/O completion module encounters the first bad slot, an error record is built with its address at X'14' into the ASMVT. It contains the LSIDs of the unusable slots. The first three bytes in the record are the address of the current entry filled, beginning address of the record, and ending address of the record. An entry contains one byte of flags and the three-byte LSID. If bit 0 is on, the error occurred on a swap data set. If bit 4 is on, there was a read error. I/O error counts are found in the ASMVT, PART entries and SART entries. ASMERRS (ASMVT+X'7C') is the total of error slots found on local page data sets. PARERRCT (PART entry + X'18') and SRERRCNT (SART entry + X'18') are the error slots encountered on the particular data set represented by the entry.

In the ASMVT there are four counts, ASMIORQR (ASMVT+X'28') and ASMIORQC (ASMVT+X'2C'), which contain both the number of paging I/O requests ASM has received and the number completed; and ASMSWRQR (ASMVT+X'30') and ASMSWRQC (ASMVT+X'34'), which contain both the number of SWAP working set requests ASM has received and the number completed. If more requests have been received than completed and the system is waiting, there is something wrong with ASM or IOS.

General Debugging Approach

This description helps isolate paging problems, the most difficult problems to debug. Paging problems (not all of which are ASM problems) fall into two main groups - paging interlocks and incorrect or duplicate pages.

Paging Interlocks

Paging interlocks result in an enabled wait state. There are two indicators that hint that the enabled wait is a paging interlock:

- The I/O request counts in the ASMVT (ASMIORQR and ASMIORQC, or ASMSWRQR and ASMSWRQC) are not equal. The PAREREQS field (PARTE X'3E') in the PART entries indicates which data sets have uncompleted I/O requests when ASMIORQR is greater than ASMIORQC.

- Field ASMPSRB points to an SRB used to redrive work through ASM. The SRB parameter field points to an 8-byte parameter list that contains the addresses of the first and last requests on the redrive queue. If these are nonzero, the ASM redrive SRB has been scheduled, but not dispatched. It is necessary to determine why the SRB has not been dispatched.

The blocks discussed here are in the *Debugging Handbook*. To find the I/O request blocks for a given page space, start with the PART entry. The PART entry points to the first IORB.

There is one IORB for each page data set on a disk, four for each page data set on a drum, and three for each page data set on a cached auxiliary storage subsystem. The first bit of the fourth byte indicates whether or not ASM has passed the IOSB to IOS. If the bit is 0, the IORB/IOSB is available. If the bit is 1, the IORB/IOSB is in use. The IORB contains fields that point to the IOSB, to the first of a queue of PCCWs, and to the last CCW in the channel program. For an active I/O request, the third word into the PCCW points to the associated AIA, and the fourth word points to the next PCCW on the chain.

If the request has been sent to IOS and not returned, it is necessary to trace IOS processing. If I/O processing has caused a page-fault or a request for an enabled lock, the interlock is probably explained. Either ASM could not get the resources to handle the page fault, the page is already in use and this request is backed up behind the previous one, or the holder of the lock has page-faulted and the page fault cannot be resolved.

Incorrect Pages

It is almost impossible to determine from a dump what caused the wrong page to be written or read. At best, a dump provides clues as to which general area is causing the problem. Intensive code reviews are then necessary to find it. Frequently, traps must be applied to narrow the area further.

The following paragraphs contain descriptions of how to find various pieces of useful information. There is no attempt to describe how to use them because there is no general method.

It is first necessary to determine which page contains bad data and whether the whole page or only part of it is bad. If possible, also determine which page has overlaid the bad page. If only part of the page is bad, the error probably occurred while handling a track overflow record to or from an alternate track. Check for an invalid first or last part of a page. ASM is unlikely to be the cause of invalid data in the middle of the page.

Incorrect pages cause a system failure when the page is used by a system task or by a routine holding a critical system resource. The invalid page is more likely to cause an address space to fail because of program checks that result from invalid data. These failures are rarely attributed to invalid pages.

Scan the SYS1.LOGREC data set for any improbable program checks and obtain any associated dumps. Multiple versions of the same problem are helpful in suggesting a pattern for the error. For example, the error might only occur for the second page of LSQA or only on a page associated with an overflow record.

Finding the LSID for a Given Page

A virtual address contains the segment number in the first byte, the page number in the next four bits, and the page displacement in the remaining bits in the form sspddd (segment, page, displacement). The ASCB for the address space points to the RSMHD. The first word of the RSMHD is the virtual address of the segment table. Multiply the segment number (ss) by the length of the segment table entry (4) to locate the correct entry. It contains the real address of the page table (PGT). Convert this address to a virtual address. Then locate the correct external page table entry (XPTE) by adding 16 times the length of the page table entry (2), and adding the page number (p) times the length of a XPTE (12).

The XPTE contains information about the status of this page on auxiliary storage. If either the XPTVALID or XPTVIOLP flag is on, there is a slot assigned to this page. If XPTVALID is on, the LSID (slot identifier) is in the XPTE. If the page is duplexed, two LSIDs are in the XPTE (one for each slot). If XPTVIOLP flag is on, an LPID instead of an LSID is in the XPTE. To relate the LPID to an LSID, see the following topic "Finding LSIDs of VIO Data Sets."

Finding LSIDs of VIO Data Sets

The ASPCT is used to record the auxiliary storage locations (LSIDs) of VIO data set pages. Only a 1088 byte base ASPCT is created at ASSIGN LGN time. This ASPCT can handle up to 1 megabyte of VIO data set space. If more than 1 megabyte of VIO space is used, the ASPCT is expanded as follows:

1. For each 256 megabytes of space up to 1 billion bytes, an ASST extension is built.

2. For each megabyte of space, a LMPE extension is built.

Each ASST or LPME extension requires 1088 bytes of storage. Each ASST extension contains a vector table of LPME extension addresses. The ASPCT (base and all extensions) resides in the LSQA of the associated address space.

The LPID is eight bytes. The first four bytes contain an LGID, logical group (VIO data set) identifier. The second four bytes contain a relative page number (RPN).

When given an LGID, there are two methods to locate an ASPCT:

1. The ASCB (of the desired address space) points to the ASMHD. ASHLGEQ in the ASMHD is the queue of LGEs (active VIO data sets) related to this address space. Searching through the address space's ASHLGE queue, one of the LGEs will have an LGELGID field that matches this LGID. This same LGE has the address of the needed ASPCT (LGEASPCT).

2. Another way to locate an ASPCT from an LGID is to follow the CVT to the LGVT (CVTASMVT, ASMLGVT). Using the LGID as an index, locate the appropriate LGVT entry. The LGVT entry contains the address of the LGE that contains the address of the needed ASPCT.

Note: The second method must be used to locate the ASPCT of a cross-memory assigned VIO data set.

With the appropriate ASPCT, now use the RPN portion of the LPID as an index to locate the LPME containing the associated LSID.

Figure 5-29 shows the pointers and control blocks described in the following paragraphs.

If A' and AA are both zero, use the LL to index ASPLPMES in the ASPCT base for the LPME containing the LSID.

Otherwise, use A' to index ASPASSTP for the address of the appropriate ASST extension. Use AA to index the ASPSECTA of the ASST extension for the address of the appropriate LPME extension. And use LL to index the ASPSECTA of the LPME extension for the LPME containing the LSID.

The LSID is the slot identifier for this page of the VIO data set. This LSID can be related to the ASM control blocks PART and PAT and to the actual paging device. See the following topic "Locate PART and PAT Bit."

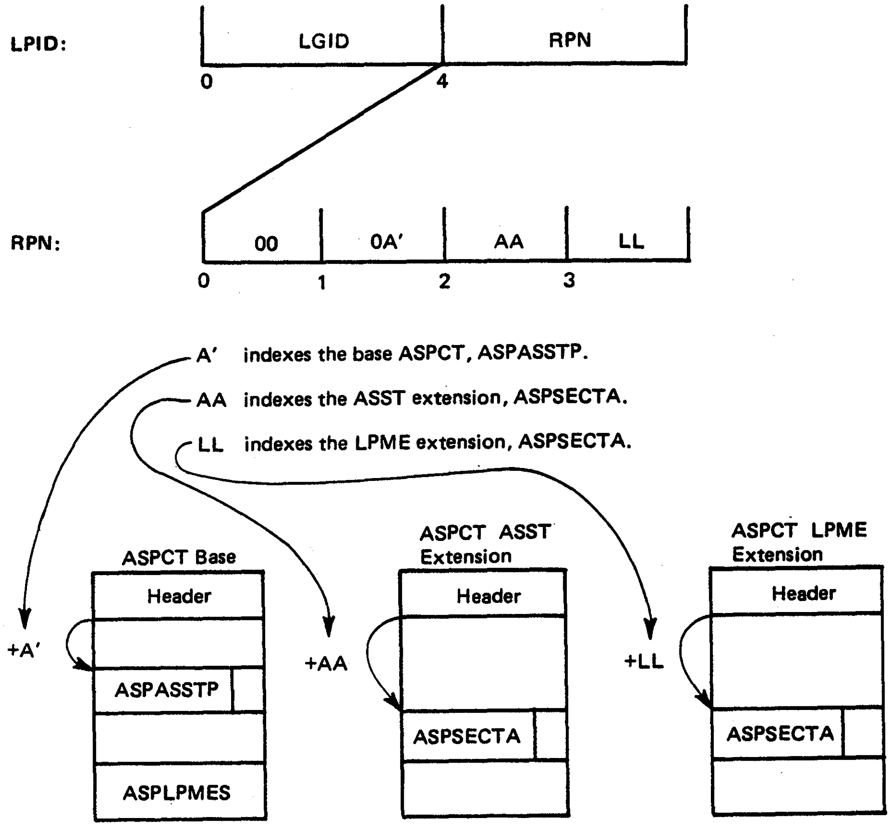


Figure 5-29. Locating An LSID From An LPID

Locate PART and PAT Bit

Suppose LSID 000403BD was found in the XPTE that represents the sample address 07A12C:

> PART entry index is 04.
> Relative slot number is 03BD.

The PART has one entry for each page data set, each having a pointer to its PAT. The PAT is a cylinder bit mapping of this page data set. PATCYLMW is the number of words that map a cylinder. PATCYLS, slots per cylinder, is the number of significant bits in each cylinder mapping.

For device 2305-2:

> PATCYLMW is 1
> PATCYLS2 is(A26).

To locate the bit in the PAT map for slot 03BD(957):

1. address of map word = (address of PATMAP) + 147 = (address of PATMAP) + (957/26) x PATCYLMW x (bytes in a word))

2. bit in the map word (origin 0) = 957/26 = 21.

Figure 5-30 shows the control blocks involved in relating a virtual address to the PART and PAT.

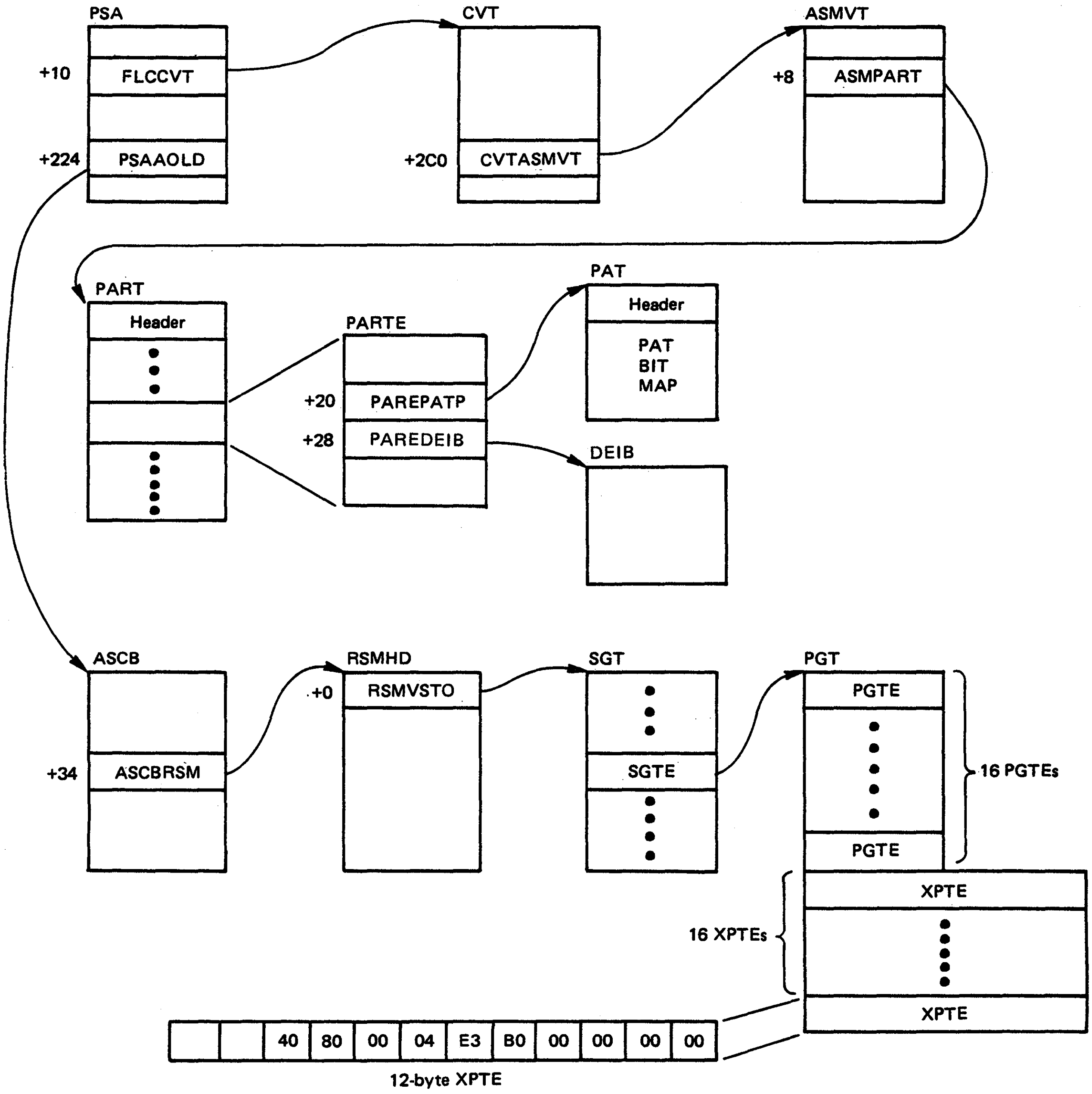


Figure 5-30. Relating the Virtual Address to the PART and PAT

Converting a Slot Number to Full Seek Address

The full seek address can be used to read the record from the disk and determine exactly what it does contain.

The PART entry points to the data set extension information block (DEIB) for the data set. The DEIB describes the page data set on the device. The DEIB consists of an 8-byte header followed by entries, one for each extent. The second word of the header contains the number of entries in this DEIB and the length of each entry.

The relative cylinder and the slot number within the cylinder that the slot is on is calculated by dividing the relative slot number by the number of slots in a cylinder. The extent containing this relative slot is found by comparing the relative cylinder found previously to the low and high relative cylinders in each extent. The physical cylinder (CC portion of MBBCCHHR) is found by obtaining the relative cylinder within the extent and adding it to the real starting cylinder of the extent (also found in the DEIB entry). The head and record (HHR portion of MBBCCHHR) is found by indexing into the physical characteristics table (PCT) of the page data set using the slot number within the cylinder found previously.

For example, when given a relative slot number of 03BD(957), calculate the MBBCCHHR for device 2305-2.

M = extent number = 0
BB = 00

relative cylinder = relative slot number/cylinder size
= 957/26 = 36

slot within cylinder = relative slot number
- (relative cylinder x slots per cylinder)
= 957 - (36 x 26)
= 957 - 936 = 21

CC = (relative cylinder - DEILOCYL + DEISTCYL)
= (36 - 0 + 0) assuming the page data set starts on physical cylinder 0.
= 36 (X'24')

HHR = (PCTHHR (slot within cylinder + 1)) assumes one origin table
= (PCTHHR(22))
= 000604
= head 6, record 4

Therefore: MBBCCHHR = X'0000000024000604'

Unusable Paging Data Sets

Certain types of I/O errors received at completion of I/O indicate that ASM is either unable to, or would be ill-advised to access a particular auxiliary storage data set any longer. ILRCMPAE, an entry in ILRCMP, receives these errors and marks the data set as unusable. For page data sets, the DSBD flag in the PART entry is turned on and the PART entry is removed from the circular queue of entries. For swap data sets, the DSBD flag in the SART entry is turned on. Both flags are X'0A' into the respective entries, and are set to X'04'. ILRMSG00 is then called to determine whether ASM can continue processing without the data set.

If ASM is unable to continue processing, ILRMSG00 issues message ILR008W and terminates the system with a X'02E' wait state.

At this point, a stand-alone dump should be taken to determine which of the above conditions occurred. The console sheet, if available, might also help because ASM may have previously issued message(s) ILR009E.

If ASM is able to continue processing without the unusable data set, message ILR009E is written to the operator. This message indicates which volume contains the unusable data set. If this message occurs, use the DUMP command to take an SVC dump of master's address space to determine what error occurred. The options specified should include NUC and SQA.

To determine from the dump what error occurred, the PART or SART entry for the unusable data set and the IOSB for the failing request must be located. Use the AMDPRDMP service aid (print dump) with ASMDATA control statement to print the dump. One of the following conditions occurred on the data set to make it unusable:

- If the number of write errors (X'18' into the entry: PARERRCT or SRERRCNT) is 175, ASM has stopped using the data set because it has incurred too many write errors (one way for this to happen is if the data set was not formatted).

- If the completion code (X'0D') in the IOSB is a X'51', ASM has stopped using the data set because there is no longer a path to the device (this could happen as a result of an ACR condition), or there is an excessively high channel rate on the path to the data set.

- If the completion code in the IOSB is a X'6D', ASM has stopped using the data set because the channel or the device has become non-operational.

- If the completion code in the IOSB is a X'41', the device status in the IOSB (offset X'18') is X'02' and the sense data in the IOSB (offset X'2A') is X'1000', ASM has stopped using the data set because an equipment check occurred.

- If the completion code in the IOSB is a X'41' and the channel status in the IOSB (offset X'19') is X'08', X'04', or X'02', ASM has stopped using the data set because a channel check occurred.

The system is terminated only if this unusable data set (or several unusable data sets) caused one of the following conditions:

- The unusable paging data set contains PLPA pages and the duplex data set, if any, is already unusable or full.

- The unusable paging data set is the duplex data set and not all PLPA pages are accessible (that is, the PLPA paging data set or a data set containing PLPA pages is unusable).

- The unusable paging data set is the last local paging data set.

Page/Swap Data Set Errors

Figure 5-31 shows the messages issued and the actions taken by the ASM I/O subsystem for various error conditions with the page and swap data sets.

Duplex Status	Error Conditions		Message(s) Issued	ASM Action Taken
Duplexing Active	PLPA Full	Common *Available	ILR005I	Spill to Common
		Common **Unavailable	ILR010I	Duplex Only
	PLPA Bad		ILR009E, ILR010I	Duplex Only
	Common Full	PLPA Available	ILR006I	Spill to PLPA
		PLPA Unavailable	ILR010I	Duplex Only
	Common Bad		ILR009E, ILR010I	Duplex Only
	Duplex Full	PLPA or Common Available	ILR007I	Suspend Duplexing
		PLPA and Common Unavailable	ILR008W	Wait X‘03C’
	Duplex Bad	PLPA and Common Available	ILR007I	Suspend Duplexing
		PLPA or Common Full	ILR007I	Suspend Duplexing
		PLPA or Common Bad	ILR008W	Wait X‘02E’
		PLPA and Common Unavailable	ILR008W	Wait X‘02E’
Duplexing Not Active	PLPA Full		ILR005I	Spill to Common
	PLPA Bad		ILR008W	Wait X‘02E’
	Common Full		ILR006I	Spill to PLPA
	Common Bad		ILR008W	Wait X‘02E’
	PLPA and Common Full		ILR008W	Wait X‘03C’
In Either Case	Local Bad		ILR009E	Stop Writes to Bad Data Set
	Last Local Bad		ILR008W	Wait X‘02E’
	Swap Bad		ILR009E	Stop Swap-outs to Bad Data Set
	Last Swap Bad		ILR009E	All Swap-outs Done to Page Data Sets
	Last Local Full		none	Wait X‘03C’
	Last Page Data Set Eligible for VIO Bad		ILR011E	Spill to NONVIO Page Data Sets

*Available - Data Set Neither Full Nor Bad
**Unavailable - Data Set Either Full or Bad

Figure 5-31. Page/Swap Data Set Error Action Matrix

Error Analysis Suggestions

The following are some guidelines for determining ASM problems:

- Print the dump specifying ASMDATA as a control statement to AMDPRDMP.

- Check SYS1.LOGREC and the LOGREC buffer to see if ASM's mainline has abended. If it has, a request might have been lost or mishandled.

- Check the trace table for recent ASM activity. The key trace table entries are SRB dispatches for ILREDRV (address of SRB in ASMPSRB, X‘58’ into ASMVT), ILRSIO (address of SRB in IORSRBP, X‘34’ into IORB), or

ILRSWPDR (address of SRB is SARSRBP, X'30' into SART). Also look for schedules of the post status SRB closely following an interrupt for ASM I/O (CSW points to the nucleus area), which could be temporary or permanent I/O errors coming to ILRCMP or one of its entry points.

- Check for outstanding I/O requests and determine the status of the I/O by looking at the UCB and IOSB.
- Check for I/O errors on the paging packs, either on the error record (X'14' into the ASMVT), or on SYS1.LOGREC.
- Scan the ASMHD's LGE process queues (LGEPROCQ) for current VIO activity. Determine the extent of ASM processing for these LGEs. Determine the logical group for which a VIO group operator has been requested.
- Check the PART no-PCCW queue for requests waiting to be redriven through ASM. Also scan the SART for the SRELOCK flag indicating that ILRSWPDR should be processing.
- If you are interested in a specific request, find the request on ASM queues and determine the extent of ASM processing for the request. For an I/O request, convert the virtual page number to an LSID.
- Scan the BWK and SWK cell pool for a work area that is not chained to another work area (offset 0). An unchained work area indicates current ASM processing or a lost work area.
- Check for suspended ILREDRV or ILRSWPDR SRBs by scanning the PCB I/O queues (pointed to by the RSMHD and the PVT) for a suspended SRB whose address matches ILREDRV, ILRSIO, or ILRSWPDR SRB's address. Although this situation should not occur, it does occur occasionally.

Validity Checking

ASM is a nucleus-resident, performance-oriented component. For this reason, there is minimal validity checking in mainline code. In addition, few of ASM's problems can be attributed to invalid control blocks; this is probably because ASM communicates only with other system components. In both mainline and recovery code, critical global control blocks such as the ASMVT, PART, and SART, are used without any validity checking. ASM's recovery routines do validity check control blocks (and queues of these control blocks) that represent work to be processed by ASM. Some of these control blocks are the ACE, AIA, LGE, and PCCW. In most cases, if a control block fails the validity check, it is no longer used by ASM. The only exception is the IORB-IOSB-SRB-SRB combination, which is refreshed.

ASM Serialization

Serialization of ASM processing is done using the SALLOC and ASM global locks, the local lock of the current address, compare-and-swap (CS) logic and control block queueing.

SALLOC Lock

ASM uses the SALLOC lock to serialize most page and swap processing in I/O control. The I/O control modules interface directly with RSM, the principal user of SALLOC, either as the called routine or the calling routine. The SALLOC is held throughout processing including calls to the VIO ILRPOS and completion routines. The SALLOC is used to serialize most processing of:

IORBs	- complete coverage, except as noted below.
XPTEs	- complete coverage.
PCB/AIAs	- complete coverage, except AIA noted below.
SPCTs	- complete coverage.
SART	- complete coverage, except where noted below.
SATs	- complete coverage.
PATs	- complete coverage.

Specific areas of other control blocks serialized by the SALLOC lock are:

ASMVT	- Work save areas. I/O control section fields. Flags - ASMDUPLX ASMNPRIM ASMCALLQ ASMNODPX ASMPLPAF ASMCOMMF - LGVT pointer - Error record pointer. - ASMCOMDS - index into PART entries for common area writes. - Request counts. - Available PCCW queue. - Non-VIO slot allocated count. Expansion of ASM pools.
ASMHD	- I/O control flags. Swap and page counters. Swap queue.
ASCB	- Non-VIO slot allocated count.
LGVT	- Available LGVTE queue. Expansion of the LGVT. Creation/deletion of LGE.
PART	- Count of local page data sets. Circular queue pointers.

Modules whose processing is serialized by the SALLOC lock are:

ILRIODRV	complete coverage, held by caller on called entries, obtained on SRB entries.
ILRPAGCM	complete coverage, held by caller.
ILRFRSLT	complete coverage, held by caller.
ILRSWAP	complete coverage; held by caller.

ILRCPBLD dcomplete coverage, except ILRSIO entry point. When held, the lock is held by the caller.

ILRCMP complete coverage, obtained at entry.

ILRMSG00 complete coverage for main entry point, held by caller.

ILRPOS complete coverage, held by caller (except for ILRTRANS entry point).

ILRVIOCM complete coverage, held by caller.

ILRGOS only obtained for LGVT processing and GETMAIN/FREEMAIN requests.

ILRPGEXP only obtained to adjust the SART to reflect addition of a new swap data set and update the count of local page data sets and new page data sets on the PART.

ILRTERMR obtained when referencing above control blocks.

ILRPEX obtained when expanding an ASM pool.

ASM Class Locks

The ASM lock is a global spin class lock. A lockword must be provided when obtaining or releasing an ASM class lock. A class lock exists for each active address space. The lockword is in the ASMHD. It is used by the VIO controller modules. The address space class locks serialize processing of the following control blocks:

AIA VIO controller flags, LPID field.

ASMHD VIO controller flags, LGE queue base pointer.

ASCB VIO slot allocation count.

LGE complete coverage, except creation/deletion (SALLOC).

ACE complete coverage.

ASPCT complete coverage while group operations are in progress. Group operations and page operations can be executed in parallel. VIO controller processing of the LGE process queue provides this serialization.

The address space class locks serialize processing in the following modules:

ILRGOS partial, obtained where processing above control blocks.
ILRPOS complete coverage.
ILRSRBC partial, obtained when searching LGE queue and LGE process queues.
ILRVIOCM complete coverage.
ILRJTERM partial, obtained when adding ACEs to LGE process queue.

Local Lock of Current Address Space

The local lock is used by VIO controller and VIO group operator modules to serialize certain VIO related operations. It is used by ILRGOS (held on entry) and ILRJTERM (obtained) to serialize release LG requests with the internal ASM deactivate function used to clean up VIO logical groups for a terminating job. The local lock is also used by most VIO-related modules to allow use of branch entry GETMAIN, rather than the SVC route.

Compare and Swap (CS) Serialization

Certain modules of ASM run without locks, requiring CS serialization of pointers, flags, and counts. Where routines running with the locks change fields used by unlocked routines, CS must be used. VIO group operators run unlocked and are the principal users of compare and swap. Control blocks serialized via CS include:

ASMVT -	group operator sections. pool controllers. VIO slot count.
SART -	A special CS lock exists in each SARTE to serialize swap driver processing of the swap data sets. Other fields updated by the swap driver are updated with CS.

The ASM modules that run without locks, using CS to serialize control block fields are:

ILRSWPDR
ILRSAV
ILRACT
ILRRLG
ILRTMRLG
ILRVSAMI

Serialization via Control Block Queues

Certain ASM control blocks are serialized via their available queues. The blocks are kept on available queues and removed when needed. While in use the block is so marked and associated with a specific operation and/or control block. Control blocks included in this category are PCCWs, IORBs for swap data sets, and SCCWs.

The ASPCT is a special case. VIO control enforces the rule that page and group operations cannot be performed in parallel for a given logical group and its ASPCT. This is controlled by the LGE process queue. While paging operations are being performed, the ASPCT is serialized via the ASM class lock of the owning address space. While a group operation is in progress, ASPCT serialization is maintained by the ACE for the group operation that is on the LGE process. This ACE prevents any other processing of ASPCT until the group operation completes.

Recovery Considerations

The philosophy of ASM's recovery is to allow the system and ASM to continue processing. To accomplish this, the first step in ASM's recovery routines is to validity check any control block or queue that might have been affected by the error. This is to allow future ASM processing to proceed without error. The second step in ASM's recovery is to notify ASM's caller that an error has occurred. In a few instances where ASM is directly invoked by RSM (such as ILRIODRV or ILRSWAP), ASM recovery attempts retry to return to RSM with a failing return code. When an error occurs during ASM processing that runs asynchronously, ASM recovery queues the failing request for eventual return to RSM. When an error occurs during ASM processing of a VIO group operator request, ASM recovery cleans up its resources and allows the associated task to terminate.

Recovery Traces

A dump of SYS1.LOGREC is a prerequisite to debugging ASM problems. ASM's recovery always records the SDWA to the SYS1.LOGREC data set. It is the most convenient way of determining that recovery has been invoked. The recovery routine ID in the SDWA indicates which recovery routine was invoked.

ASM has a number of system abend completion codes ('08x' series) that are always set up for retry. The purpose of these ABENDs is to record to SYS1.LOGREC logical errors that have occurred in ASM's mainline or VIO processing.

Recovery Structure

ASM has eight recovery routines for ASM mainline:

- ILRIOFRR is an FRR that provides recovery for ILRPAGCM, and ILRVIOCM. It also acts as a router, giving control to the routines in ILRSWP01 and to ILRPOS01.
- ILRSWP01 contains recovery routines for ILRSWPDR and ILRSWAP.
- ILRDRV01 is an FRR that provides recovery for I/O driver (ILRIODRV) and channel program build (ILRCPBLD), and also routes control to ILRPOS01.
- ILRCMP01 is an FRR that provides recovery for the I/O completion routine (ILRCMP).
- ILRGOS01 is both an FRR and an ESTAE that provides recovery for ILRGOS, for the group operators ILRSAV, ILRACT, and ILRRLG, and for ILRVSAMI.
- ILRTMI01 is the ESTAE that provides recovery for ILRTMRLG and for its path through ILRVSAMI.
- ILRSRB01 is an FRR that provides recovery for ILRSRBC.
- ILRFRR01 is a validity check routine called by most of the recovery routines.

Additional recovery routines are:

- TERMRFRR is an FRR that is an entry point into and provides recovery for ILRTERMR.
- ILRJTM01 is an FRR that is an entry point into and provides recovery for ILRJTERM.
- ILRMSG01 is an FRR that is an entry point into and provides recovery for ILRMSG00.
- ILRPOS01 is an alternate entry in ILRIOFRR and provides recovery for ILRPOS.

- ESTAER is an ESTAE that is the entry point into and provides recovery for ILRPGEXP.

- ESTAEXIT is an ESTAE that is an entry point into and provides recovery for ILRPREAD.

Recovery As a Debugging Tool

Recovery has a beneficial effect on problem solving primarily because having it invoked isolates the problem to a specific area of ASM. If there is a paging interlock or duplicate page problem subsequent to an abend in ASM, the two are probably related and the first error provides information useful in debugging the second problem.

Recovery ignores invalid control blocks and truncates some of ASM's internal queues in order to allow ASM to continue processing. Therefore, recovery will cover up valid problems that cause code overlays in ASM and other system components.

The primary culprit in covering up errors is the non-historical nature of ASM resource queues that results in rapid reuses of critical control blocks. The only valuable information left by the recovery is the SDWA with its variable recording area in the SYS1.LOGREC data set. At the very least, this record provides sufficient information to trap the problem when it recurs.

Recovery Footprints

FRR/ESTAE Work Areas

ILRATA and ILREPATH are mapping macros that define the areas required by ASM modules to provide tracking information for the FRRs and ESTAEs.

- ILRATA defines the six-word parameter area passed to the ASM routine issuing the SETFRR macro, or it defines the parameter area passed to the ASM routine issuing the ESTAE macro. It contains a module ID in the first byte, flags in the next three bytes, and four words which have module-dependent contents. The IDs of the ASM modules are:

ID	Module	Entry	ID	Module	Entry
X'01'	ILRIODRV	ILRIODRV	X'0D'	ILRIODRV	ILREDRV
X'02'	ILRPAGCM	ILRPAGCM	X'0E'	ILRCMP	ILRCMPCI
X'03'	ILRSWAP	ILRSWAP	X'0F'	ILRCMP	ILRCMPNT
X'04'	ILRTRPAG	ILRTRPAG	X'10'	ILRIODRV	ILRSWLIO
X'05'	ILRSWPDR	ILRSWPDR	X'11'	ILRIODRV	ILRSWPIN
X'06'	ILRGOS	ILRGOS	X'12'	ILRIODRV	ILRSWAPO
X'07'	ILRIODRV	ILRIODR	X'13'	ILRCPBLD	ILRSIO
X'08'	ILRSRBC	ILRSRBC			
X'09'	ILRCMP	ILRCMPDI			
X'0A'	ILRCMP	ILRCMPNE			
X'0B'	ILRCMP	ILRCMPAE			
X'0C'	ILRCMP	ILRCMP			

- ILREPATH defines a variable-length area containing any additional recovery audit-trail data required for recovery by ASM recovery routines. The address of the EPATH, if present, is in the ATA. There are four variations of the EPATH area.

The formats of ILRATA (ASM tracking area - ATA) and ILREPATH (recovery audit trail area - EPATH) are described later in this chapter in the topic "ASM Recovery Control Blocks."

SDWA Variable Recording Area

ASM uses the SDWA variable recording area (SDWAVRA) to save the contents of the ATA (and EPATH, if present) upon entry to some of the recovery routines. This preserves the original state of the error before recovery took place. ILRIOFRR and ILRCMP01 save the ATA. ILRGOS01 and ILRDRV01 save the ATA and EPATH. ILRTM101 saves only the EPATH.

ASM Diagnostic Aids

This section contains diagnostic aids that are helpful in debugging problems in ASM. Topics included are:

- C0D ABEND Meanings for ASM
- ASM Recovery Control Blocks
- Additional ASM Data Areas

C0D ABEND Meanings for ASM

An ASM routine has found one of the following conditions which should not occur and has set the appropriate return code in the ASM tracking area (ATA):

RC 4 - The count of available swap sets for a specific swap data is non-zero but no available swap sets could be found.

RC 8 - The total count of available swap sets is non-zero but none of the swap data sets contain available swap sets.

RC 12 - The group operations starter has returned from one of the group operators but the ACE is not the first one on the LGE queue.

RC 16 - The memory termination resource manager for ASM has found that LSQA is not available for an address space that is abnormally terminating for one of the following reasons:

1. address space is not swapped in
2. address space is in process of being swapped in
3. RSMLSQA frame queue is unusable.

RC 20 - The ASM SRB controller has found an AIA or ACE on the LGE process queue which does not have the LPID converted flag on.

A software error record is written to SYS1.LOGREC and recovery processing continues.

ASM Recovery Control Blocks

During error recovery and cleanup processing, the ASM recovery routines communicate with other routines by using the ASM tracking area (ATA) and recovery audit trail area (EPATH).

ASM Tracking Area (ATA)

The ATA contains information necessary for the recovery or cleanup processing performed by the ASM recovery routines. The ATA is mapped to the six word work area returned by SETFRR when an FRR is established. For task mode routines, the ATA is mapped to the parameter area that is passed via the ESTAE macro.

The mapping macro name is: ILRATA.

Disp	Name	Size	Description
0	ATA	24	ASM Tracking Area
0	ATAMODID	1	ID of module establishing recovery routine. (See the previous topic "Recovery Foot- prints" for module IDs.)
1	ATASFLGS	3	Bit map representing logical sections of ASM routines; set to 1 on entry, set to 0 on exit.
	ATAIOPPR	800000	PROCPARE subroutine of ILRIODRV flag.
	ATASLSQA	400000	ILRSLSQA flag.
	ATASCOMP	200000	SWAPCOMP flag.
	ATAVIOCM	100000	ILRVIOCM flag.
	ATAPCOMP	080000	PAGECOMP flag.
	ATAPOS	040000	ILRPOS flag.
	ATAIOBSL	020000	BLOCKSEL subroutine of ILRIODRV flag.
	ATAPAGCM	010000	ILRPAGCM flag.
	ATASWAP	008000	ILRSWAP flag.
	ATATRPAG	004000	ILRTRPAG flag.
	ATASWPDR	002000	ILRSWPDR flag.
	ATACPBLD	001000	ILRCPBLD flag.
	ATAIOSSL	000800	SLOTSEL subroutine of ILRIODRV flag.
	ATAIOSCM	000400	STARTCOM subroutine of ILRIODRV flag.
	ATAIOMXA	000200	MIXAIA subroutine of ILRIODRV flag.
	ATAIOPF	000100	PGFLT subroutine of ILRIODRV flag.
The remaining flags are reserved:			
4	ATARFLGS	2	Other recovery flags.
	ATASGNST	8000	ILRSLSQA flag-in ASSIGNSET subroutine.
	ATASCCWP	4000	ILRSLSQA flag-in SCCWPROC subroutine.
	ATABADPK	2000	ILRCMPAE flag-in BADPACK subroutine.
	ATAPGVIO	1000	VIO flag.
The remaining flags are reserved:			
6	ATARCRSN	1	Recursion flags.
	ATARCRF1	80	Recursion flag-function 1.
	ATARCRF2	40	Recursion flag-function 2.
	ATARCRF3	20	Recursion flag-function 3.
	ATARCRF4	10	Recursion flag-function 4.
	ATARCRF5	08	Recursion flag-function 5.
	ATARCRF6	04	Recursion flag-function 6.
	ATARCRF7	02	Recursion flag-function 7.
	ATARCRF8	01	Recursion flag-function 8.
7	ATARCODE	1	Reason code for ASM-issued ABEND's.

The mapping of the remaining four words is dependent on the recovery routine involved.

For the recovery routine ILRIOFRR:

8	ATACLEAR	16	Maximum size of four-word area.
8	ATAAIA	4	Address of in-process AIA.
8	ATAACE	4	Address of in-process ACE.
C	ATAASCB	4	Address of in-process ASCB, or TRAS′d-to address space.
C	ATALGE	4	Address of in-process LGE.
C	ATAAIAQ	4	Address of AIA queue.

For the recovery routine ILRSWP01:

8	ATACLEAR	16	Definition allowing next four words to be cleared.
8	ATAAIA	4	Address of in-process AIA.
C	ATASARTE	4	Address of SART entry.
10	ATASCCW	4	Address of in-process SCCW.
14	ATAIORB	4	Address of in-process IORB.

For the recovery routine ILRGOS01:

8	ATAWORKA	4	Address of work-area cell.
C	ATAEPATH	4	Address of EPATH.

For the recovery routine ILRDRV01:

8	ATAAIAQ	4	Address of AIA queue to be processed.
C	ATAIOSB	4	IOSB checkpointed by ILRSIO.
10	ATAEPATH	4	ADDRESS of EPATH.

For the recovery routine ILRSRB01:

8	ATAAIACE	4	Address of in-process AIA/ACE.
C	ATAAIAQ	4	Address of AIA queue.
10	ATAACEQ	4	Address of ACE queue.
14	ATAEPATH	4	Address of EPATH.

For the recovery routine ILRCMP01:

8	ATAIOSB	4	Address of in-process IOSB.
C	ATAPCCWQ	4	Queue of PCCWs to be put back on PCCW available queue.
10	ATACOMPQ	4	Queue of AIAs to be returned to ILRPAGCM.
14	ATACPCCW	4	Address of in-process PCCW, not on IORB queue and not on ATAPCCWQ.

For the recovery routine ILRJTM01:

8	ATASAVE	4	Address of register save area.
8	ATAACEQ	4	Address of ACE queue.

For the recovery routine TERMRFRR:

8	ATARMPL	4	Address of RMPL, resource manager parameter list.
C	ATAWORKA	4	Address of work area.

Recovery Audit Trail Area (EPATH)

The EPATH is a communication area between the mainline routine and its corresponding recovery routine. The EPATH is necessary when the 6-word ATA is not large enough to accommodate the data to be tracked. The mapping of the EPATH is dependent on the recovery routine or mainline routine including the macro.

EPATH for ILRIODRV, ILRCPBLD, and recovery routine ILRDRV01:

Disp	Name	Size	Description
0	EPAAIA1	4	Input AIA chain.
4	EPAAIA2	4	PROCPARE AIA chain.
8	EPAAIA3	4	PARTESEL AIA chain.
C	EPAAIA4	4	STARTCOM AIA chain.
10	EPAAIA5	4	New NN VIO read AIA.
14	EPAAIA6	4	Reserved.
18	EPACPBLD	36	Channel program build input.
18	EPACPAIA	4	Start of AIA input to ILRCPBLD.
1C	EPACPENQ	4	Last AIA on ILRCPBLD AIA chain.
20	EPACPIOR	4	Primary IORB address.
24	EPACPIO2	4	Duplex IORB address.
28	EPACPWKA	4	Address of ILRCPBLD work area.
2C	EPABITS1	4	Used by ILRCPBLD.
30	EPACPRS1	4	Used by ILRCPBLD to build primary channel program.
34	EPABITS2	4	Used by ILRCPBLD.
38	EPACPRS2	4	Used by ILRCPBLD to build duplex channel program.

EPATH for VIO group operators and their recovery routines - ILRGOS, ILRSAV, ILRRLG, ILRACT, ILRVSAMI, ILRGOS01, ILRTMRLG, ILRTMI00, ILRTMI01, ILRSRBC, and ILRSRB01. ILRGOS01 is the recovery routine for ILRGOS which calls ILRSAV, ILRRLG, and ILRACT which call ILRVSAMI. ILRTMI01 is the recovery routine for ILRTMRLG which calls ILRVSAMI and ILRTMI00. ILRSRB01 is the recovery routine for ILRSRBC which calls ILRRLG. The first section is common because of the use of ILRVSAMI. The second section is dependent on the recovery routine involved.

Disp	Name	Size	Description
0	EPAOWKA	4	Group Operator's or ILRTMRLG's workarea address.
4	EPAVWKA	4	ILRVSAMI workarea address also points to RPL in workarea.
4	EPATMWKA	4	ILRTMI00 workarea address.
4	EPASWKA	4	ILRSRBC workarea address.
8	EPAAASP	4	Address of active ASPCT.
8	EPADSLST	4	Address of data set name list storage.
C	EPABASP	4	Address of buffer ASPCT.
C	EPATMIBA	4	Base address value for ILRTMI00.
10	EPARASP	4	Address of retrieved ASPCT.
10	EPATMACB	4	Address of storage used to build ACB for STGINDEX in ILRTMI00.
14	EPARTYRG	4	Address of 15-word save area containing retry registers R0-R14 for record-only abends.
14	EPABKSLT	4	Backing slots, only used for assign processing.
18	EPAFLAG1	1	Recovery flags.
	EPAVSAMI	X'80'	ILRVSAMI currently processing.
	EPAGRPOP	X'70'	One of group operators processing.
	EPARLG	X'40'	ILRRLG is currently processing.
	EPASAVE	X'20'	ILRSAV is currently processing.
	EPAACT	X'10'	ILRACT is currently processing.
	EPAACASR	X'08'	Activate or assign request.
	EPAASGN	X'04'	Assign processing - backing slots count (ASMBKSLT) has been updated.
	EPAUNSAV	X'02'	Mark slots unsaved in active ASPCT.
	EPARPLB	X'01'	RPL has been built.

19	EPAFLAG2	1	Recovery flags.
	EPATMXIT	X'80'	ILRTMI00 completed processing.
	EPAWARM	X'40'	ILRTMI00 warm start is processing.
	EPACOLD	X'20'	ILRTMI00 CVIOSTRT is processing.
	EPABUILD	X'10'	ILRTMI00 BUILDSNL is processing.
	EPAMAST	X'08'	Master scheduler initialization has been posted.
	EPATMI	X'04'	ILRTMI00 is currently processing.
	EPARECUR	X'02'	Recursion indicator for retry into mainline ILRTMRLG.
	*	X'01'	Reserved.

For ILRGOS01, ILRSAV, ILRACT, ILRRLG, ILRSRBC, and ILRSRB01:

Disp	Name	Size	Description
1A	EPALSIZE	2	Size of LGVT expansion.
1C	EPALGVTP	4	New LGVT address for LGVT expansion in ILRGOS.
20	EPALGEP	4	Logical group entry for request being processed.
24	EPASRB	4	Address of SRB for SRB controller.
28	EPAACE	4	Address of current ACE being processed.
2C	EPARBASP	4	Address of rebuilt ASPCT (LSQA).
30	EPARSIZE	2	LSQA block storage size for rebuilt ASPCT.
32	*	2	Reserved.
34	EPAERAIA	4	Queue of AIAs to be passed to ILRPAGCM.

For ILRTMI01 and ILRTMRLG, and ILRTMI00:

Disp	Name	Size	Description
1A	*	2	Reserved.
1C	EPAACE	4	Address of ACE currently being processed.
1C	EPAMSECB	4	Address of master scheduler initialization ECB.
20	EPATMRSV	4	Address of ILRTMRLG save area.
24	EPAABEND	4	Retry address for record-only abends.
24	EPATMIRT	4	Current retry address for failure in ILRTMI00.
28	EPATPART	4	Address of TPARTBLE while in ILRTMI00.

Additional ASM Data Areas

The following four ASM data areas (BSHEADER, BUFCONBK, DSNLIST, and MSGBUFFER) are not contained in *Data Areas*. For debugging ASM, BSHEADER (bad slot record) may be especially helpful.

BSHEADER

Acronym: BSHEADER.

Full Name: ASM error record (bad slots).

Macro ID: None.

Size: 1024 bytes.

Function: Trace table of the last 253 slots that ASM has found to be bad. Patterns of bad LSIDs can indicate where and what paging data sets are having difficulties.

Location: Pointed to by ASMVT (ASMEREC).

Offset	Length	Name	Description
0(0)	4	BSCURR	Current bad slot entry filled.
4(4)	4	BSFIRST	Beginning address of table.
8(8)	4	BSLAST	End address of table.
12(C)	1012	BSLIST	253 four-byte bad slot identifiers (LSIDs).

BSLIST entry

Offset	Length	Name	Description
0(0)	1	BSFLAG	
	1...		BSSPLSID if 1, LSID entry is swap. if 0, LSID entry is page.
	.... 1...		BSRDLSID if 1, LSID entry is for a read error. if 0, LSID entry is for a write error.
1(1)	3	BSTABNTY	LSID that is bad.

BUFCONBK

Acronym: BUFCONBK.

Full Name: VSAM buffer control block.

Macro ID: None.

Size: 12 bytes.

Function: Queue VIO group operation for later processing until VSAM resources are available.

Location: Pointed to by ASMVT (ASMGOSQS).

Offset	Length	Name	Description
0(0)	4	BUFCHAIN	Pointer to next BUFCONBK.
4(4)	4	BUFASCB	Pointer to ASCB.
8(8)	4	BUFACE	Pointer to ACE.

DSNLIST

Acronym: DSNLIST.

Full Name: Data Set Name List (ASM).

Macro ID: None.

Size: 44 times number of possible page/swap data sets. There are two DSNLISTs, one for page data sets and one for swap data sets.

Function: Make data set names available in non-fixed (pageable) storage.

Location: Pointed to by PART (PARTDSNL) for page data sets, and by SART (SARDSNL) for swap data sets.

Offset	Length	Name	Description
0(0)	44	DSNENTRY	Data set name left-justified and padded with blanks.

MSGBUFER

Acronym: MSGBUFER.

Full Name: ASM message buffer.

Macro ID: None.

Size: 376 bytes.

Function: Ensure that WTOR with LOGREC request will have a buffer to use.

Location: Pointed to by ASMVT (ASMMSGBF).

Offset	Length	Name	Description
0(0)	4	MSGCURR	Pointer to current buffer used.
4(4)	4	MSGFIRST	Pointer to first buffer.
8(8)	4	MSGLAST	Pointer to last buffer.
12(C)	4	MSGTERM	Pointer to special termination buffer.
16(10)	240	MSGBFRS	Three 80-byte buffers.
256(100)	120	MSGTBFR	Special termination buffer.

System Resources Manager (SRM)

The system resources manager (SRM) is a component of the MVS control program. It determines which, of all active address spaces should be given access to system resources, and the rate at which each address space is allowed to consume the resources.

An installation controls the MVS system primarily through the SRM. The evaluations and resulting decisions made by the SRM are dependent on the constants and parameters with which it is provided. The reader should understand the philosophy inherent in the use of these constants and parameters, so that their use will produce the desired effect. The *OS/VS2 System Programming Library: Initialization and Tuning Guide* provides the background information necessary to understand the controls available through the SRM, and the implementation of these controls.

SRM Objectives

The SRM bases its decision on two fundamental objectives:

1. To distribute system resources among individual address spaces in accordance with the installation's response, turnaround, and work priority requirements.

2. To achieve optimal system-throughput through use of system resources.

An installation specifies its requirements for the first objective in members of SYS1.PARMLIB called the installation performance specification (IPS) and the installation control specification. Through the IPS, the installation divides its types of work into distinct groups called *domains*, assigns relative importance to each domain, and describes sets of performance groups. Through the installation control specification, the installation can assign the correct performance characteristics to each address space. Another input to the SRM is the OPT member of SYS1.PARMLIB. Through a combination of the installation control specification, IPS, and OPT parameters, an installation can exercise a degree of control over system throughput characteristics.

When the need arises, trade-offs can be made between SRM's objectives. That is, the installation can specify whether, and under what circumstances, throughput considerations take priority over turnaround requirements. The SRM attempts to ensure optimal use of system resources by periodically monitoring and balancing resource utilization. If resources are under-utilized, the SRM attempts to increase the system load. If, on the other hand, resources are over-utilized, the SRM attempts to reduce the system load or to shift commitments to low-usage resources such as the processor, logical channels, auxiliary storage, and pageable real storage.

Address Space States

The SRM recognizes address spaces as being in one of three general states. Each state corresponds in concept to a queue on which SRM places the SRM user control block (OUCB) which describes the address space. These three states are:

1. In - The working set of an address space in this state occupies real storage.

2. Wait - The working set of an address space in this state may or may not occupy real storage. If the address space is logically swapped, the working set pages remain in real storage. If a physical swap takes place, the working set pages reside on auxiliary storage. An address space in this state has no ready work, is therefore incapable of executing, and not considered for swap-in.

3. Out - The working set of an address space in this state may or may not occupy real storage. If the address space is logically swapped, the working set pages remain in real storage. If a physical swap takes place, the working set pages reside on auxiliary storage. However, the address space is capable of executing and can be considered for swapping-in. For a logically swapped address space, the swap-in is reduced to a restore operation by RCT because no physical transfer of pages has occurred.

It is important to recognize that the correspondence between these states and presence on the associated queue is not precise; an address space can be in transit between two states (for example, it may be in the process of being swapped-out). Thus, the presence on a particular queue might not exactly mirror the physical state of affairs. Further, these classes are necessarily broad, and SRM recognizes subclasses; this is especially true among address spaces belonging to the "In" class. The use of the swap transition flags, in conjunction with the presence of an OUCB on a particular queue, mirrors the exact physical state of an address space. For wait state analysis, the exact state of given address spaces is important. If you can determine precisely what state SRM considers the various address spaces to be in, and the reasons why, you will gain insight for further analysis. The OUCB is the primary address-space-related control block in which much of the above information can be found.

OUCBs on the above three queues can be formatted via the SRMDATA format control statement of AMDPRDMP. The domain table (DMDT) is also formatted by SRMDATA and indicates the swapping status of the system.

In the OUCBQFL field (OUCB+X'01'), when the OUCBGOB bit is on, the SRM's OUCB repositioning routine is to be invoked. The destination of this pending OUCB repositioning is indicated by the following bit settings:

1. OUCBOUT='0'B - The OUCB will be placed on the "In" queue.

2. OUCBOUT='1'B and OUCBOFF='1'B - The OUCB will be placed on the "Wait" queue.

3. OUCBOUT='1'B and OUCBOFF='0'B - The OUCB will be placed on the "Out" queue.

When the repositioning is completed, the OUCBGOB bit is turned off; the setting of the OUCBOUT and OUCBOFF bits indicates the location of the OUCB.

A logically swapped address space can be identified by the OUCB being on the wait queue or the out queue and OUCBLSW = '1'B.

The setting of the swap transition flags for swap-out processing occurs in the following order:

1. If swap-out is initiated successfully, the OUCBGOO bit is set.
2. At quiesce-complete time, the repositioning of the OUCB takes place.
3. At swap-out-complete time, the OUCBGOO bit it turned off.

The setting of the swap transition flags for swap-in processing occurs in the following order:

1. If swap-in is initiated successfully, the OUCBGOI bit is set.

2. At swap-in status II time, the repositioning of the OUCB takes place and the OUCBGOI bit is turned off.

SRM Indicators

It is helpful to understand how SRM views the total MVS system, as well as the individual address spaces. This understanding can assist you in further problem analysis, especially of enabled wait state situations. A discussion of some of the SRM system and individual user indicators follows. Figure 5-32 shows the relationships among important SRM control blocks and queues.

A study of several counters and flags aids in further understanding of SRM processing. The counters and flags that pertain to the entire system are located in the SRM constants module (IRARMCNS), which resides in the nucleus. The counters and flags that pertain to a specific user are found in that user's OUCB.

System Indicators

The SRM control table (RMCT) is located at the start of module IRARMCNS. This address is found in field CVTOPCTP of the CVT+X'25C'. Generally, when SRM is in control, the address of the RMCT is contained in register 2. In the module IRARMCNS, the following fields provide information concerning SRM's current processing:

MCTAVQ1 This bit indicates that the count of available pages has fallen below the PVTAFCLO value, so the real storage manager (RSM) has called SRM to steal pages in order to increase the count of available pages. If this bit is on, it could indicate a normal condition.

MCTSQA1 This bit indicates that the number of available SQA pages is critically low. If MCTSMS1 is 1, the operator was notified of this situation.

MCTSQA2 This bit indicates that the number of available SQA pages has fallen below a second, more critical threshold than the one noted above. If MCTSMS2 is 1, the operator was notified of this situation.

MCTASM1 This bit indicates that the SRM has detected that less than 30% of all local slots are available. The SRM has informed the operator of this fact and has taken appropriate action to relieve the shortage.

MCTAMS2 This bit indicates that the SRM has detected that less than 15% of the total local auxiliary storage slots are available. The SRM has informed the operator of the slot shortage, and has taken appropriate action to relieve the shortage.

MCTFAVQ When this bit is on, a pageable storage shortage condition has been detected by SRM or RSM. If bit MCTPHPSS is also on, the shortage was detected by RSM because the count of fixed frames (PVTCNTFX) exceeded the threshold in PVTMAXFX. If bit MCTLGPSS is on, the shortage was detected by SRM because the sum of the count of fixed frames and the number of page I/Os in progress to the page data sets exceeded the threshold in MCCMAXFX.

MCTLGAVQ If this bit is on, SRM has increased the thresholds in PVTAFCLO and PVTAFCOK in order to cause the frame stealing necessary to swap-in an address space. This bit indicates that SRM has initiated the steal processing rather than waiting for an ANQLOW SYSEVENT from RSM.

MCVTWSS This halfword contains the target working set size for the common area. SRM attempts to keep this minimum number of frames assigned to the common area.

RCVUICA
RCVCPUA
RCVDPR
RCVMSP

These halfword values are the system contention indicators that the resource monitor examined for the last interval. They represent, in the order given: the average high referenced interval count (UIC), the average processor utilization, the demand paging rate, and the page delay time (in milliseconds). Based on these values, the target MPL for a domain might be altered.

RCVFXIOP This halfword contains the average percent of frames that are fixed or used for page I/O.

RCVMFXA This halfword contains the average percent of frames eligible to be fixed that are fixed.

RMCAINUS This halfword indicates the count of address spaces currently residing in storage. This count includes non-swappable address spaces. If this count is high, look at the next field.

CCVENQCT This halfword indicates the count of address spaces currently residing in storage and marked non-swappable because they are holding ENQ resources that other address spaces want.

LSCTCNT This fullword contains a count of the number of address spaces currently in the logically swapped state and swapped for terminal wait.

LSCTCNTW This halfword contains a count of the number of address spaces currently in the logically swapped state and swapped for a long or detected wait.

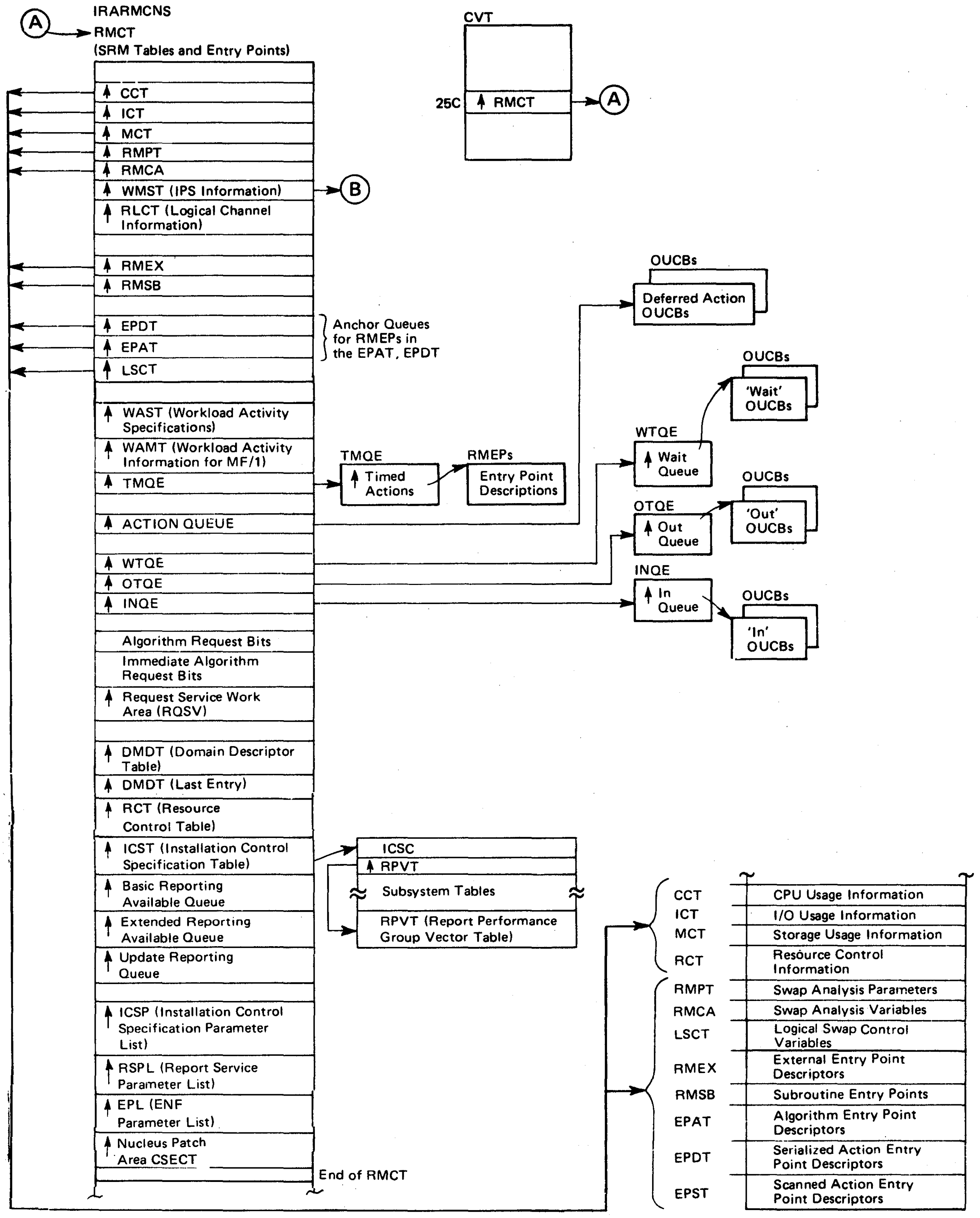

A
IRARMCNS
RMCT
(SRM Tables and Entry Points)
CCT
ICT
MCT
RMPT
RMCA
WMST (IPS Information)
B
RLCT (Logical Channel Information)
RMEX
RMSB
EPDT
EPAT
LSCT
Anchor Queues for RMEPs in the EPAT, EPDT
WAST (Workload Activity Specifications)
WAMT (Workload Activity Information for MF/1)
TMQE
ACTION QUEUE
WTQE
OTQE
INQE
Algorithm Request Bits
Immediate Algorithm Request Bits
Request Service Work Area (RQSV)
DMDT (Domain Descriptor Table)
DMDT (Last Entry)
RCT (Resource Control Table)
ICST (Installation Control Specification Table)
Basic Reporting Available Queue
Extended Reporting Available Queue
Update Reporting Queue
ICSP (Installation Control Specification Parameter List)
RSPL (Report Service Parameter List)
EPL (ENF Parameter List)
Nucleus Patch Area CSECT
End of RMCT
CVT
25C
RMCT
A
OUCBs
Deferred Action OUCBs
OUCBs
'Wait' OUCBs
WTQE
Wait Queue
TMQE
Timed Actions
RMEPs
Entry Point Descriptions
OUCBs
OTQE
Out Queue
'Out' OUCBs
INQE
In Queue
OUCBs
'In' OUCBs
ICSC
RPVT
Subsystem Tables
RPVT (Report Performance Group Vector Table)
CCT
ICT
MCT
RCT
RMPT
RMCA
LSCT
RMEX
RMSB
EPAT
EPDT
EPST
CPU Usage Information
I/O Usage Information
Storage Usage Information
Resource Control Information
Swap Analysis Parameters
Swap Analysis Variables
Logical Swap Control Variables
External Entry Point Descriptors
Subroutine Entry Points
Algorithm Entry Point Descriptors
Serialized Action Entry Point Descriptors
Scanned Action Entry Point Descriptors

Figure 5-32 (Part 1 of 2). SRM Control Block Overview

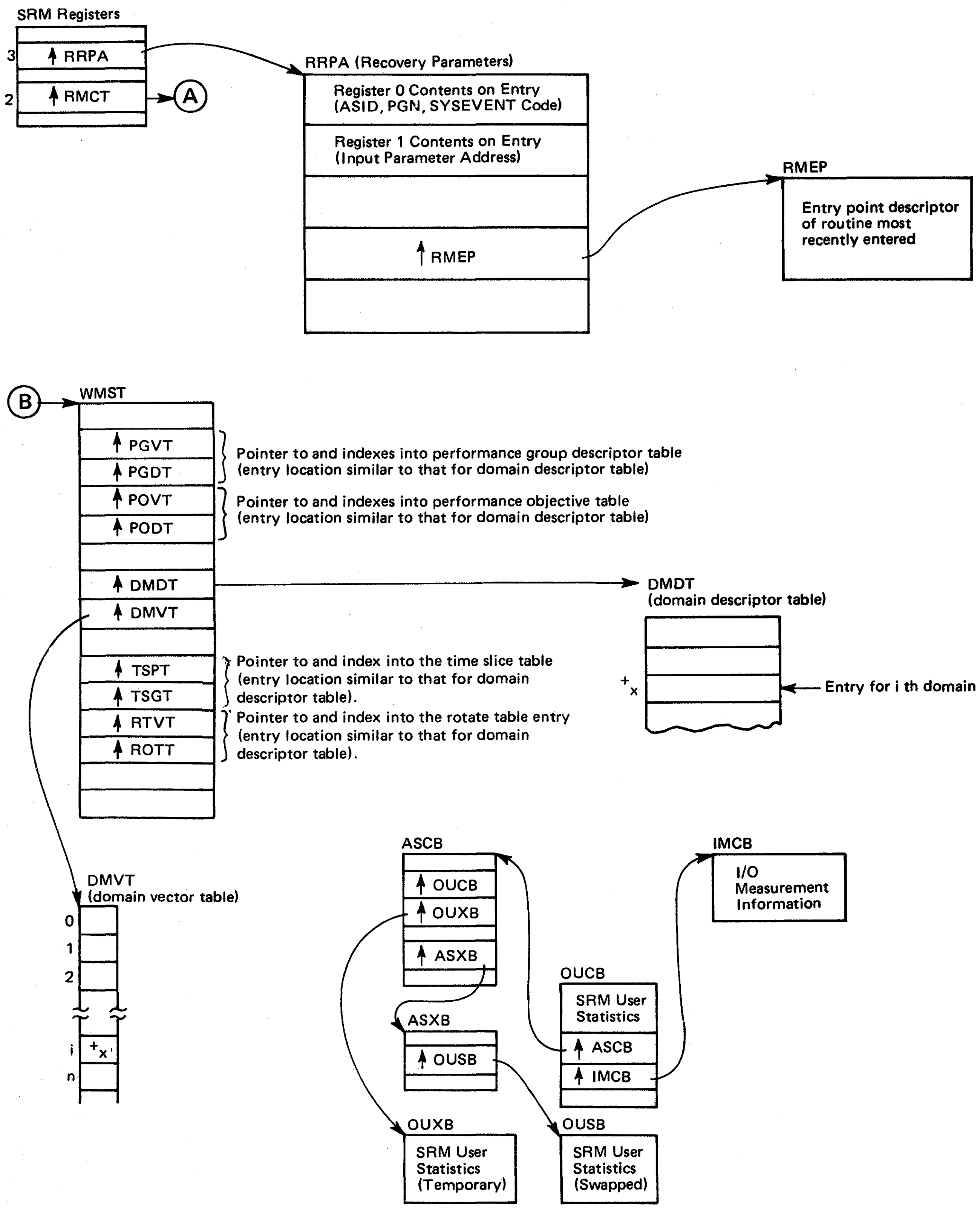

SRM Registers
3
RRPA
2
RMCT
A
RRPA (Recovery Parameters)
Register 0 Contents on Entry
(ASID, PGN, SYSEVENT Code)
Register 1 Contents on Entry
(Input Parameter Address)
RMEP
RMEP
Entry point descriptor
of routine most
recently entered
B
WMST
PGVT
PGDT
Pointer to and indexes into performance group descriptor table
(entry location similar to that for domain descriptor table)
POVT
PODT
Pointer to and indexes into performance objective table
(entry location similar to that for domain descriptor table)
DMDT
DMVT
DMDT
(domain descriptor table)
+x
Entry for i th domain
TSPT
TSGT
Pointer to and index into the time slice table
(entry location similar to that for domain
descriptor table).
RTVT
ROTT
Pointer to and index into the rotate table entry
(entry location similar to that for domain
descriptor table).
DMVT
(domain vector table)
0
1
2
i
+x
n
ASCB
OUCB
OUXB
ASXB
IMCB
I/O
Measurement
Information
OUCB
SRM User
Statistics
ASCB
IMCB
ASXB
OUSB
OUXB
SRM User
Statistics
(Temporary)
OUSB
SRM User
Statistics
(Swapped)

Figure 5-32 (Part 2 of 2). SRM Control Block Overview

Individual User Indicators

The register conventions generally used by SRM to process individual user functions can help you locate important SRM control blocks:

Register	Contents
2	Address of the RMCT
3	Address of the RRPA
4	Address of the OUCB
5	Address of the ASCB (if used by the requested SYSEVENT)

The SRM user control block (OUCB) contains flags and counters to provide information about a specific user. There is one OUCB for each address space, pointed to by ASCBOUCB (ASCB + X'90').

The following fields help in the understanding of specific user characteristics.

OUCBLSW When this bit is on, the address space is in the logically swapped state.

OUCBMWT If this bit is on, the SRM has detected that this user has not been dispatched, but was occupying storage for at least delta seconds. This interval is processor-model dependent. The user will be swapped-out until the dispatcher informs SRM that the address space has work to do.

OUCBAXS When this bit is on, the user has been swapped-out of storage because the user's address space was obtaining auxiliary storage slots at the fastest rate in the system when an ASM slot shortage occurred.

OUCBENQ This bit indicates that a different address space has tried to ENQ on a resource held by this address space. This user is treated as non-swappable for an installation-defined time period.

OUCBYFL See specific bit designations below:

- Bit 1 - indicates that the user was created via a START command.
- Bit 2 - indicates that the user was created via a TSO LOGON command.
- Bit 3 - indicates that the user was created via a MOUNT command.

OUCBCSFS If this bit is on, the user is being delayed. Either swap-in has failed for this address space due to a lack of available storage, or the user was swapped because of a pageable frame shortage.

OUCBFXS This bit indicates that the address space was selected for swap-out in order to relieve a pageable storage shortage condition. If bit OUCBLGFX is also on, the address space had more frames allocated to it than any other swappable address space when SRM detected the pageable storage shortage. If OUCBLGFX is off, the address space had more fixed frames than an average address space when RSM detected the shortage.

OUCBDFSW If this bit is on, swap-in has been delayed. The PVTAFCLO and PVTAFCOK fields have been increased by the number of frames needed to complete the swap-in.

OUCBJSAS When this bit is on, it indicates that, at the time of job select processing for this user, there was an auxiliary slot shortage. This user's initiation is being delayed until the shortage is relieved.

OUCBJSFS When this bit is on, it indicates that there was a pageable frame shortage at the time of job select processing for this user. This user's initiation is being delayed until the shortage is relieved.

OUCBSRC This field contains a code describing why this user was last swapped-out. The codes are:

01 - Terminal output wait
02 - Terminal input wait
03 - Long wait
04 - Auxiliary storage shortage
05 - Real storage shortage
06 - Detected wait
07 - Reqswap SYSEVENT issued
08 - ENQ exchange by swap analysis
09 - Exchange based on recommendation values by swap analysis
0A - Unilateral swapout by swap analysis.

OUCBRDY This bit indicates that ready work became available for this address space which was swapped-out due to a wait. The address space is now capable of executing and is a candidate for swap-in.

OUCBTWSS This halfword contains the target working set size for the address space. SRM attempts to keep this minimum number of frames assigned to the address space.

OUCBHOLD This fullword contains a count of outstanding "Hold" sysevents issued by this address space. A non-zero count will result in quiesce turning the swap-out around and restoring the address space.

Other Indicators

The SRM domain descriptor table can be useful in pinpointing a problem involving SRM's MPL control. Mapping of the table can reveal why a user is kept out of main storage, why erratic response time occurs, and other user and system information.

SRM Error Recovery

SRM maintains two functional recovery routines (FRRs) that are located in IRARMERR. One FRR (recovery routine 1 - RR1) gets control whenever errors occur after SRM is branch-entered by a routine that holds a lock higher in the lock hierarchy than the SRM lock. The other FRR (recovery routine 2 - RR2) gets control whenever errors occur and SRM is running with the SRM lock.

If it is suspected that SRM is entering error recovery and a stop is necessary at the time of error, RMRR2INT is a subroutine common to both RR1 and RR2.

Recovery routine 1 (RR1) retries if a retry routine exists. If no routine exists, or if the error recurs, RR1 percolates the error.

With recovery routine 2 (RR2), many special situations such as the following are first checked:

- Is RMF active and should it be terminated?
- Is SET IPS active and should abend code be converted?
- Is OUCB valid and should abend code be converted?

Then RR2 retries if a retry routine exists. If no retry routine exists, or if the error recurs, RR2 percolates the error.

SRM SDWA Data

When either FRR is entered, the FRR fills in the SDWA fields prior to scheduling the SVC dump so that the dump matches the SYS1.LOGREC entry. However, in some cases, note that the FRR changes the abend code or reason code after the dump is scheduled and before the LOGREC entry is written, which results in the LOGREC entry reflecting a different code than the dump.

The FRR also puts problem determination data into the SDWA variable recording area (SDWAVRA) in key-length-data format using standard keys. (See the SDWAVRA area in the *Debugging Handbook* for a description of the keys.) Additional information is provided for several of the fields as follows:

Key **Contents**

VRAEBC The EBCDIC message "IRARMCNS OFFSET TO CURR RTNE PTR IS xxx," where xxx is the hexadecimal offset into the nucleus module IRARMCNS that contains the entry point address of either the SRM routine that was in control at the time of the error, or, if a subroutine was in control, the routine that called the subroutine.

VRARRP A copy of the recovery routine parameter area (RRPA). The RRPA contains status information used on exit from the SRM and during SRM recovery processing. Note that the low-order byte in the first word in the RRPA contains the SYSEVENT code of the original entry to SRM. The format of the RRPA can be found in the IRARRPA mapping macro.

VRAFP A copy of the RRPA (as in field VRARRP)) but with several entries cleared. Entries are cleared because they can be different for different invocations of the same function. The VRAFP is SRM's footprint area used for recognizing duplicate problems.

Module Entry Point Summaries

For a description of SRM modules and entry points, refer to *OS/VS2 System Logic Library.*

VTAM

Note to Readers

Component information for MVS VTAM is deleted from this book because it is obsolete.

For diagnostic information for ACF/VTAM, see the following:

- *ACF/VTAM Diagnosis Guide*
- *ACF/VTAM Diagnosis Reference*

VSAM

The virtual storage access method (VSAM) consists of three major subcomponents:

- Record management
- Open/close/end-of-volume
- I/O manager

Record Management

Record management processing produces no messages. Problem determination normally begins with an examination of the request parameter list (RPL). If a physical error occurs and the user has provided a large enough message area (pointed to by RPLERMSA), VSAM (IDA019R5) builds a SYNADAF-type record in that area for the user to examine. For both logical and physical errors, VSAM sets return codes in the RPL.

RPL

Three fields in the RPL are used to indicate an error:

1. RPLERREG - (RPL+X'D') a one-byte value which is also returned in register 15 after a request:

   0 - request completed normally
   8 - a logical error occurred
   12 - a physical error occurred.

2. RPLCMPON - (RPL+X'E') a one-byte value that indicates which component was being processed at the time of the error if the request involved alternate indexes. This value also indicates whether upgrading was valid or was incorrect because of the error.

Code	Component	Status of Upgrade
0	base cluster	valid
1	base cluster	might be incorrect
2	alternate index	valid
3	alternate index	might be incorrect
4	upgrade set	valid
5	upgrade set	might be incorrect

3. RPLERRCD - (RPL+X'F') a one-byte value describing the error (see the Diagnostic Aids section of *OS/VS2 VSAM Logic).*

Other important fields in the RPL are:

RPLREQ -	(+X'02') request type
RPLPLHPT -	(+X'04') pointer to the PLH
RPLECB -	(+X'08') ECB or pointer to the ECB
RPLDACB -	(+X'18') pointer to the ACB
RPLAREA -	(+X'20') pointer to the user's record area
RPLARG -	(+X'24') pointer to the user's search argument
RPLOPTCD -	(+X'28') two bytes of option flags
RPLDDDD -	(+X'40') last successful request's RBA value (returned to user by VSAM).

PLH

Once the information in the RPL has been evaluated, the next block to examine is the placeholder (PLH). The PLH contains current information about the request, including positioning and pointers to associated control blocks such as buffer control blocks (BUFCs) and the I/O management block (IOMB).

The following fields are important for understanding the request:

PLHFLG1 - (+X‘02’) status flags

PLHFLG2 - (+X‘03’) status flags

PLHEFLGS - (+X‘04’) two bytes of exception flags

PLHFLG3 - (+X‘06’) status flags

PLHAFLG3 - (+X‘07’) status flags

PLHCRPL - (+X‘14’) pointer to the current RPL

PLHDBUFC - (+X‘34’) pointer to the current data BUFC

PLHDIOB - (+X‘4C’) pointer to IOMB

PLHRET0 - (+X‘74’) halfword offset into register 14 pushdown save area. If the halfword at +X‘76’ is zero, PLHRET0 is an offset from +X‘78’ into a 14-word save area and points to the next available word. If the halfword at +X‘76’ is not zero, then it is the offset from +X‘78’ to the beginning of a 20-word save area at the end of the PLH, and PLHRET0 is an offset from +X‘78’ into that save area.

PLHIBUFC - (+X‘BC’) pointer to the current index BUFC

PLHIXSPL - (+X‘C8’) 32-byte index search parameter list (IXSPL) containing information about the results of the last index search.

PLHKEYPT - (+X‘F8’) pointer to the current key value or relative record number.

BUFC

The buffer control block (BUFC) contains function codes, status indicators, and relative byte address (RBA) values describing the associated buffer.

BUFFLG1 - (+X‘01’) BUFC status flags
BUFCIOFL - (+X‘02’) I/O status flags
BUFCDDDD - (+X‘08’) RBA for input if BUFCVAL is on
BUFCORBA - (+X‘0C’) RBA for output if BUFCMW is on
BUFCBAD - (+X‘14’) pointer to associated buffer

During record management processing, register usage is as follows:

R1 - RPL pointer
R2 - PLH pointer
R3 - pointer to the access method block (AMB) of the component being processed
R4 - BUFC pointer

Use the register 14 save area in the PLH to find the path taken by a request through record management.

Record Management Debugging Aids

It is not always desirable to cause program checks as a method of getting dumps, because some applications have sophisticated error recovery routines that can possibly change the environment. It is preferable to get documentation of the error before such routines get control, and then allow these routines to do their cleanup function after the dump is taken. The following code is an example of a console-activated communications vector table (CVT) trap for record management errors that causes the failing application to loop, allowing a console dump to be taken. Following the dump the trap can be deactivated, allowing the application to continue processing. The code can be inserted into CSECT IDA019R1 at label 'POSTRPL', label 'POSTRPL2', and the patch area at the end of the module.

```
NAME IDA019L1     IDA019R1

VER POSTRPL'      950C,100D

VER POSTRPL2'     1851,9101,1028

VER PATCH         0000,0000             X'54' bytes of patch area

REP POSTRPL'      45E0,Bxxx             to PATCH1

REP POSTRPL2'     1851,45E0,Bxxx        to PATCH2

REP PATCH1        58F0,0010,            point to CVT

    LOOP1         9102,F108,            is trap activated?
                  4780,Bxxx,            no, go to EXIT1
                  D500,F10A,100D,       compare error type
                  4770,Bxxx,            no, go to EXIT1
                  D500,F10B,100F,       compare error code
                  4770,Bxxx,            no, go to EXIT1
                  47F0,Bxxx,            yes, go to LOOP1. Loop until trap bit in CVT is
                                        turned off.

    EXIT1         950C,100D,            restore instruction
                  07FE,                 branch back inline

    PATCH2        58F0,0010,            point to CVT

    LOOP2         9102,F108,            is trap activated?
                  4780,Bxxx,            no, go to EXIT2
                  D500,F10A,100D        compare error type
                  4770,Bxxx,            no, go to EXIT2
                  D500,F10B,100F,       compare error code
                  4770,Bxxx,            no, go to EXIT2
                  47F0,Bxxx,            yes, go to LOOP2. Loop until trap bit in CVT is
                                        turned off.

    EXIT2         9101,1028,            restore instruction
                  07FE                  branch back inline
```

To activate the trap, set CVT+X'10A-10B' to logical error (X'08xx') where xx is the error code (RPLERRCD), or to physical error (X'0C00'). Then 'OR' on bit 6 (X'02') in CVT+X'108' taking care to leave the other bits in that byte undisturbed. After the loop occurs and a console dump of the failing address space has been taken, turn off bit 6 in CVT+X'108' to deactivate the trap and allow the application to continue processing. Be sure that the dump taken includes the region, SQA, and CSA. Note that when using the trap for physical errors the RPLERRCD is X'00' at the point of the trap because VSAM has not yet gone to IDA019R5. Physical errors caused by unit check (for example -

incorrect length, no record found on a search id, require that the I/O supervisor block (IOSB) be examined. To get a dump with the IOSB still valid, a trap can be inserted into nucleus CSECT IDA121A4 (abnormal end appendage) at label 'PERMERR'. Since this is in the nucleus, the trap can be set from the console. (See I/O Manager Debugging.)

Record management error codes (RPLERRCD) are described in the Diagnostic Aids section of *OS/VS2 VSAM Logic*. It is useful to know which module sets each error and the name of each error, so that you can find where it is set in the module via the cross reference.

Error Code (hex)	**Symbolic Name**	**Module (IDA019xx)**
Logical		
04	RPLEODER	RD, RR, RY, R2, R4, R8
08	RPLDUP	RA, RQ, RX, R4
0C	RPLSEQCK	RA, RR, RX, R4
10	RPLNOREC	RA, RR, RY
14	RPLEXCL	RF, RY, R2, R8
18	RPLNOMNT	RW, RY, R2, R5
1C	RPLNOEXT	RE, RF, RM, R5, R8
20	RPLINRBA	RA, R8
24	RPLNOKR	RM
28	RPLNOVRT	RG, RU, RX
2C	RPLINBUF	RR, RT, RY, R4, R8
40	RPLNOPLII	RU, RX, R1
44	RPLINACC	RQ, R4, R8
48	RPLINKEY	R1, R8
4C	RPLINADR	R1, R8
50	RPLERSER	RL, RX, R8
54	RPLINLOC	RQ, R1, R4, R8
58	RPLNOPTR	RD, RR, R4, R8
5C	RPLINUPD	RQ, RX, R4, R8
60	RPLKEYCH	RL, RX
64	RPLDLCER	RL, RQ
68	RPLINVP	RA, RR, RY, RX, R1, R4, R8
6C	RPLINLEN	RL, RQ, RU, R4, R8
70	RPLKEYLC	R1
74	RPLINLRQ	RR, R4, R8
78	RPLINTCB	RP
84	RPLSRLOC	RT
88	RPLARSRK	RT
8C	RPLSRISG	R4
90	RPLNBRCD	RX
94	RPLNXPTR	RU
98	RPLNOBFR	RY
C0	RPLIRRNO	RQ, RR
C4	RPLRRADR	R1
C8	RPLPAACI	RX
CC	RPLPUTBK	RQ, R4
D0	RPLINVEQ	RP
Physical		
04	RPLRDERD	R5
08	RPLRDERI	R5
0C	RPLRDERS	R5
10	RPLWTERD	R5
14	RPLWTERI	R5
18	RPLWTERS	R5

Record management processing sometimes requires serialization of internal resources. When the needed resource can be acquired, processing proceeds normally. However, when another request has control of the resource the request is deferred. As each request completes, a scan is made for requests which have been deferred. If the resource has become available, the deferred request is restarted. While a request is deferred, PLHDRPND is set in the PLH and PLHDRRSC points to the resource byte to be tested for availability.

Open/Close/End-Of-Volume

O/C/EOV documents errors by means of error messages and access method control block (ACB) return codes. The codes returned in the ACB (ACBERFLG) are explained in the Diagnostic Aids section of *OS/VS2 VSAM Logic,* along with an indication of the modules that set each error. In the cross reference of the modules, these error codes have the symbolic name of OPERRddd, where ddd is the decimal error code. The most significant problem determination feature of O/C/EOV however, is its message facility. The following messages are issued:

MSGIEC070I - END OF VOLUME
MSGIEC161I - OPEN
MSGIEC251I - CLOSE
MSGIEC252I - CLOSE (TYPE=T)

The messages contain both problem codes (symbolic PPddd) and function codes (symbolic PDFddd). The problem codes that describe the error are explained with each message in *System Messages.* The function codes are described best in the Diagnostic Aids section of *OS/VS2 VSAM Logic,* along with the module that was performing the the function at the time of the error.

O/C/EOV Debugging Aids

There is a built-in trap for O/C/EOV (see the **Caution** later in this topic). There are two bits involved. Bit 4 (X'08') at CVT+X'108' can be OR'd on (being careful to leave the other bits in that byte undisturbed) to cause an abend dump (U888) when the message is issued. Bit 6 (X'02') at CVT+X'10A' when turned on prevents the freeing of module work areas. When both these bits are on, the U888 dump produced contains the module work area for every module gone through in the open path. There is a discussion in the Diagnostic Aids section of *OS/VS2 VSAM Logic* on finding the work areas in the dump and a diagram showing how the work areas are chained together.

GTF trace is also available for debugging. If GTF is active for TRACE=USR at the time of the error, VSAM Open (IDA0192P) writes user records FFF and FF5 containing the VSAM control blocks at the time of the failure. The standard OPEN work area trace is also available by coding AMP='TRACE' on the DD statement.

The following ENQs are issued by O/C/EOV:

Major Name	**Minor Name** (Note 1)	**Modules**	**Reason**
SYSVSAM	NNNCCCCB	IDA0192A IDA0200T IDA0231T IDA0557A	The 'B' or busy ENQ is used to serialize the modification of the control block chains by allowing only one of the functions (OPEN, CLOSE, TCLOSE, or END of VOLUME) to process the data set. This resource is held for the life of the function.
SYSVSAM	NNNCCCCI	IDA0192A	The 'I' ENQ is issued for each component of a data set being opened for input processing. DEQ is issued when the data set is closed.
SYSVSAM	NNNCCCCO	IDA0192A	The 'O' ENQ is issued for each component of a data set being opened for output processing. DEQ is issued when the data set is closed.

Note: If the data set is opened for both input and output, both the 'I' and 'O' resources will be held for each component.

Note 1: In the minor name, NNN = the 3-byte CI number of the component's catalog record
CCCC = the 4-byte catalog ACB address.

When a VSAM (non-catalog) ACB is opened, data extent blocks (DEBs) are constructed and chained as follows:

- A DEB containing the data set ACB address at DEB+X'18' is chained on the DEB chain of the current TCB. This DEB is referred to as the '*dummy*' DEB. Its purpose is to allow abend to close the VSAM data set if abnormal termination occurs.
- A DEB containing the component access method block (AMB) address at DEB+X'18' is chained on the DEB chain of the jobstep TCB for each component being opened. These are the '*real*' DEBs and are the ones actually used by VSAM processing.

When an ACB is being opened for DSNAME or DDNAME sharing and the data set is already open, the ACB is just connected to the existing control block structure and only the '*dummy*' DEB is built and chained on the current TCB.

Caution: When using the O/C/EOV trap be aware that:

- If the bit is turned on to prevent the freeing of work areas and the job causes many calls to O/C/EOV, the region size may have to be increased to prevent ABEND80A.
- JOBCATs and STEPCATs are opened under the initiator TCB. The work area core is owned by the initiator TCB. If this core is not freed because the CVT debug bit is on, the initiator may get an ABEND20A when it issues FREEMAIN for subpool 247 at job termination.

I/O Manager

I/O management includes the following modules:

IDA019R3	Problem state I/O driver; a CSECT of LPA load module IDA019L1
IGC121	Supervisor state I/O driver (SIOD); a CSECT in the nucleus
IDA121A2	Actual block processor (ABP); a CSECT in the nucleus
IDA121A3	Channel end appendage; a CSECT in the nucleus
IDA121A4	Abnormal end appendage; a CSECT in the nucleus

The drivers and the ABP translate requests for access to the contents of control intervals into requests for reading and writing physical records. They also build the channel program to be passed to IOS.

I/O Manager Debugging

The combination of the I/O management block (IOMB), the I/O supervisor block (IOSB), and the service request block (SRB), is used by I/O management to control the processing of a request. The PLH (PLHIOB) points to the IOMB. The IOMB points to the IOSB (IOMIOSB), which in turn points to the SRB (IOSSRB).

For debugging unit checks (for example: no record found, incorrect length, channel program check, channel protection check) the best place to trap for a dump is at label 'PERMERR' in nucleus csect IDA121A4.

Catalog Management

Catalog management manages system requests for references and updates to the master catalog. The following description of catalog management includes these topics:

- Major Registers and Control Blocks
- Module Structure
- VSAM Catalog Recovery Logic
- Debugging Hints

Major Registers and Control Blocks

This section describes the major catalog management registers and control blocks, shows how each can be located, and describes those control block fields and flags that have proven to be useful in debugging.

How to Find Registers

Catalog management runs under control of an SVRB. The registers are saved across supervisor-assisted linkages and interruptions in the standard ways. Depending upon the nature of the problem, the registers can usually be found in one of the following areas:

- For abends, registers are stored in RTM's SVRB and SDWA.

- For program checks, registers are stored in RTM's SVRB, the SDWA, and the LCCA.

- For catalog-management-issued type 2, 3, and 4 SVCs, registers are stored in the successor SVRB.

- For waits, registers are stored in the TCB.

The registers stored in any of these areas will be the registers that existed when the code that was running under a catalog SVRB gave up control. These registers will either be the registers of one of the three catalog management routines or the registers of a routine that was branch-entered by catalog management. If register 11 points to the CCA (identifiable via a X'ACCA' in the first word), the registers probably belong to IGG0CLA1; register 12 will be the base register for the CSECT last in control. Otherwise, if register 11 is a base register, the code that if references may be inspected to determine the routine in control. If the routine in control is one that was branch-entered by catalog management, then catalog management's registers may have been saved in a standard area pointed to by register 13.

Major Registers

IGC0002F

Register 11 -	Base register
Register 12 -	Work area pointer

IGC0CLA1

Register 11 -	CCA pointer
Register 12 -	Base register (current CSECT)
Register 13 -	Register save push down list pointer (see CCAREGS) or standard save area pointer

IGG0CLCA

Register 11 -	Base register
Register 12 -	Work area pointer

Major Control Blocks

The control blocks described in this section (AMCBS, PCCB, ACB, CAXWA, CTGPL and CCA) are those that are most useful from a debugging standpoint. The AMCBS and PCCB are useful in locating the control block structures for open catalogs. The ACB and CAXWA relate to a particular catalog or catalog recovery area (CRA) data set. The CTGPL and CCA relate to a particular catalog request.

AMCBS

The AMCBS (access method control block structure) is essentially a VSAM vector table. It is constructed within the SQA during early NIP processing (IEAVNP11) and resides there throughout the life of the system. The AMCBS is found through CVT+X'100' (field CVTCBSP). Major fields in the AMCBS are:

Field	Description
CBSACB	Pointer to the master catalog's ACB.
CBSCMP	Pointer to the IGG0CLA1 load module.
CBSCAXCN	CAXWA chain pointer. The CAXWAs of all currently open VSAM catalogs are included in this chain. The master catalog's CAXWA is the last CAXWA in this chain.

PCCB

A PCCB (private catalog control block) connects a VSAM user catalog to a particular initiator or job step. A PCCB is constructed (in SWA) for each user catalog opened during the life of a job step. PCCBs are chained together to form an initiator or job-step-oriented PCCB chain. Generally, PCCBs are freed by step termination. A PCCB is not required for the master catalog.

PCCBs are located through the TCB: TCB+X'B4' (field TCBJSCB) points to the JSCB; JSCB+X'15C' (field JSCBACT) points to the active JSCB; the active JSCB+X'CC' (field JSCBPCC) points to the first PCCB. PCCBs are chained via PCCNEXTP.

Major fields in a PCCB are:

Field	Description
PCCACRO	PCCB identifier ('PCCB').
PCCNEXTP	Pointer to the next PCCB. This field is 0 if it is the last PCCB.
PCCACBP	Pointer to the catalog's ACB.
PCCDSNAM	Catalog's name.
PCCTGCON	Catalog's alias name.

Major flags in a PCCB are:

Flag	Description
PCCSTEPC	The catalog was specified to the job step through the use of a JOBCAT or STEPCAT DD card.
PCCACTIV	The catalog is allocated and active.
PCOSCVOL	The catalog is an OS CVOL.

ACB

There is one ACB (access method control block) for each open VSAM catalog or CRA. The ACB is created by the routine that opens the data set. Catalog and CRA ACBs generally reside in the CSA.

An ACB can be located in the following ways:

1. The master catalog's ACB can be located from the AMCBS (CBSACB).

2. A particular user catalog's ACB can be located either via the CAXWA chain or via the PCCB chain. To locate the ACB via the CAXWA chain, inspect the CAXCNAM field of each CAXWA in turn until the desired catalog name is found. The first CAXWA is pointed to by the AMCBS (CBSCAXCN). The CAXWAs are chained via CAXCHN. When the desired CAXWA is found, it points to the desired ACB (CAXACB).

   To locate the ACB via the PCCB chain, inspect the PCCDSNAM and PCCTGCON fields of each PCCB in turn until the desired catalog name or alias name is found. The first PCCB is pointed to by the job step's active JSCB (JSCBPCC). The PCCBs are chained via PCCNEXTP. When the desired PCCB is found, it points to the desired ACB via PCCACBP.

3. A particular CRA's ACB can be located as follows:

   a. Find the owning catalog's ACB (via steps 1 or 2).

   b. Find the owning catalog's CAXWA (pointed to by ACBUAPTR).

   c. Find the first CRA's ACB (pointed to by CAXCRACB).

   d. Find the first CRA's CAXWA (pointed to by the CRA ACB's ACBUAPTR field at ACB+X'40').

   e. Inspect the CAXVOLID field for the desired CRA volume serial number.

f. If the desired CRA's ACB has not yet been found, then search the remaining CAXWAs in the CRA CAXWA chain. Inspect the CAXVOLID field of each remaining CRA CAXWA in turn until the desired CRA volume serial number is found. The remaining CRA CAXWAs are chained to the first CRA CAXWA (and to each other) via CAXCHN. When the desired CRA CAXWA is located, it points to the desired CRA ACB via CAXCRACB.

4. The ACB representing the VSAM catalog that is currently being processed by a particular catalog request can be located via the CCA (CCAACB).

5. The ACB representing the CRA that is currently being processed by a particular catalog request can be located via the CCA (CCARAACB).

Major fields in the ACB are:

Field	Description
ACBID	Control block identifier (X'A0').
ACBAMBL	Pointer to the VSAM record management control block structure. This set of control blocks is built at OPEN time, resides in CSA, and consists of those control blocks required to support a KSDS (catalog) or an ESDS (CRA).
ACBERFLG	Error code stored by OPEN or CLOSE when the operation is unsuccessful.
ACBUAPTR	Pointer to the CAXWA.

Major flags in the ACB are:

Flag	Description
ACBCAT	ACB represents a catalog.
ACBSCRA	ACB represents a CRA that has been opened for catalog management use.
ACBUCRA	ACB represents a CRA that has been opened for use by an access method services (AMS) utility function.

CAXWA

There is one CAXWA (catalog ACB extended work area) for each open catalog or CRA. The CAXWA is created during the OPEN process (either before the OPEN or by the catalog OPEN routines). CAXWAs generally reside in the CSA. The CAXWA is pointed to by the ACB (field ACBUAPTR). See step 3 for locating the ACB under the heading "ACB" earlier in this chapter. Major fields in the CAXWA are:

Field	Description
CAXID	Control block identifier (X'CA').
CAXCHN	Pointer to the next CAXWA in the CAXWA chain. This is 0 if it is the last CAXWA in the chain.
CAXACT	Count of the number of job steps for which this catalog is currently open.
CAXACB	Pointer to the catalog ACB.
CAXUCB	Pointer to the catalog's or CRA's UCB.

CAXRPL Pointer to a pool of RPLs. This pool is obtained at OPEN time and resides in CSA. (**Note:** This field is not used in CRA CAXWAs. CRA RPLs are included within the owning catalog's RPL pool.)

CAXCNAM Catalog name (for catalog CAXWA only).

CAXVOLID CRA volume serial number (for CRA CAXWA only).

CAXCRACB For a catalog CAXWA: pointer to the first CRA ACB. For a CRA CAXWA: pointer to the CRA ACB.

Major flags in the CAXWA are:

Flag	**Description**
CAXBLD	The catalog or CRA is in the process of being created.
CAXOPN	The catalog or CRA is being opened.
CAXCLS	The catalog or CRA is being closed.
CAXEOV	The catalog or CRA is being extended.
CAXMCT	The CAXWA represents the master catalog.
CAXF2DT	The catalog has been deleted.
CAXF2NDD	Unable to OPEN or CLOSE - DDNAME not found.
CAXF2NCR	Unable to OPEN or CLOSE - insufficient main storage.
CAXF2IOE	Unable to OPEN or CLOSE - I/O error.
CAXF2REC	The catalog is a recoverable catalog (catalog CAXWA only).

CTGPL

The CTGPL (catalog parameter list) is built by the routines that issue SVC 26 to represent the desired catalog management request. The storage area where this block resides varies and is controlled by the building routine. When a caller issues SVC 26, the caller's registers are saved in the SVRB under which catalog management operates. Register 1 of this SVRB's register save area points to the CTGPL. The CTGPL may also be located via the CCA (CCACPL).

Note: At times, catalog management processing uses CCACPL as a pointer to an internal CTGPL. Therefore, you should be careful when you use this pointer to locate the caller's CTGPL.

Major fields in the CTGPL are:

Field	**Description**
CTGOPT1 CTGOPT2 CTGOPT3 CTGOPT4 CTGOPTNS CTGTYPE	These fields contain the codes and flags that indicate the type of function requested.
CTGENT	Pointer to the entry name or CI number (for types of requests other than DEFINE or ALTER).
CTGFVT	Pointer to the field vector table (FVT) for DEFINE and ALTER requests.
CTGCAT	Pointer to an area that indicates the specific catalog (if any) to be used in processing this request. The area may contain either the catalog name or a pointer to the catalog's ACB. If no specific catalog is indicated, CTGCAT will be 0.
CTGWKA	Pointer to the work area. In general, catalog management stores the requested information into this area.
CTGNOFLD	Number of FPL pointers in CTGFIELD.
CTGFIELD	An array of 4-byte RPL pointers. The FPLs describe the data fields that the request is to process.

CCA

The CCA (catalog communications area) is the main VSAM catalog work area. It is built upon entry to the VSAM catalog processor and freed just before exit. The CCA resides in subpool 252 of the caller's address space. Register 11 points to the CCA.

Major fields in the CCA are:

Field	**Description**
CCAID	Control block identifier (X'ACCA').
CCAPROB	Error data - consists of a CSECT ID (2 bytes), reason code (1 byte), and error code (1 byte).
CCATCB	Pointer to the caller's TCB.
CCACPL	Pointer to the CTGPL.
CCAACB	Pointer to the ACB of the catalog that is currently being processed.
CCAURAB	Pointer to the record area block (RAB) of the record area currently in use.
CCASRCH	Search argument for I/O requests.
CCARxREC	Pointer to record area x. (There are six record areas, record area 0 through record area 5; x indicates the number of the record area in question.)
CCARPL1	Pointer to the RPL that is currently assigned to this request.
CCAEQDQ	An ENQ/DEQ parameter list that is used when VSAM catalog management issues the RESERVE macro.
CCAMSSPL	A GETMAIN/FREEMAIN parameter list that the VSAM catalog processor uses for most GETMAIN/FREEMAINs.
CCACMS	Pointer to the catalog management services work area (CMSWA); it is used only for DELETE, ALTER, DEFINE, and LISTCATALOG requests.
CCAREGS	An array of small (12-byte) register save areas. When a VSAM catalog processor routine calls a lower level (nested) routine, the contents of registers 12-14 are saved in the next save area by the routine that is called. Registers 12 and 14 contain the calling routine's base address and return address, respectively. Register 13 is used to maintain position within the array. Each time register 13 is saved, it points to the preceding save area. During a lower level routine's processing, register 13 points to the current save area (that is, the area containing the caller's registers). When a lower level routine exits, registers 12-14 are restored which causes register 13 to be automatically switched (the preceding save area becomes the current save area). Whenever VSAM catalog processor routines branch-enter external routines, they pass a standard 72-byte save area to the external routine. This is accomplished by increasing register 13 by 12 during the process of setting up the linking conventions for the branch and link. (The 72 bytes that follow the current save area are used as the standard save area. **Note:** The register contents stored within this array can be used in debugging to identify predecessor routines and modules.)
CCARAACB	Pointer to the ACB of the CRA that is currently being processed, or zero.
CCARARPL	Pointer to the RPL that is currently assigned to this request for CRA I/O use.

Major flags in the CCA are:

Flag	Description
CCAFLG1-4	Miscellaneous processing control flags.
CCARPLX	I/O option flags: 00.....0 PUT direct 00.....1 PUT sequential 01...... ERASE 1......0 GET direct 1......1 GET key equal to or greater than ..0..... Use the record area pointed to indirectly by CCAURAB ..1..... Use record area 0 ...0.... Addressed or CI operation ...1.... Keyed operation ....0... Update operation ....1... Non-update operation .....0.. Check for errors .....1.. Bypass error checking ......0. 505-byte low-key range record ......1. 47-byte high-key range record
CCAFLG9	Miscellaneous CRA processing flags
CCARVFG1	Miscellaneous recovery (ESTAE) control flags

Module Structure

Catalog management is packaged into three load modules. These modules are the following:

1. IGC0002F - Catalog Controller
2. IGG0CLA1 - VSAM Catalog Processor
3. IGG0CLCA - CVOL Processor

This set of modules resides within SYS1.LPALIB and can be viewed as a type 4 SVC routine consisting of three load modules. Catalog management receives control via SVC 26 and operates under an SVRB. Control is passed between the three load modules via XCTL. Each load module establishes its own ESTAE routine. A brief description of each load module follows.

1. ***IGC0002F - Catalog Controller***

   The function of this module is to translate (map) interfaces. The module logically processes from a front end and a back end.

   The front end receives control from the SVC SLIH whenever SVC 26 is issued. Register 1 points either to an OS CAMLIST or a VSAM CTGPL. If register 1 points to an OS CAMLIST, the OS request is translated into an appropriate VSAM request (a CTGPL is constructed). Control is then passed to IGG0CLA1.

   The back end receives control (at EP IGG0102F) from IGG0CLA1 upon completion of a VSAM request for a VSAM catalog. It determines if the original request was an OS CAMLIST request and if so, it translates the CTGPL output and the IGG0CLA1 return code into appropriate CAMLIST format. It then returns control to the issuer of SVC 26. For a more detailed description of this module, see *OS/VS2 Catalog Management Logic.*

2. ***IGG0CLA1 - VSAM Catalog Processor***

   IGG0CLA1 is a large load module that consists of many CSECTs and procedures. Control is passed between the various procedures via CALLs. This module relates a request to a specific catalog and also determines the catalog type. If the catalog is an OS CVOL, IGG0CLA1 passes control to the CVOL processor (IGG0CLCA). Otherwise, IGG0CLA1 accesses the VSAM catalog and performs the function indicated by the CTGPL. When the function is completed, IGG0CLA1 exits by passing control to the back end of IGC0002F. For a detailed description of VSAM catalog management, see *OS/VS2 Catalog Management Logic.*

3. ***IGG0CLA - CVOL Processor***

   IGG0CLCA is a load module that consists of several CSECTs and procedures. Control is passed between the various procedures via CALLs. This module translates CTGPL requests into OS catalog requests and accesses OS CVOLs to perform the indicated function. Upon completion of processing this module returns control to the issuer of SVC 26. For a detailed description of this module, see *OS/VS2 CVOL Processor Logic.*

VSAM Catalog Recovery Logic

This section describes how mainline VSAM catalog management supports recovery and also how its recovery routine works.

Mainline VSAM catalog management does the following:

- Establishes/releases the recovery environment
- Maintains a pushdown list end mark
- Tracks GETMAIN/FREEMAIN activity
- Maintains a CMS (catalog management services) function gate

Establishing/Releasing a Recovery Environment

To establish or release a recovery environment, the following actions occur:

1. Subfunction BLDCCA in module IGG0CLC9 issues a branch entry to ESTAE to establish the recovery environment. This is done immediately after storage has been obtained for the CCA via GETMAIN.

2. When BLDCCA completes the initialization of the CCA, it sets RVCCAV to indicate that the CCA is now valid.

3. Subfunction IGGPRCLU (request cleanup) in module IGG0CLC9 performs the following:

   - Indicates that the CCA is no longer valid (RVCCAV = off)
   - Frees any GETMAIN/FREEMAIN tracking spill blocks that may exist
   - Branch enters ESTAE to remove the recovery environment

Maintaining a Pushdown List End Mark

A pushdown list end mark is maintained so that the ESTAE recovery routine can reliably locate the last pushdown list entry. This enables the recovery routine to determine:

1. The address at which the last call to a nested subfunction was issued.
2. The routine to which this call was directed.

There is an instruction in the exit procedure code contained within each CSECT to insure that the first byte following the last active entry contains an end-of-list marker. (Note that X'00' and X'FF' are considered end-of-list markers.)

Tracking GETMAIN/FREEMAIN Activity

GETMAIN/FREEMAIN tracking provides the recovery routine with the information it needs to automatically issue FREEMAINs against those areas of main storage that have been acquired and not yet freed by VSAM catalog management. The GETMAIN/FREEMAIN tracking function is implemented as follows:

1. A 256-byte contiguous area is defined in the CCA. The area consists of:

   a. A 248-byte tracking buffer.

   b. A single entry GETMAIN/FREEMAIN length list (four bytes) with the high-order byte initialized to X'80' and the low-order three bytes defined as CCAMNLEN.

   c. The GETMAIN/FREEMAIN address word (CCAMNADR).

2. The ?GETMS and ?FREEMS macros generate code that:

   a. Track the operation. This is accomplished by an MVC instruction that traces the GETMAIN/FREEMAIN length and address by pushing it (shifting it left) to the bottom (low address) of the 248-byte tracking area.

   b. Check for full tracking buffer. If the buffer is full, a spill routine (IGGPARFS) is called before the tracking MVS instruction is issued. This spill routine:

      1) Issues GETMAIN to obtain a 256-byte spill buffer.

      2) Chains this buffer to the end of the spill buffer chain. (**Note:**Chain anchor words are located in the CCA.)

      3) Copies the CCA tracking buffer into the new spill buffer.

      4) Clears the CCA tracking buffer.

   c. If the ?GETMS macro call is specified with CLASS(S) for storage (global), a flag (MNATSCLS) is set in the first byte of the two-word trace entry to indicate this. Refer to the description of CCAMNCAT, a work area that is located at CCA+X'308', contained in *OS/VS2 Catalog Management Logic.*

CMS Function Gate

The CMS function gate assists the recovery routine in determining if DEFINE or DELETE backout action is required. This gate is represented by a bit (RVCMSFG) in field CCARVFG1. The bit is turned on by the CMS driver (IGGPCDVR in module IGG0CLAT) immediately after a successful return from the check authorization function. The bit is reset upon entry to the CMS cleanup function (IGGPCCLN in module IGG0CLAT).

Recovery Routine Functions

VSAM's catalog processor recovery routine is labelled IGGPCMRR (CSECT IGG0CLA9). This recovery routine is entered from MVS's recovery termination manager (RTM) whenever an error or interruption occurs either in VSAM catalog management or in any successor routine that VSAM catalog management can cause to receive control. A pointer to the STAE diagnostic work area (SDWA) is passed as input to IGGPCMRR. IGGPCMRR performs the following functions. (Functions 2-13 are performed only when the CCA is marked valid, that is, RVCCAV=ON.)

1. Retrieves the CCA pointer from the SDWA and puts it into register 11.
2. Saves the RTM return address in CCAR14S.
3. Saves the SDWA pointer in CCASDWAP.
4. Produces diagnostic output.
5. Initializes register 13 to point to the first register save area.
6. Cleans up RPLs (if required).
7. Determines if backout is to be performed.
8. Checkpoints the CCR (if required).
9. Drops catalog orientation.
10. Frees storage (using GETMAIN/FREEMAIN tracking information).
11. Frees GETMAIN/FREEMAIN tracking spill blocks (if any exist).
12. Performs DEFINE/DELETE backout (if applicable).
13. Restores the RTM return address and the SDWA pointer.
14. Frees the CCA.
15. Returns to RTM indicating that RTM should continue with termination.

The following sections describe the more complex of these recovery routine functions in greater detail.

Diagnostic Output (Function 4)

Diagnostic output is produced except in those situations where the recovery routine is invoked only for clean up type functions, such as CANCEL. Diagnostic output can be produced in two forms:

1. Information is placed in a variable recording area (SDWAVRA) within the SDWA. This data is written to the SYS1.LOGREC data set as part of an entry describing the error.

This variable data is formatted as follows:

Byte	Length	Description of Data
0(0)	8	VSAM catalog processor module name - 'IGC0CLA1'
8(8)	3	IGG0CLA1's entry point address
11(B)	8	Procedure name of the last-called routine
19(13)	3	Address of the last-called routine
22(16)	8	Procedure name of the routine that called the last-called routine
30(1E)	3	Address of the CALL to the last-called routine
33(21)	4	The characters 'CPL='
37(25)	28	A copy of the user's CTGPL

2. An SDUMP is taken (if allowed by the system).

Backout (Function 7)

Backout is performed for DEFINE or DELETE requests (except for DEFINE or DELETE catalog requests) when the CMS function gate is active (RVCMSFG = ON). When backout is to be performed, a switch (RVESBOR) is set. The backout function (Function 12) is described later in this chapter.

Drop Catalog Orientation (Function 9)

This function uses the normal IGGPRPLF subfunction to perform the RPL freeup/DEQ functions.

Storage Freeup (Function 10)

This function frees all the storage (with the exception of the CCA and any existing tracking spill blocks) that has been acquired and is still owned by the current VSAM catalog management request. Storage freeup is done as follows:

1. The GETMAIN/FREEMAIN tracking data is scanned starting at the first spill block (if any) and following the chain of spill blocks. When the last spill block has been processed, the scan continues with the first valid entry in the CCA tracking buffer. This first scan selects and eliminates paired entries; a paired entry consists of two entries with matching storage addresses, which indicate that the storage area in question has already been freed.

2. The tracking data is scanned again. During this second scan, each valid remaining entry is processed as follows:

   a. The length and address of the storage to be freed are extracted from the entry.

   b. The subpool is determined from a switch setting within the entry.

   c. A ?FREEMS macro is issued to free the main storage. This macro specifies "RFR (NO)" to prevent recursive tracking.

DEFINE/DELETE Backout (Function 12)

This function attempts to preserve catalog integrity by cleanup up partially-completed DEFINE or DELETE operations. It uses the normal DELETE function to accomplish this. The switch indicating that backout is required is tested. If this switch is on, the following actions are performed:

1. A backout work area is obtained.

2. A DELETE CTGPL is constructed in the backout work area. This CTGPL is set up to cause a DELETE of the object that was being defined (with DEFINE) or deleted (with DELETE) whenever the error occurred.

3. The CCA is rebuilt as follows:

   a. CCACPL, CCASZ, CCATCB, CCASDWAP, CCAR14S, and CCARVFG1 are saved (in the backout work area).

   b. The complete CCA is cleared.

   c. The previously-saved fields (with the exception of CCACPL) are restored.

   d. CCAPCL is initialized to point to the CTGPL, which was built into the backout work area.

   e. CCAID, CCAURAB, CCAR0REC through CCAR5REC, CCAEDXFF, CCAMNPTR, CCAMNLLP, CCAMNLL, and register 13 are reinitialized to their original values.

   f. CCAF2SYS is set on.

   g. RVESBO is set on to indicate that backout is in control.

4. The CMS driver (IGGPCDVR is invoked which then invokes the DELETE function; when the DELETE action is complete, control is returned to the recovery routine.

5. The CCR is checkpointed (if required).

6. Catalog orientation is dropped (via a call to IGGPRPLF).

7. CCACPL is restored.

8. The backout work area is freed.

9. Any spill blocks acquired during the backout process are freed.

Debugging Aids

The control block structures for the VSAM catalog reside in the CSA. There is a built-in communications vector table (CVT) debug word which allows you to get a console dump at the time of the failure. This word is located at CVT+X'108' and is examined by module IGG0CLC9 at the end of each catalog request. Following are the contents of the CVT debug word:

Byte 0 (X'108') bits 0-3 must remain unchanged.

bit 4 not used by catalog.

bit 5=1 causes message IEC331I to be issued when condition specified in byte 1 (X'109') is met. IEC331I contains the name of the catalog module which detected the error.

bit 6 not used by catalog

bit 7=1 prevents catalog FRR (IGG0CLA9) from freeing the catalog communications area (CCA) so that it is available in the dump.

Byte 1 (X'109') Condition for which action specified at location X'10A-10B' is to be taken.

X'01' - take action at end of every catalog request

X'02' - take action for any non-zero catalog return code

X'03' - take action for return codes other than those considered to be "normal." (The following are considered to be normal return codes - X'00, 08, 24,28, 2C, 4C, 8C' and reason codes X'28, BC, and F0'.)

X'04' to X'FF' - take action only when catalog return code equals value in this byte.

Bytes 2 and 3 (X'10A-10B')
Action to be taken on above condition:

X'07FE' - return immediately to inline catalog code and continue processing. This setting, in conjunction with bit 5 of byte 0, causes no action other than message IEC331I.

X'07FF' - will cause loop here at CVT+X'10A' to allow console dump of failing ASID. To break job out of loop, either cancel the job or set these bytes to X'07FE' to continue processing.

When message IEC331I appears by itself, use the above CVT trap to get a dump of the failure. When messages IEC331I, IEC332I, and IEC333I appear together, the error is the result of a call to record management. Message IEC333I contains the record management return code in the form Lxxx (for logical error) or Pxxx (for physical error) where xxx = decimal return code. In these cases use the CVT trap discussed earlier in the Record Management Debugging Aids section of VSAM component analysis.

In situations where an attempt to open a VSAM catalog results in message IEC161I 004-080, it is difficult to determine the exact nature of the problem because there are many conditions which can cause this error. The best place to trap dump is at label 'CAPERR' in modules IFG0191X and IFG0191Y. Register 14 at that point will be in the calling routine which detected the failure.

It is sometimes necessary to examine the records in the catalog as part of the problem analysis. The following is an example of the access method services job necessary for this.

```
//PRINT       EXEC  PGM=IDCAMS
//STEPCAT     DD  DSN=catalogname,DISP=SHR
//DD1         DD  DSN=catalogname,DISP=SHR
//SYSPRINT    DD  SYSOUT=A
//SYSIN       DD *
  PRINT INFILE(DD1)
/*
```

The following ENQs are issued for catalog processing:

Major Name	Minor Name	Modules	Reason
SYSIGGV1	MCATOPEN	IGG0CLAC IGG0CLAD	Open master catalog
SYSIGGV2	catalogname	IGG0CLA3	Assign RPL processing
SYSVTOC	volser	IGG0CLBU	Read/Write format 4 DSCB
SYSZCAXW	CAXW	IDACAT11 IDACAT12 IGG0CLBG	Open, close, or delete Catalog request
SYSZPCCB	PCCB	IGG0CLA3	While building PCCB for catalog open
SYSZTIOT	asid	IDACAT11 IDACAT12	Open and close of catalog
		IGG0CLAD	Component recovery area (CRA) orientation
IEZIGGV3	addr of caxwa	IGG0CLA3	While Caxwa RPL count is being altered.

Allocation/Unallocation

This section is divided into four parts. Part one provides a description of the six major functional areas of allocation/unallocation and the way in which they interrelate. Parts two, three, and four contain general debugging aids, debugging hints, and reason codes.

Functional Description

Figure 5-33 illustrates the control-flow discussion that is presented in the following paragraphs.

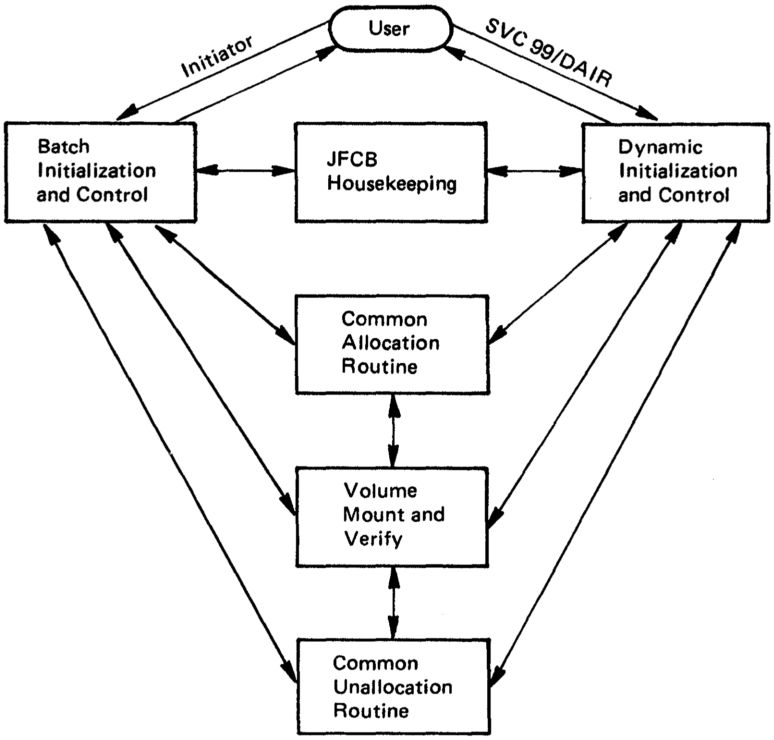

User
Initiator
SVC 99/DAIR
Batch Initialization and Control
JFCB Housekeeping
Dynamic Initialization and Control
Common Allocation Routine
Volume Mount and Verify
Common Unallocation Routine

Figure 5-33. Relationship of the Six Major Functions of Allocation/Unallocation

Allocation

The flow through allocation following either batch initialization or dynamic initialization is the same:

- Batch/dynamic initialization and control invokes JFCB housekeeping.

- Batch/dynamic initialization and control then invokes common allocation.

- Common allocation invokes volume mount and verify (if volume unloading or mounting is needed).

Unallocation

At batch/dynamic unallocation, the control flow is as follows:

- Batch/Dynamic initialization and control invokes common unallocation.

- Common unallocation invokes volume mount and verify (if any volume unloading is needed).

- Batch initialization and control invokes volume mount and verify (if volume unloading is needed).

Batch Initialization and Control

Batch initialization and control uses the following control blocks:

- Job control table (JCT)
- Step control table (SCT)
- Linkage control table (LCT)
- Job step control block (JSCB)

The SCT is needed to locate the chain of step I/O tables (SIOTs) and job file control blocks (JFCBs) in the scheduler work area (SWA). A SIOT and its corresponding JFCB are constructed by the converter/interpreter for each DD statement in a job step's JCL. Allocation allocates one step at a time. The SIOTs and JFCBs for a step are read by batch initialization and control when initializing for the allocation or unallocation of a step. At step initiation, space for the task I/O table (TIOT) is obtained, and the JSCB is initialized to point at the top of the chain of data set association blocks (DSABs), which are actually constructed by common allocation. At job step allocation, the SIOTs and JFCBs are passed as the main input, first to JFCB housekeeping, and then to common allocation. At job step unallocation, the SIOTs and JFCBs are passed as the main input to common unallocation. At the end of the job, batch initialization and control uses a volume unload table (VUT) to determine those private volumes that belong to the ending job and that are to be unloaded. Unloading is done by volume mount and verify (VM&V).

Dynamic Initialization and Control

When dynamic initialization and control is invoked, the job step's SIOTs and JFCBs must be read. This is done only for the first dynamic allocation during a given job step. The caller's parameters are syntax- and validity-checked and used to build a SIOT and JFCB, just as in a DD statement. Existing allocations (represented by an existing DSAB and TIOT entry) are used where possible to satisfy the request. If the requested data set is already allocated, certain information is copied from the SIOT and JFCB of the existing allocation to those of the new allocation. By using the existing allocation, invocation of JFCB housekeeping and common allocation is avoided. If an existing allocation cannot be used to satisfy the dynamic request, the SIOT and JFCB built by dynamic initialization and control are used, first as input to JFCB housekeeping, then to common allocation. After common allocation completes, the SIOT(s) representing the request is chained to the step's other SIOTs.

If dynamic unallocation is being requested, the parameters must be syntax- and validity-checked. The correct SIOT is located and passed to common unallocation.

JFCB Housekeeping

The major input to JFCB housekeeping is the SIOT chain, each SIOT having an associated JFCB. JFCB housekeeping completes needed information about either batch or dynamic allocation requests that was not placed in SIOTs and JFCBs by the converter/interpreter. Allocation parameters that JFCB housekeeping completes are the name, volume, unit, DCB, and disposition of the data set. Before processing these parameters, JFCB housekeeping, using dynamic allocation, allocates to the initiator's task control block (TCB) any STEPCAT DD or JOBCAT DD statements. A private catalog control block (PCCB) is built for each such catalog allocated, and all SIOTs are processed, one at a time. This JOBCAT/STEPCAT processing takes place in a batch environment only. Information for a request is placed in the JFCB housekeeping work area as a SIOT/JFCB pair, is processed and reinitialized for each SIOT. If volume information was not specified for an old data set, the passed data set information (PDI) is searched (only in a batch environment) in the SWA to locate volume and unit information. If not found, or if the data set name is a generation data group (GDG) single name, a catalog LOCATE is issued to obtain the volume and unit information. If volume reference is specified in the SIOT, either the data set referenced is located in the PDI or via catalog LOCATE, or the SIOT/JFCB of the referenced DD statement is found. The source of volume and unit information is recorded in the JFCB housekeeping work area; the information is then retrieved and placed into the SIOT/JFCB being processed. A DCB reference to a cataloged data set is resolved by LOCATE and OBTAIN. A DCB reference to a DD statement is resolved by going to the JFCB of the referenced DD statement and then issuing an OBTAIN. Finally, disposition-related information is entered into the SIOT/JFCB.

Common Allocation

Common allocation receives as input the SIOTs and JFCBs of allocation requests. For requests that do not require a unit to be allocated, namely, DUMMY, VIO, and subsystems, DSAB and TIOT entries are built and the SIOT is marked "allocated." For each request requiring units, a list of eligible devices called the eligible device list (EDL) is constructed, and pointed to by the requestor's SIOT. An entry is built into the volunit table representing each volume/unit required. Inter-DD relationships are represented primarily by setting fields in the VU table for use by the remainder of common allocation.

The remainder of common allocation is divided into:

- Fixed Device Allocation
- TP Allocation
- Generic Allocation
- Recovery Allocation

Common allocation control invokes each of these functions in the order indicated.

If all requests have been allocated, any requests needing volumes mounted have volume mount and verify (VM&V) RBs chained to their SIOTs. These VM&V

RBs are chained to each other and sent to VM&V on input. VM&V mounts the necessary volumes.

Fixed Device Allocation

Allocation for any request that can be allocated to a volume on a permanently-resident or reserved DASD uses fixed device allocation. The allocation of a request (VU entry) involves:

- The selection of the device
- The building of the DSAB (pointed to by a SIOT)
- The building of a TIOT entry (pointed to by a DSAB)
- Setting indicators in the unit control block (UCB) of the selected device
- Issuing DADSM commands

TP Allocation

This is a small specialized operation for teleprocessing lines. TP lines, once allocated, remain allocated whether online or not, and cannot be reallocated.

Generic Allocation

Generic allocation attempts to allocate the remaining requests that were not allocated by previous processes. Requests for tapes, demountable direct access volumes, graphics devices, and unit record devices are not considered until generic allocation. A special set of tables, the generic allocation tables are built to represent the units eligible for each request (VU entry). These tables are used throughout generic and recovery allocation. Generic allocation processes requests not sequentially but on the basis of generic device type. The order in which generic device types are chosen is determined by a table, built at SYSGEN time, called the device preference table.

Recovery Allocation

Requests left unallocated by previous steps are allocated by recovery allocation. The main functions of recovery allocation are to interface with the operator to request that offline devices be brought online, and, once online, to allocate these devices to unallocated VU entries.

Common Unallocation

The input to common unallocation is a chain of RBs, each of which points to a SIOT to be unallocated. Disposition processing uses the SIOT/JFCB and common unallocation RB to give the data set a disposition. Units allocated to each SIOT are unallocated by using the TIOT entry. Private tape volumes are unloaded and the VUT is updated with volume serials to indicate which of the job's volumes were left mounted at unallocation time, but need demounting by batch initialization and control at end of job. Data sets are released (dequeued) by using the data set enqueue table to determine if the data set's last use in the job is in the current step. All volumes used by a step are released by a generic dequeue if unallocation is for a step. In the dynamic unallocation environment, only the subject request's volumes are dequeued.

Volume Mount and Verify

Volume mount and verify (VM&V) mounts, verifies, and unloads volumes. VM&V is driven by a chain of VM&V request blocks. A VM&V count table is built in which the numbers of mount, verify, and unload requests are maintained. In mounting and verifying direct access volumes, VM&V builds a mount verification communication area (MVCA) in CSA. This contains a pointer to an MVCA extension (MVCAX), which VM&V builds in the user region. The MVCAX contains a device-end ECB and UCB pointers for each device for which a mount has been issued. After issuing mounts and building the MVCA/MVCAX blocks, VM&V waits for the device-end ECB in the MVCAX. Whenever a device-end occurs on a unit that VM&V is waiting for, a nucleus routine (IEFDPOST) posts the device-end ECBs in all MVCAXs. Any VM&V that is waiting looks at all UCBs being waited for. Volume serials for DASD are read and verified when the devices become ready.

Volume unloading is accomplished for DASD by signaling a volume unload event to the event notification facility (ENF), issuing an unload message to the operator, and clearing volume-related data from the UCB. For tape volume unloading, a physical rewind/unload operation is also performed. Virtual volume unloading is accomplished by signaling a volume unload event to the event notification facility (ENF), issuing an unload SVC (SVC 126), and clearing volume-related data from the UCB.

General Debugging Aids

Described here in general terms are the following:

- Allocation Module Naming Conventions
- Registers and Save Areas
- Common Allocation Control Block Processing
- ESTAE Processing

Allocation Module Naming Conventions

All Allocation module names have the following format:

IEF B4

IEF indicates the module is a scheduler module. The fourth character has the following meaning:

- If A, the module is part of common allocation, common unallocation, JFCB housekeeping, or volume mount and verify.
- If B, the module is part of batch allocation or batch unallocation.
- If D, the module is part of dynamic allocation or dynamic unallocation.
- If E, the module is part of an externally available allocation service.
- If H, the module uses cross memory operations.

B4 identifies the module as a part of allocation. The last two characters are a unique module identifier.

Registers and Save Areas

Allocation follows standard register saving and usage conventions. Register 13 is used as a save area pointer, register 14 as a return address, and register 15 as a branch address. Register save areas are chained in the standard manner.

Since allocation is coded completely in top-down fashion, it is a simple matter to find the flow of control leading to the current point of processing by tracing back through the save areas. All allocation modules have identifiers just after the beginning of the module, which contain the module name in EBCDIC. A graphic representation of control flow can be found under "Allocation/Unallocation" in "Module-to-Module Control Flow" of *OS/VS2 System Logic Library.*

Space for the allocation save areas is obtained in a unique manner, which can be of help in debugging. On entry to allocation, a 4K block of space is obtained from subpool 230. This block is used to contain the save area and data area for each module called, until the block is full, at which time another 4K block is obtained. Save areas of modules that had been given control but then returned are still valid, that is not freed, if the 4K block in which they had been placed has not been freed. Allocation does not keep the address of a control block in any particular register. Register 13 always points at the save area of the module in control. Register 12 is usually the base register of the module in control.

Common Allocation Control Block Processing

This section graphically describes the control blocks used by common allocation and explains how these control blocks reflect allocation processing. Figure 5-34 shows the control blocks which are input to common allocation. Data set association blocks (DSABs) and their associated task input/output table (TIOT) entries are shown as input. Note that DSABs exist only if common allocation was called by dynamic allocation. When batch allocation calls common allocation, there are no DSABs, but there is a DSAB queue descriptor block (QDB).

The first major step in common allocation processing is the construction of the allocation work area (ALCWA). Following this, requests that do not require units, such as DUMMY and SYSOUT DD requests, are allocated. A DSAB and TIOT entry are built for each of these requests as they are allocated. SIOTETIO is initialized to point to the DSAB whenever it is created for a given SIOT. Bit SIOTALCD is set to 1 whenever a request (SIOT) is fully allocated.

After allocating these requests, the volunit table (VU table) is created to represent the unit requirements of remaining (unallocated) SIOTs. In addition, an eligible devices list (EDL) is created for each remaining SIOT. The EDL contains the unit control block (UCB) pointers to all UCBs representing devices eligible for allocation to the SIOT. (A device is "eligible" at this point whether on or offline, either logically or physically.) Figure 5-35 shows the relationship of the ALCWA, SIOTs, etc., after the VU table and EDLs are built. The first SIOT on the chain (SIOT A) represents a SYSOUT DD statement that has already been allocated. The second SIOT on the chain (SIOT B) represents a SIOT that requires one or more units. It is shown to have 2 volunit entries, which indicates the total number of units that can be allocated to that SIOT. SVOLUNNO in the SIOT contains the number of VU entries for a SIOT. (Note that the total number of units allocated to a request can exceed the number of units requested.

This happens, for example, if a specifically requested volume was found to be mounted with the permanently-resident mount attribute.)

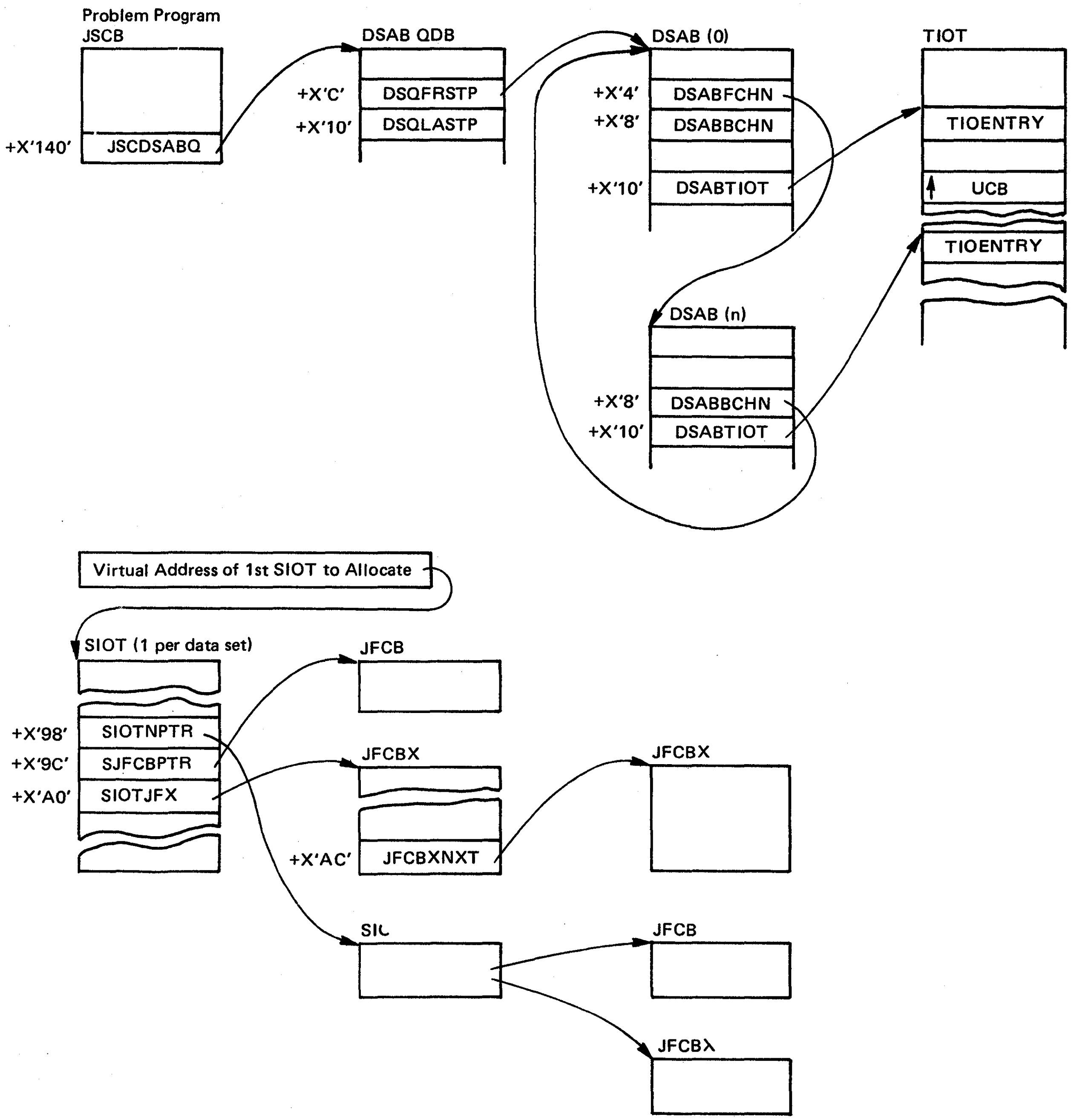


Figure 5-34. Common Allocation Input

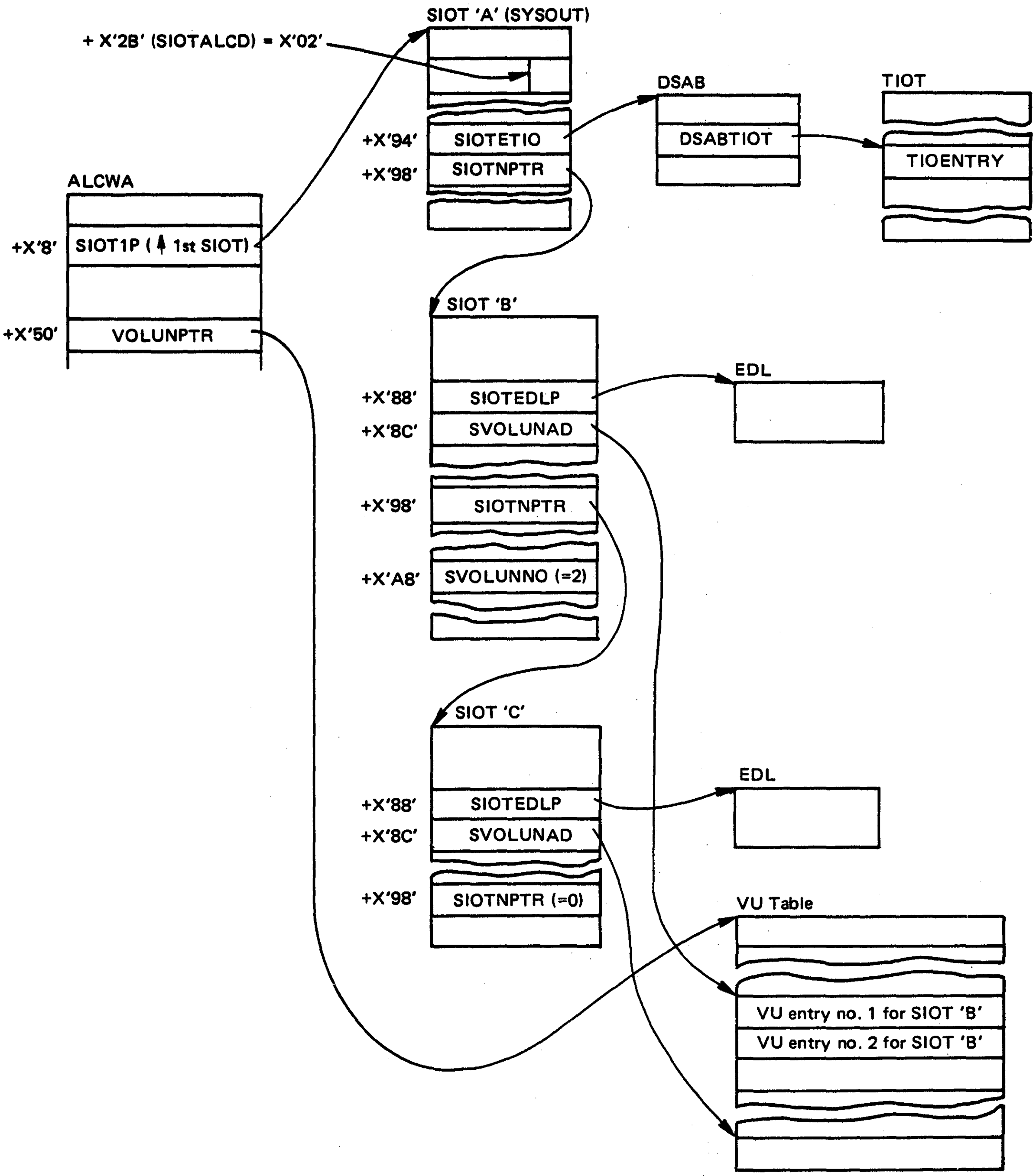


Figure 5-35. Common Allocation Control Blocks After Construction of Volunit Table and EDLs

Common allocation processing is reflected by the status of request's SIOT and VU entries. As each VU entry requiring a unit is allocated, bit VOLALOC (bit 0 (X'80') at +7 into the VU entry) is set on. Bit VDEVREQD (bit 2 (X'20') at +7 into the VU entry), if on, indicates that the VU entry requires a unit. Once all VU entries with VDEVREQD=1 for a given SIOT are allocated and VOLALOC=1, the SIOT is marked allocated by setting on SIOTALCD (bit 6 (X'02') at X'2B' into the SIOT).

As each unit is allocated to a request, that allocation is reflected in (1) the unit's UCB by setting UCBALOC (bit 4 (X'08') at +3 in the UCB) on, and in (2) the request's TIOT entry by placing the UCB pointer into field TIOUCBP in the TIOT entry. (TIOUCBP is at a X'10' into a TIOT entry for the first unit allocated, at +X'14' for the second, etc.) The first time a VU entry for a SIOT is allocated, a DSAB and TIOT entry are created. For subsequent VU entries allocated to a SIOT, the DSAB and TIOT entries are updated.

ESTAE Processing

All of allocation is protected from abends by ESTAE processing. Only one ESTAE is issued during allocation. The batch allocation ESTAE exit routine, IEFAB4E4, performs a retry, causing routine IEFAB4E3 to get control. IEFAB4E3 returns to the initiator with a failure return code, causing the initiator to fail the job. All other ESTAE exit routines percolate to the next higher level of ESTAE protection. In a batch unallocation environment, this causes the initiator to terminate.

When an abend occurs in a batch environment, message IEF197I "SYSTEM ERROR DURING ALLOCATION/UNALLOCATION" is issued to SYSOUT by ESTAE processing. If the abend occurs in batch allocation or a routine called by batch allocation, such as JFCB housekeeping, message IEF197I is issued to the job's SYSOUT. If the abend occurs during batch unallocation, the same message goes to the initiator's SYSOUT.

An SVC dump is always taken if an abend occurs when allocation is in control.

Unit Allocation Status Recording

Unit allocation status recording uses the control blocks in the allocation address space (ALLOCAS) to record the use count of all units that are allocated to all address spaces. (The count is kept in the DALTUSE fields in the DALTs.) Each time that a unit is allocated to an address space, the use count is increased by one; and each time that the unit is unallocated from the address space, the use count is decreased by one. The value in the DALTUSE field represents the number of times that a unit is currently allocated to a given address space.

By keeping count of each time that a unit is allocated to each address space:

- When the DISPLAY U,,ALLOC operator command is issued, message IEE106I displays allocation information which includes the jobnames and ASIDs of each job to which the unit is allocated.

- If an address space is abnormally terminated, allocation can unallocate shared units from the terminating address space. (Without unit allocation status recording, allocation does not unallocate shared units from an abnormally terminating address space.)

The control blocks in the allocation address space are created and initialized by the allocation module IEFHB4I1 during system initialization. The control blocks are:

ADB - Allocation descriptor block - anchors the other control blocks in the allocation address space.

DAIT - Display allocation index table - contains an array of eight-byte entries starting at offset X'8'. An entry contains an index value into the DALTs for each unit defined during system generation and the address of the UCB for each unit. A unit address, which is the same as the UCBNAME field in the unit's UCB, is used as the index to the unit's entry in the DAIT.

Note: If the unit address in the UCBNAME name field does not equal the unit address that was used to index into the DAIT, then a DDR swap has occurred. In this case, the unit address in the located UCBNAME field is used to index into the DAIT to locate the proper entry for the unit.

DALT - Display allocation lookup table (one for each ASID) - contains an array of two-byte entries starting at offset X'8'. An entry contains the use count for each unit that is allocated to an address space. The index value found in the DAIT for a given unit address is used to index into the DALT to locate the use count for the unit. Index values are used (in the DAIT) so that the DALTs only need to contain as many two-byte use count entries as there are units defined during initialization (rather than a maximum number of entries, 4096).

DVT - Display allocation vector table - contains an array of four-byte entries starting at offset X'8'. An entry contains the address to the DALT for each possible ASID in the system. An ASID + 1 value is used as the index into the DVT to locate the address of the DALT for a given ASID. The DALT for ASID 000 is used by allocation to associate a use count of those units for which allocation could not determine the ASID that allocated the unit (such as those units that are allocated during IPL before unit allocation status recording is initialized).

Important fields in these control blocks are:

DAITUNIT - two-byte field containing the index value into the DALTs for each unit.
DAITUCB - four-byte field containing the pointer to the UCB for each unit.
DVTDALT - four-byte field containing the pointer to the DALT for each ASID.
DALTUSE - two-byte field containing the use count for each unit allocated to each ASID.

Figure 5-36 shows the control block structure of the control blocks in the allocation address space (ALLOCAS).

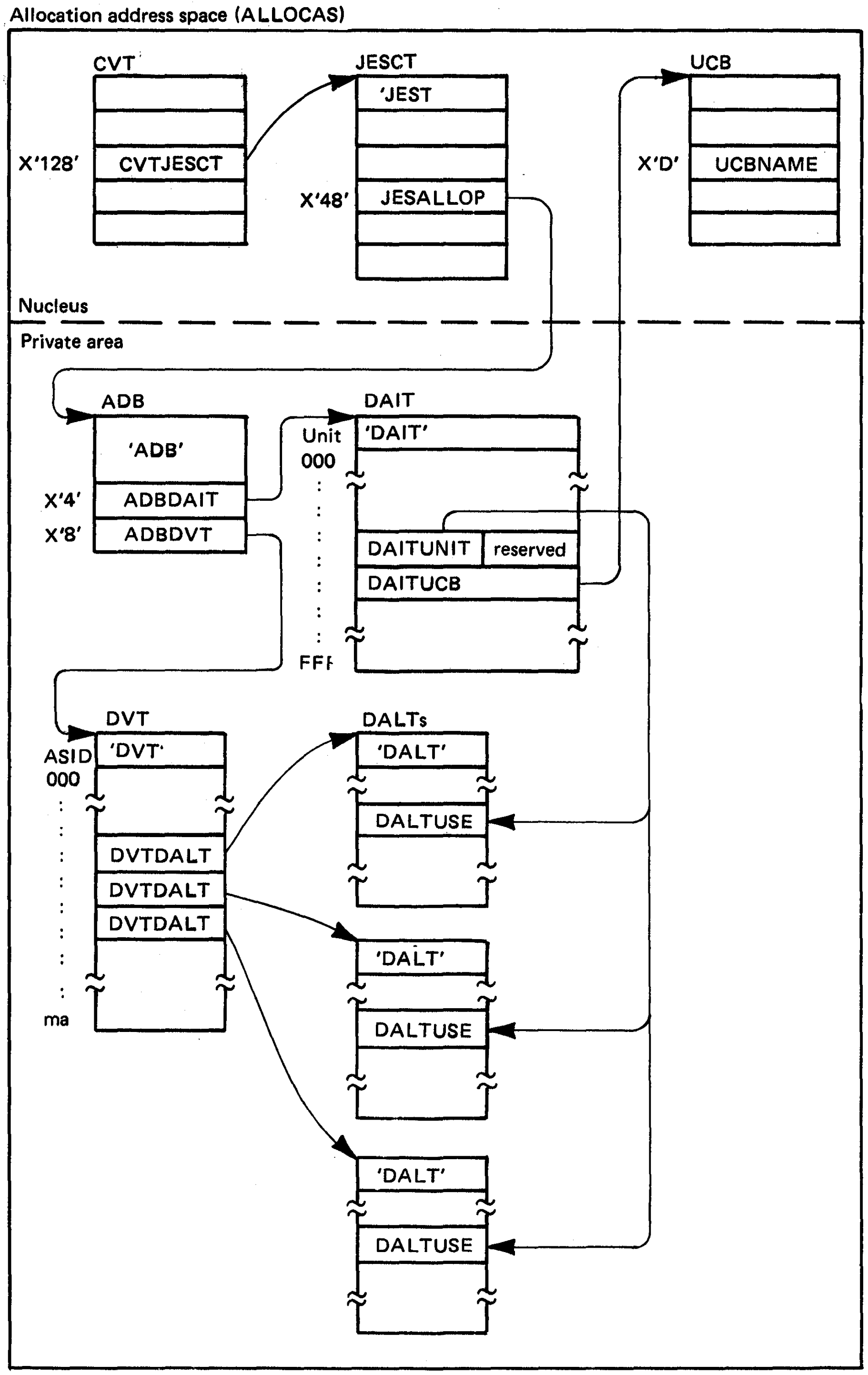


Figure 5-36. ALLOCAS Control Block Structure

Important fields in the JES control table (JESCT) related to allocation status recording are:

Offset	Length	Name	Description
X'48'	4	JESALLOP	Contains the address of the allocation descriptor block (ADB).
X'4C'	2	JESALLOA	Contains the ASID value of the allocation address space (ALLOCAS).
X'4E'	1	JESALLOF	ALLOCAS flags:
	10..	JESUASR	Indicates that unit allocation status recording was initialized successfully and is active.
	01..	JESUASF	Indicates that unit allocation status recording has failed and is not active.
X'54'	4	JESAUCBS	Contains the total number of UCBs in the system.

ALLOCAS Recovery Considerations

If the allocation address space (ALLOCAS) abnormally terminates, module IEFAB4E6 sets bit JESUASF on, sets bit JESUASR off, records the error to SYS1.LOGREC, issues message IEF100I, and takes an SVC dump. In this case, allocation continues to process requests but without unit allocation status recording.

ALLOCAS Debugging Hints

To help you debug problems with unit allocation status recording, this topic contains procedures to determine:

- The UCB address corresponding to a DALTUSE field
- The ASID corresponding to a DALTUSE field
- The DAIT entry for a unit address
- The DVT entry for an ASID
- The DALT entry (DALTUSE field) for an ASID and unit address.

In the procedures that follow, use Figure 5-36 to determine the location of the DAIT and DVT control blocks.

Figure 5-37 (5 parts) shows a sample dump of the ALLOCAS address space. The following procedures, formulas, and examples refer to the circled letters shown on the ALLOCAS dump. (Note that the examples indicate that a DDR swap has occurred between unit 351 and 354.)

1. *Determining the UCB address corresponding to a DALTUSE field.*

   a. Determine the address of the two-byte DALTUSE field (circle A) in the dump. The DALTUSE field contains the use count (0005) for the unit and begins on a halfword boundary.

   b. Determine the beginning address of the DALT (circle B) that the DALTUSE field is part of. To do this, locate the acronym DALT (X'C4C1D3E3') that precedes the DALTUSE field.

c. Find the index value of the DALTUSE field into the DALT by using the formula:

Index value = $\frac{A-(B+8)}{2} + 1$

Where:

A is the address of the DALTUSE field
B is the address of the DALT

Example: $\frac{A71C7E-(A71918+8)}{2} + 1 = 01B0$

The index value into the DALT is 01B0.

d. Scan the DAIT (circle C) to find the DAIT entry containing the index value (01B0) found on step c. Find the index value of the DAIT entry into the DAIT by using the formula:

Index value = $\frac{D-(C+8)}{8}$

Where:

D is the address of the DAIT entry
C is the address of the DAIT

Example: $\frac{A68190-(A666E8+8)}{8} = 354$

The index value into the DAIT is 354.

Note: The index value into the DAIT is usually equal to the UCBNAME field of the UCB representing the unit.

e. The fullword (circle E) after the DAIT index value is the pointer to the UCB (circle F) for the unit. In the example, the UCBNAME field (at X'D') indicates the unit address (circle G) is 351.

Note: In the example, because the UCBNAME field (351) is not equal to the index value into the DAIT (354), a DDR swap has occurred between units 351 and 354. In the example, the DALTUSE field (at circle A) represents the use count for unit address 351.

2. *Determining the ASID corresponding to a DALTUSE field.*

a. Determine the beginning address of the DALT (circle B) that the DALTUSE field (circle A) is part of.

b. Scan the DVT (circle H) to locate the entry that contains the address (circle I) of the DALT found on step a.

c. Find the index value of the DALT entry into the DVT by using the formula:

$$\text{Index value} = \frac{\text{I-(H+8)}}{4}$$

Where:

I is the address of the DALT entry
H is the address of the DVT

$$\text{Example: } \frac{\text{A6E714-(A6E6F0+8)}}{4} = 7$$

The index value into the DVT represents ASID 7.

3. *Determining the DAIT entry for a unit address.*

a. Determine the beginning address of the DAIT (circle C).

b. Find the address of the eight-byte DAIT entry (for an example unit address of 354).

Entry address = C+8+(Ux8)

Where:

C is the address of the DAIT
U is the unit address

Example: A666E8 + 8 + (354 x 8) = A68190

The address of the DAIT entry (for unit 354) is A68190.

c. The first halfword of the DAIT entry (circle D) is the index value (01B0) into the DALT for the given unit address 354.

d. The second fullword of the DAIT entry (circle E) is the pointer to the corresponding UCB (circle F) for the unit.

e. The UCBNAME field (at X'D') in the UCB contains the UCB's unit address.

Note: In the example, the unit address (circle G) in the UCBNAME field is 351. Because the UCBNAME field does not equal the unit address used on step b, a DDR swap has occurred between units 354 and 351. To find the proper DAIT entry for unit 354, repeat the procedure and use the DDR-swapped unit address of 351 in the formula on step b. (If the UCBNAME field is still not equal to 354, then more than one DDR swap has occurred; in this case, repeat the procedure again.)

4. *Determining the DVT entry for an ASID.*

   a. Find the address of the DVT entry in the DVT (circle H) for a given ASID (for example, ASID 7) by using the formula:

      Entry address = H+8+(Sx4)

      Where:

      H is the address of the DVT
      S is the ASID value

      Example: A6E6F0 + 8 + (7 x 4) = A6E714

   b. For ASID 7, the DVT entry is at A6E714 (circle I) which contains the DALT address A71918 (circle B).

5. *Determining the DALT entry for an ASID and unit address.*

   a. Find the DALT address for a given ASID (for example, ASID 7). See procedure 4.

   b. Find the DALT index value into the DAIT for a given unit address (for example, 351). See procedure 3. (Be aware of a DDR swap.)

   c. Find the address of the DALT entry (DALTUSE field) by using the formula:

      DALTUSE address = B+8+(Lx2)-2

      Where:

      B is the address of the DALT
      L is the DALT index value

      Example: A71918 + 8 + (01B0 x 2) - 2 = A71C7E

   d. Address A71C7E (circle A) is the DALTUSE field.

Note: In the examples, a DDR swap has occurred. Refer to the notes on procedures 1 and 3 for additional information about how to use these procedures when a DDR swap has occurred.

```
CR1                                   DUMP OF ALLOCATION ADDRESS SPACE (ALLOCAS)                                      PAGE 00013
V005440   00000000 00FF6808 0089FF88 03510000   70000700 00F3F5F1 30502009 0000E02C *.....................351.&......* R005440 06
V005460   00010100 F3F3F3F0 F0F10800 00000500   00000000 04FF6808 0189FF88 03550000 *....333001......................* R005460 06
V005480   70000700 00F3F5F5 30502009 0000E04C   00010100 E4F3F0D7 C1D20801 00000100 *.....355.&.....<....U30PAK......* R005480 06
V0054A0   00000000 00FF67B8 0089FF00 03564002   00000700 04F3F5F6 30502009 0000E06C *.............. ......356.&.....%* R0054A0 06
V0054C0   00000000 00000    00000000 00000000   00000    00FF67B8 0089FF00 03574002 *.............................. .* R0054C0 06
V0054E0   00000700 04F3F  F 30502009 0000E08C   00000  G 00000000 00000000 00000000 *.....357.&......................* R0054E0 06
V005500   00000000 00FF67B8 0089FF00 03604002   00000600 04F3F6F0 3050200D 0000E0AC *.............- ......360.&......* R005500 06
V005520   00000000 00000000 00000000 00000000   00000000 00FF67B8 0089FF00 03614002 *............................./ .* R005520 06
V005540   00000600 04F3F6F1 3050200D 0000E0CC   00000000 00000000 00000000 00000000 *.....361.&......................* R005540 06
V005560   00000000 00FF67B8 0089FF00 03624002   00000600 04F3F6F2 3050200D 0000E0EC *.............. ......362.&......* R005560 06
V005580   00000000 00000000 00000000 00000000   00000000 00FF67B8 0089FF00 03634002 *.............................. .* R005580 06
V0055A0   00000600 04F3F6F3 3050200D 0000E10C   00000000 00000000 00000000 00000000 *.....363.&......................* R0055A0 06
V0055C0   00000000 00FF67B8 0089FF00 03644002   00000600 04F3F6F4 3050200D 0000E12C *.............. ......364.&......* R0055C0 06
V0055E0   00000000 00000000 00000000 00000000   00000000 00FF67B8 0089FF00 03654002 *.............................. .* R0055E0 06
V005600   00000600 04F3F6F5 3050200D 0000E14C   00000000 00000000 00000000 00000000 *.....365.&.....<................* R005600 06
V005620   00000000 00FF67B8 0089FF00 03664002   00000600 04F3F6F6 3050200D 0000E16C *.............. ......366.&.....%* R005620 06
V005640   00000000 00000000 00000000 00000000   00000000 00FF67B8 0089FF00 03674002 *.............................. .* R005640 06
V005660   00000600 04F3F6F7 3050200D 0000E18C   00000000 00000000 00000000 00000000 *.....367.&......................* R005660 06
V005680   00000000 00FF67B8 0000FF00 03700002   00000500 04F3F7F0 12401009 0000E1AC *.....................370. ......* R005680 06
V0056A0   40000000 00000000 00000000 00000000   00000000 00FF67B8 0000FF00 03710002 * ...............................* R0056A0 06
V0056C0   00000500 04F3F7F1 12401009 0000E1CC   40000000 00000000 00000000 00000000 *.....371. ...... ...............* R0056C0 06
V0056E0   00000000 00FF67B8 0000FF00 03720002   00000500 04F3F7F2 12401009 0000E1EC *.....................372. ......* R0056E0 06
V005700   40000000 00000000 00000000 00000000   00000000 00FF67B8 0000FF00 03730002 * ...............................* R005700 06
V005720   00000500 04F3F7F3 12401009 0000E20C   40000000 00000000 00000000 00000000 *.....373. ....S. ...............* R005720 06
V005740   00000000 00FF67B8 0000FF00 03740002   00000500 04F3F7F4 12401009 0000E22C *.....................374. ....S.* R005740 06
V005760   40000000 00000000 00000000 00000000   00000000 00FF67B8 0000FF00 03750002 * ...............................* R005760 06
V005780   00000500 04F3F7F5 12401009 0000E24C   40000000 00000000 00000000 00000000 *.....375. ....S< ...............* R005780 06
V0057A0   00000000 00FF67B8 0000FF00 03760002   00000500 04F3F7F6 1200100B 0000E26C *.....................376......S%* R0057A0 06
V0057C0   00000000 00000000 00000000 00000000   00000000 00FF67B8 0000FF00 03770002 *................................* R0057C0 06
V0057E0   00000500 04F3F7F7 1200100B 0000E28C   00000000 00000000 00000000 00000000 *.....377......S.................* R0057E0 06
V005800   00000000 00FF67B8 0000FF00 03A00002   00000500 04F3C1F0 12501009 0000E2AC *.....................3A0.&....S.* R005800 04
V005820   40000000 00000000 00000000 00000000   00000000 00FF67B8 0000FF00 03A10002 * ...............................* R005820 04
V005840   00000500 04F3C1F1 12501009 0000E2CC   40000000 00000000 00000000 00000000 *.....3A1.&....S. ...............* R005840 04
V005860   00000000 00FF67B8 0000FF00 03A20002   00000500 04F3C1F2 12501009 0000E2EC *.....................3A2.&....S.* R005860 04
V005880   40000000 00000000 00000000 00000000   00000000 00FF67B8 0000FF00 03A30002 * ...............................* R005880 04
V0058A0   00000500 04F3C1F3 12501009 0000E30C   40000000 00000000 00000000 00000000 *.....3A3.&....T. ...............* R0058A0 04
V0058C0   00000000 00FF67B8 0000FF00 03A40002   00000500 04F3C1F4 11501009 0000E32C *.....................3A4.&....T.* R0058C0 04
V0058E0   40000000 00000000 00000000 00000000   00000000 00FF67B8 0000FF00 03A50002 * ...............................* R0058E0 04
V005900   00000500 04F3C1F5 12501009 0000E34C   40000000 00000000 00000000 00000000 *.....3A5.&....T< ...............* R005900 04
V005920   00000000 00FF67B8 0000FF00 03A60002   00000500 04F3C1F6 12501009 0000E36C *.....................3A6.&....T%* R005920 04
V005940   40000000 00000000 00000000 00000000   00000000 00FF67B8 0000FF00 03A70002 * ...............................* R005940 04
V005960   00000500 04F3C1F7 12501009 0000E38C   40000000 00000000 00000000 00000000 *.....3A7.&....T. ...............* R005960 04
V005980   00000000 00FF67B8 0000FF00 03A80002   00000500 04F3C1F8 12501009 0000E3AC *.....................3A8.&....T.* R005980 04
V0059A0   40000000 00000000 00000000 00000000   00000000 00FF67B8 0000FF00 03B10002 * ...............................* R0059A0 04
V0059C0   00000500 04F3C2F1 11001009 0000E3CC   00000000 00000000 00000000 00000000 *.....3B1......T.................* R0059C0 04
V0059E0   00000000 00FF67B8 0000FF00 03B20002   00000500 04F3C2F2 1200100A 0000E3EC *.....................3B2......T.* R0059E0 04
V005A00   00000000 00000000 00000000 00000000   00000000 00FF67B8 0000FF00 03B30002 *................................* R005A00 04
V005A20   00000500 04F3C2F3 12001009 0000E40C   00000000 00000000 00000000 00000000 *.....3B3......U.................* R005A20 04
V005A40   00000000 00FF67B8 0000FF00 03B40002   00000500 04F3C2F4 12001009 0000E42C *.....................3B4......U.* R005A40 04
V005A60   00000000 00000000 00000000 00000000   00000000 00FF67B8 0000FF00 03B50002 *................................* R005A60 04
V005A80   00000500 04F3C2F5 1200100B 0000E44C   00000000 00000000 00000000 00000000 *.....3B5......U<................* R005A80 04
V005AA0   00000000 00FF67B8 0000FF00 03B60002   00000500 04F3C2F6 1100100A 0000E46C *.....................3B6......U%* R005AA0 04
V005AC0   00000000 00000000 00000000 00000000   00000000 00FF67B8 0000FF00 03B80002 *................................* R005AC0 04
V005AE0   00000500 04F3C2F8 115C1003 0000E48C   00000000 00000000 00000000 00000000 *.....3B8.*....U.................* R005AE0 04
V005B00   00000000 00FF67B8 0000FF00 03B90002   00000500 04F3C2F9 115C1003 0000E4AC *.....................3B9.*....U.* R005B00 04
V005B20   00000000 00000000 00000000 00000000   00000000 00FF67B8 0000FF00 03C00002 *................................* R005B20 04
V005B40   00000500 04F3C3F0 10000822 0000E4CC   00000000 00FF67B8 0000FF00 03C10002 *.....3C0......U..............A..* R005B40 04
V005B60   00000500 04F3C3F1 10000822 0000E4EC   00000000 00FF67B8 0000FF00 03C20002 *.....3C1......U..............B..* R005B60 04
```

F — Start of the UCB

G — UCBNAME field

Figure 5-37 (Part 1 of 5). ALLOCAS Dump

```
0005                                          DUMP OF ALLOCATION ADDRESS SPACE (ALLOCAS)                                          PAGE 00006

PAGE TABLE FOR SEGMENT  A6

     0008      0008      0008      0008
     0008      0008      13D1      30C9
     1DF1      1321      2E29      3389
     2B49      32F9      13E1      32C1
     000000000000000000000000    000000000000000000000000    000000000000000000000000    000000000000000000000000
     000000000000000000000000    000000000000000000000000    0800408000301D000000000    08004080003017800000000
     0800408000301C400000000    0800408000301D100000000    08004080003018800000000    08004080003018900000000
     08004080003018A00000000    08004080003018B00000000    0800408000301C800000000    0800408000301A400000000

VA66000   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000 *................................* R13D000 08
                                                         ABOVE LINE IS REPEATED
VA666C0   00000000 00000000 00000000 00000000   00000000 00000000 C1C4C240 00A666E8 *........................ADB ...Y* R13D6C0 08
VA666E0   00A6E6F0 80A8CCF0 C4C1C9E3 00000000   00000000 00000000 00010000 00001008 *..WO...ODAIT....................* R13D6E0 08
VA66700   00020000 00001030 00030000 00001058   00040000 00001080 00050000 000010A8 *................................* R13D700 08
VA66720   00060000 000010D0 00070000 000010F8   00080000 00001120 00090000 00001148 *...............8................* R13D720 08
VA66740   000A0000 00001    000B0000 00001188   000C0000 000011A8 000D0000 000011C8 *...............................H* R13D740 08
VA66760   000E0000 00001    000F0000 00001208   00100000 00001230 00110000 00001250 *.......Y.......................&* R13D760 08
VA66780   00120000 00001270 00130000 00001290   00140000 000012B0 00150000 000012E0 *................................* R13D780 08
VA667A0   00160000 00001300 00170000 00001330   00180000 00001350 00190000 00001380 *.......................&........* R13D7A0 08
VA667C0   001A0000 000013B0 001B0000 000013E0   001C0000 00001410 001D0000 00001430 *................................* R13D7C0 08
VA667E0   00000000 00000000 001E0000 00001450   001F0000 00001470 00200000 00001490 *...............&................* R13D7E0 08
VA66800   00210000 000014B0 00220000 000014D0   00230000 000014F0 00240000 00001510 *.......................0........* R13D800 08
VA66820   00250000 00001530 00260000 00001550   00270000 00001570 00280000 00001590 *...............&................* R13D820 08
VA66840   00290000 000015B0 002A0000 000015D0   002B0000 000015F0 002C0000 00001610 *.......................0........* R13D840 08
VA66860   002D0000 00001630 002E0000 00001650   002F0000 00001670 00300000 00001690 *...............&................* R13D860 08
VA66880   00310000 000016B0 00320000 000016D0   00330000 000016F0 00340000 00001710 *.......................0........* R13D880 08
VA668A0   00350000 00001730 00360000 00001750   00370000 00001770 00380000 00001790 *...............&................* R13D8A0 08
VA668C0   00390000 000017B0 003A0000 000017D0   003B0000 000017F0 003C0000 00001810 *.......................0........* R13D8C0 08
VA668E0   003D0000 00001830 003E0000 00001850   003F0000 00001870 00400000 00001890 *...............&................* R13D8E0 08
VA66900   00410000 000018B0 00420000 000018D0   00430000 000018F0 00440000 00001910 *.......................0........* R13D900 08
VA66920   00450000 00001930 00460000 00001950   00470000 00001970 00480000 00001990 *...............&................* R13D920 08
VA66940   00490000 000019B0 004A0000 000019D0   004B0000 000019F0 004C0000 00001A10 *.......................0.<......* R13D940 08
VA66960   004D0000 00001A30 004E0000 00001A50   004F0000 00001A70 00500000 00001A90 *.(.......+.....&.|.......&......* R13D960 08
VA66980   00510000 00001AB0 00520000 00001AD0   00530000 00001AF0 00540000 00001B10 *.......................0........* R13D980 08
VA669A0   00550000 00001B30 00560000 00001B50   00570000 00001B70 00580000 00001B90 *...............&................* R13D9A0 08
VA669C0   00590000 00001BB0 005A0000 00001BD0   005B0000 00001BF0 005C0000 00001C10 *.................$.....0.*......* R13D9C0 08
VA669E0   005D0000 00001C30 005E0000 00001C50   005F0000 00001C70 00600000 00001C90 *.).......;.....&.¬.......-......* R13D9E0 08
VA66A00   00610000 00001CB0 00620000 00001CD0   00630000 00001CF0 00640000 00001D10 *./.....................0........* R13DA00 08
VA66A20   00650000 00001D30 00660000 00001D50   00670000 00001D70 00680000 00001D90 *...............&................* R13DA20 08
VA66A40   00690000 00001DB0 006A0000 00001DD0   006B0000 00001DF0 006C0000 00001E10 *.................,.....0.%......* R13DA40 08
VA66A60   006D0000 00001E30 006E0000 00001E50   006F0000 00001E70 00700000 00001E90 *._.......>.....&.?..............* R13DA60 08
VA66A80   00710000 00001EB0 00720000 00001ED0   00730000 00001EF0 00740000 00001F10 *.......................0........* R13DA80 08
VA66AA0   00750000 00001F30 00760000 00001F50   00770000 00001F70 00780000 00001F90 *...............&................* R13DAA0 08
VA66AC0   00790000 00001FB0 007A0000 00001FD0   007B0000 00001FF0 007C0000 00002010 *.........:.......#.....0.@......* R13DAC0 08
VA66AE0   007D0000 00002030 007E0000 00002050   007F0000 00002070 00800000 00002090 *.'.......=.....&."..............* R13DAE0 08
VA66B00   00810000 000020B0 00820000 000020D0   00830000 000020F0 00840000 00002110 *.......................0........* R13DB00 08
VA66B20   00850000 00002130 00860000 00002150   00870000 00002170 00880000 00002190 *...............&................* R13DB20 08
VA66B40   00890000 000021B0 008A0000 000021D0   008B0000 000021F0 008C0000 00002210 *.......................0........* R13DB40 08
VA66B60   008D0000 00002230 008E0000 00002250   008F0000 00002270 00900000 00002290 *...............&................* R13DB60 08
VA66B80   00910000 000022B0 00920000 000022D0   00930000 000022F0 00940000 00002310 *.......................0........* R13DB80 08
VA66BA0   00950000 00002330 00960000 00002350   00970000 00002370 00000000 00000000 *...............&................* R13DBA0 08
VA66BC0   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000 *................................* R13DBC0 08
VA66BE0   00000000 00000000 00000000 00000000   00980000 00002390 00990000 000023B0 *................................* R13DBE0 08
VA66C00   009A0000 000023D0 009B0000 000023F0   009C0000 00002410 009D0000 00002430 *...............0................* R13DC00 08
VA66C20   009E0000 00002450 009F0000 00002470   00A00000 00002490 00A10000 000024B0 *.......&........................* R13DC20 08
```

C — Start of the DAIT

Figure 5-37 (Part 2 of 5). ALLOCAS Dump

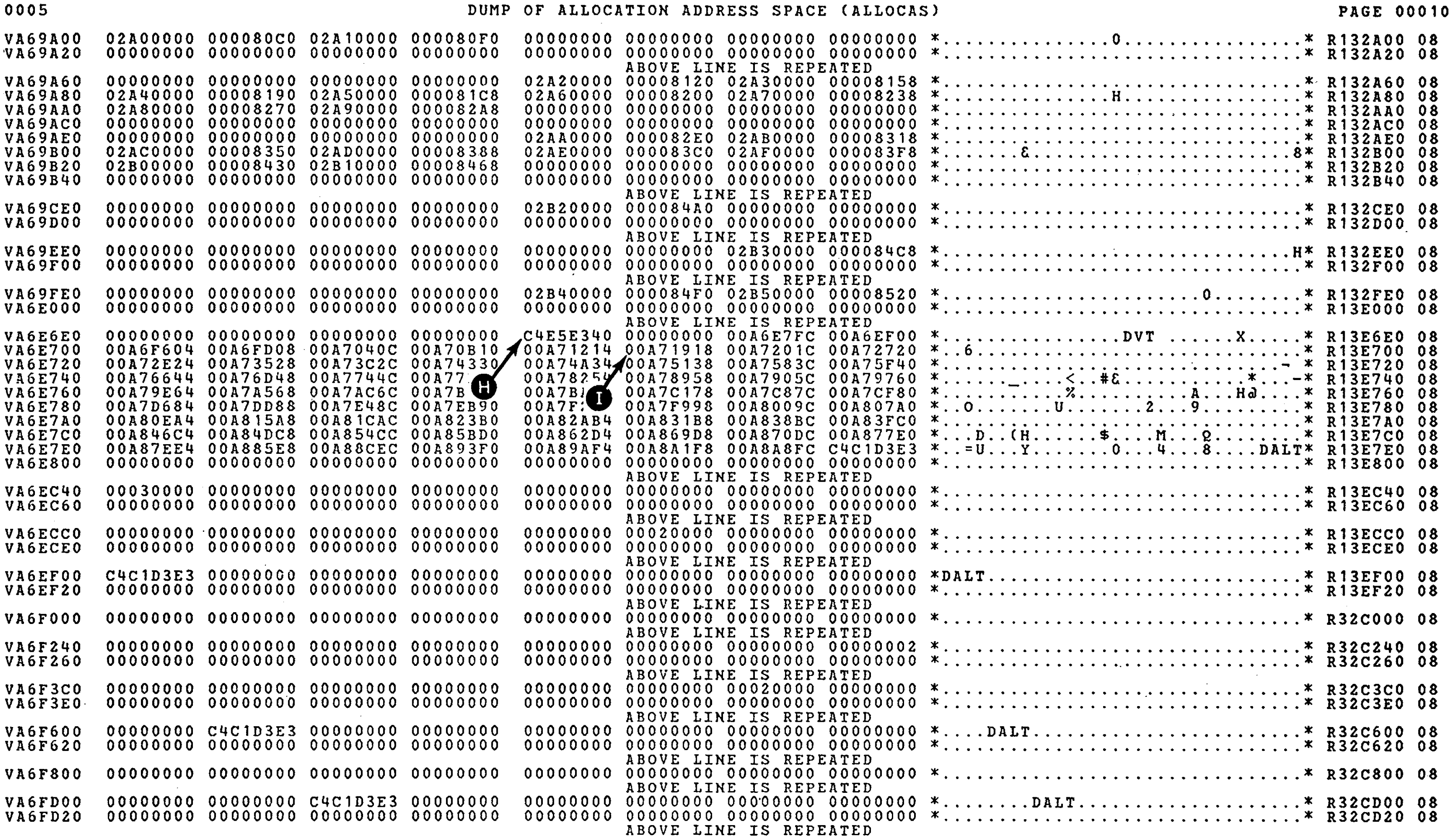
```
0005                                     DUMP OF ALLOCATION ADDRESS SPACE (ALLOCAS)                                        PAGE 00007

VA66C40   00A20000 000024D0 00A30000 000024F0   00A40000 00002510 00A50000 00002530  *...............0................*  R13DC40 08
VA66C60   00A60000 00002550 00A70000 00002570   00000000 00000000 00000000 00000000  *.......&........................*  R13DC60 08
VA66C80   00A80000 00002590 00A90000 000025B0   00AA0000 000025D0 00AB0000 000025F0  *...............................0*  R13DC80 08
VA66CA0   00AC0000 00002610 00AD0000 00002630   00AE0000 00002650 00AF0000 00002670  *.......................&........*  R13DCA0 08
VA66CC0   00B00000 00002690 00B10000 000026B0   00B20000 000026D0 00B30000 000026F0  *...............................0*  R13DCC0 08
VA66CE0   00B40000 00002710 00B50000 00002730   00B60000 00002750 00B70000 00002770  *.......................&........*  R13DCE0 08
VA66D00   00B80000 00002790 00B90000 000027B0   00BA0000 000027D0 00BB0000 000027F0  *...............................0*  R13DD00 08
VA66D20   00BC0000 00002810 00BD0000 00002830   00BE0000 00002850 00BF0000 00002870  *.......................&........*  R13DD20 08
VA66D40   00C00000 00002890 00C10000 000028B0   00C20000 000028D0 00C30000 000028F0  *.........A.......B.......C.....0*  R13DD40 08
VA66D60   00C40000 00002910 00C50000 00002930   00C60000 00002950 00C70000 00002970  *.D.......E.......F.....&.G......*  R13DD60 08
VA66D80   00C80000 00002990 00C90000 000029B0   00CA0000 000029D0 00CB0000 000029F0  *.H.......I.....................0*  R13DD80 08
VA66DA0   00CC0000 00002A10 00CD0000 00002A30   00CE0000 00002A50 00CF0000 00002A70  *.......................&........*  R13DDA0 08
VA66DC0   00D00000 00002A90 00D10000 00002AB0   00D20000 00002AD0 00D30000 00002AF0  *.........J.......K.......L.....0*  R13DDC0 08
VA66DE0   00D40000 00002B10 00D50000 00002B30   00D60000 00002B50 00D70000 00002B70  *.M.......N.......O.....&.P......*  R13DDE0 08
VA66E00   00D80000 00002B90 00D90000 00002BB0   00DA0000 00002BD0 00DB0000 00002BF0  *.Q.......R.....................0*  R13DE00 08
VA66E20   00DC0000 00002C10 00DD0000 00002C30   00DE0000 00002C50 00DF0000 00002C70  *.......................&........*  R13DE20 08
VA66E40   00E00000 00002C90 00E10000 00002CB0   00E20000 00002CD0 00E30000 00002CF0  *.................S.......T.....0*  R13DE40 08
VA66E60   00E40000 00002D10 00E50000 00002D30   00E60000 00002D50 00000000 00000000  *.U.......V.......W.....&........*  R13DE60 08
VA66E80   00E70000 00002D78 00E80000 00002D98   00E90000 00002DB8 00EA0000 00002DD8  *.X.......Y.......Z.............Q*  R13DE80 08
VA66EA0   00EB0000 00002DF8 00EC0000 00002E18   00ED0000 00002E38 00EE0000 00002E58  *.......8........................*  R13DEA0 08
VA66EC0   00EF0000 00002E78 00F00000 00002EA0   00F10000 00002EC8 00000000 00000000  *.........0.......1.....H........*  R13DEC0 08
VA66EE0   00F20000 00002EF0 00F30000 00002F18   00F40000 00002F40 00F50000 00002F68  *.2.....0.3.......4..... .5......*  R13DEE0 08
VA66F00   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000  *................................*  R13DF00 08
                                                         ABOVE LINE IS REPEATED
VA68000   019E0000 000050E8 019F0000 00005118   01A00000 00005148 01A10000 00005178  *......&Y........................*  R1DF000 08
VA68020   01A20000 000051A8 01A30000 000051D8   00000000 00000000 00000000 00000000  *...............Q................*  R1DF020 08
VA68040   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000  *................................*  R1DF040 08
                                                         ABOVE LINE IS REPEATED
VA680E0   00000000 00000000 00000000 00000000   01A40000 00005208 01A50000 00005238  *................................*  R1DF0E0 08
VA68100   01A60000 00005268 01A70000 00005298   00000000 00000000 00000000 00000000  *................................*  R1DF100 08
VA68120   00000000 00000000 00000000 00000000   01A80000 000052C8 01A90000 000052F8  *.......................H.......8*  R1DF120 08
VA68140   01AA0000 00005328 01AB0000 00005358   00000000 00000000 00000000 00000000  *................................*  R1DF140 08
VA68160   00000000 00000000 00000000 00000000   01AC0000 00005388 01AD0000 000053B8  *................................*  R1DF160 08
VA68180   01AE0000 000053E8 01AF0000 00005418   01B00000 00005448 01B10000 00005478  *.......Y........................*  R1DF180 08
VA681A0   01B20000 000054A8 01B30000 000054D8   00000000 00000000 00000000 00000000  *...............Q................*  R1DF1A0 08
VA681C0   00000000 00000000 00000000 00000000   000000   00000000 00000000 00000000  *................................*  R1DF1C0 08
VA681E0   00000000 00000000 00000000 000000     01B40    00005508 01B50000 00005538  *................................*  R1DF1E0 08
VA68200   01B60000 00005568 01B70000 00005      01B80    000055C8 01B90000 000055F8  *.......................H.......8*  R1DF200 08
VA68220   01BA0000 00005628 01BB0000 00005658   00000000 00000000 00000000 00000000  *................................*  R1DF220 08
VA68240   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000  *................................*  R1DF240 08
VA68260   00000000 00000000 00000000 00000000   01BC0000 00005688 01BD0000 000056B8  *................................*  R1DF260 08
VA68280   01BE0000 000056E8 01BF0000 00005718   01C00000 00005748 01C10000 00005778  *.......Y.................A......*  R1DF280 08
VA682A0   01C20000 000057A8 01C30000 000057D8   00000000 00000000 00000000 00000000  *.B.......C.....Q................*  R1DF2A0 08
VA682C0   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000  *................................*  R1DF2C0 08
                                                         ABOVE LINE IS REPEATED
VA683E0   00000000 00000000 00000000 00000000   01C40000 00005808 01C50000 00005838  *.................D.......E......*  R1DF3E0 08
VA68400   01C60000 00005868 01C70000 00005898   01C80000 000058C8 01C90000 000058F8  *.F.......G.......H.....H.I.....8*  R1DF400 08
VA68420   01CA0000 00005928 01CB0000 00005958   01CC0000 00005988 00000000 00000000  *................................*  R1DF420 08
VA68440   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000  *................................*  R1DF440 08
VA68460   00000000 00000000 00000000 00000000   00000000 00000000 01CD0000 000059B8  *................................*  R1DF460 08
VA68480   01CE0000 000059E8 01CF0000 00005A18   01D00000 00005A48 01D10000 00005A78  *.......Y.................J......*  R1DF480 08
VA684A0   01D20000 00005AA8 00000000 00000000   01D30000 00005AD8 01D40000 00005B08  *.K...............L.....Q.M....$.*  R1DF4A0 08
VA684C0   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000  *................................*  R1DF4C0 08
VA684E0   00000000 00000000 00000000 00000000   01D50000 00005B38 01D60000 00005B58  *.................N....$..O....$.*  R1DF4E0 08
VA68500   01D70000 00005B78 01D80000 00005B98   01D90000 00005BB8 01DA0000 00005BD8  *.P....$..Q....$..R....$.......$Q*  R1DF500 08
VA68520   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000  *................................*  R1DF520 08
                                                         ABOVE LINE IS REPEATED
VA68560   00000000 00000000 00000000 00000000   01DB0000 00005BF8 01DC0000 00005C28  *......................$8......*.*  R1DF560 08
```

D – DAIT entry

E – Pointer to UCB

Figure 5-37 (Part 3 of 5). ALLOCAS Dump



```
0005                                      DUMP OF ALLOCATION ADDRESS SPACE (ALLOCAS)                                          PAGE 00010
VA69A00   02A00000 000080C0 02A10000 000080F0  00000000 00000000 00000000 00000000 *...............0................* R132A00 08
VA69A20   00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000 *................................* R132A20 08
                                                        ABOVE LINE IS REPEATED
VA69A60   00000000 00000000 00000000 00000000  02A20000 00008120 02A30000 00008158 *................................* R132A60 08
VA69A80   02A40000 00008190 02A50000 000081C8  02A60000 00008200 02A70000 00008238 *...............H................* R132A80 08
VA69AA0   02A80000 00008270 02A90000 000082A8  00000000 00000000 00000000 00000000 *................................* R132AA0 08
VA69AC0   00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000 *................................* R132AC0 08
VA69AE0   00000000 00000000 00000000 00000000  02AA0000 000082E0 02AB0000 00008318 *................................* R132AE0 08
VA69B00   02AC0000 00008350 02AD0000 00008388  02AE0000 000083C0 02AF0000 000083F8 *.......&.......................8* R132B00 08
VA69B20   02B00000 00008430 02B10000 00008468  00000000 00000000 00000000 00000000 *................................* R132B20 08
VA69B40   00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000 *................................* R132B40 08
                                                        ABOVE LINE IS REPEATED
VA69CE0   00000000 00000000 00000000 00000000  02B20000 000084A0 00000000 00000000 *................................* R132CE0 08
VA69D00   00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000 *................................* R132D00 08
                                                        ABOVE LINE IS REPEATED
VA69EE0   00000000 00000000 00000000 00000000  00000000 00000000 02B30000 000084C8 *...............................H* R132EE0 08
VA69F00   00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000 *................................* R132F00 08
                                                        ABOVE LINE IS REPEATED
VA69FE0   00000000 00000000 00000000 00000000  02B40000 000084F0 02B50000 00008520 *.......................0........* R132FE0 08
VA6E000   00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000 *................................* R13E000 08
                                                        ABOVE LINE IS REPEATED
VA6E6E0   00000000 00000000 00000000 00000000  C4E5E340 00000000 00A6E7FC 00A6EF00 *................DVT ......X.....* R13E6E0 08
VA6E700   00A6F604 00A6FD08 00A7040C 00A70B10  00A71214 00A71918 00A7201C 00A72720 *..6.............................* R13E700 08
VA6E720   00A72E24 00A73528 00A73C2C 00A74330  00A74A34 00A75138 00A7583C 00A75F40 *..............................¬ * R13E720 08
VA6E740   00A76644 00A76D48 00A7744C 00A77     00A78254 00A78958 00A7905C 00A79760 *......_....<..#&...........*...-* R13E740 08
VA6E760   00A79E64 00A7A568 00A7AC6C 00A7B     00A7B    00A7C178 00A7C87C 00A7CF80 *...........%.........A...H@....* R13E760 08
VA6E780   00A7D684 00A7DD88 00A7E48C 00A7EB90  00A7F    00A7F998 00A8009C 00A807A0 *..O.......U.......2...9........* R13E780 08
VA6E7A0   00A80EA4 00A815A8 00A81CAC 00A823B0  00A82AB4 00A831B8 00A838BC 00A83FC0 *................................* R13E7A0 08
VA6E7C0   00A846C4 00A84DC8 00A854CC 00A85BD0  00A862D4 00A869D8 00A870DC 00A877E0 *...D..(H.....$....M...Q........* R13E7C0 08
VA6E7E0   00A87EE4 00A885E8 00A88CEC 00A893F0  00A89AF4 00A8A1F8 00A8A8FC C4C1D3E3 *..=U...Y.......0...4...8....DALT* R13E7E0 08
VA6E800   00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000 *................................* R13E800 08
                                                        ABOVE LINE IS REPEATED
VA6EC40   00030000 00000000 00000000 00000000  00000000 00000000 00000000 00000000 *................................* R13EC40 08
VA6EC60   00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000 *................................* R13EC60 08
                                                        ABOVE LINE IS REPEATED
VA6ECC0   00000000 00000000 00000000 00000000  00000000 00020000 00000000 00000000 *................................* R13ECC0 08
VA6ECE0   00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000 *................................* R13ECE0 08
                                                        ABOVE LINE IS REPEATED
VA6EF00   C4C1D3E3 00000000 00000000 00000000  00000000 00000000 00000000 00000000 *DALT............................* R13EF00 08
VA6EF20   00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000 *................................* R13EF20 08
                                                        ABOVE LINE IS REPEATED
VA6F000   00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000 *................................* R32C000 08
                                                        ABOVE LINE IS REPEATED
VA6F240   00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000002 *................................* R32C240 08
VA6F260   00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000 *................................* R32C260 08
                                                        ABOVE LINE IS REPEATED
VA6F3C0   00000000 00000000 00000000 00000000  00000000 00000000 00020000 00000000 *................................* R32C3C0 08
VA6F3E0   00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000 *................................* R32C3E0 08
                                                        ABOVE LINE IS REPEATED
VA6F600   00000000 C4C1D3E3 00000000 00000000  00000000 00000000 00000000 00000000 *....DALT........................* R32C600 08
VA6F620   00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000 *................................* R32C620 08
                                                        ABOVE LINE IS REPEATED
VA6F800   00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000 *................................* R32C800 08
                                                        ABOVE LINE IS REPEATED
VA6FD00   00000000 00000000 C4C1D3E3 00000000  00000000 00000000 00000000 00000000 *........DALT....................* R32CD00 08
VA6FD20   00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000 *................................* R32CD20 08
                                                        ABOVE LINE IS REPEATED
```

H — Start of the DVT

I — Address of the DALT

Figure 5-37 (Part 4 of 5). ALLOCAS Dump

```
0005                                   DUMP OF ALLOCATION ADDRESS SPACE (ALLOCAS)                                    PAGE 00011
PAGE TABLE FOR SEGMENT  A7

    3391     1331     3541     3209
    32B9     2CC9     32A9     23D9
    36C9     3849     2B29     34E9
    3219     3809     36F9     3709
    0800408000030 1A500000000    08004080000301CA00000000    08004080000301AD00000000    08004080000301AE00000000
    08004080000301AF00000000    08004080000301C500000000    08004080000301B000000000    08004080000301B100000000
    08004080000301C600000000    08004080000301C700000000    08004080000301A300000000    08004080000301AC00000000
    08004080000301B200000000    08004080000301A600000000    08004080000301A700000000    08004080000301B300000000

VA70000   00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000 *................................* R339000 08
                                                        ABOVE LINE IS REPEATED
VA70400   00000000 00000000 00000000 C4C1D3E3  00000000 00000000 00000000 00000000 *............DALT................* R339400 08
VA70420   00000000 0000000C 00000000 00000000  00000000 00000000 00000000 00000000 *................................* R339420 08
                                                        ABOVE LINE IS REPEATED
VA70800   00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000 *................................* R339800 08
                                                        ABOVE LINE IS REPEATED
VA70B00   00000000 00000000 00000000 00000000  C4C1D3E3 00000000 00000000 00000000 *................DALT............* R339B00 08
VA70B20   00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000 *................................* R339B20 08
                                                        ABOVE LINE IS REPEATED
VA71000   00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000 *................................* R133000 0A
                                                        ABOVE LINE IS REPEATED
VA71200   00000000 00000000 00000000 00000000  00000000 C4C1D3E3 00000000 00000000 *....................DALT........* R133200 0A
VA71220   00000000 00000000 00000000 00000000  00000000 00000001 00000000 00000001 *................................* R133220 0A
VA71240   00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000 *................................* R133240 0A
                                                        ABOVE LINE IS REPEATED
VA71560   00000000 00000000 00000000 00000000  00000002 00000000 00000000 00000000 *................................* R133560 0A
VA71580   00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000 *................................* R133580 0A
                                                        ABOVE LINE IS REPEATED
VA716E0   00000000 00000000 00000000 00040000  00000000 00000000 00000000 00000000 *................................* R1336E0 0A
VA71700   00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000 *................................* R133700 0A
                                                        ABOVE LINE IS REPEATED
VA71800   00000000 00000000 C0000000 00000000  00000000 00000000 00000000 00000000 *................................* R133800 0A
                                                        ABOVE LINE IS REPEATED
VA71900   00000000 00000000 00000000 00000000  00000000 00000000 C4C1D3E3 00000000 *........................DALT....* R133900 0A
VA71920   00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000 *................................* R133920 0A
                                                        ABOVE LINE IS REPEATED
VA71C60   00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000005 *................................* R133C60 0A
VA71C80   00010000 00000000 00000000 00000000  00000000 00000000 00000000 00000000 *................................* R133C80 0A
VA71CA0   00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000 *................................* R133CA0 0A
                                                        ABOVE LINE IS REPEATED
VA72000   00000000 00000000 00000000 00000000  00000000 00000000 00000000 C4C1D3E3 *............................DALT* R354000 0A
VA72020   00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000 *................................* R354020 0A
                                                        ABOVE LINE IS REPEATED
VA72720   C4C1D3E3 00000000 00000000 00000000  00000000 00000000 00000000 00000000 *DALT............................* R354720 0A
VA72740   00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000 *................................* R354740 0A
                                                        ABOVE LINE IS REPEATED
VA72800   00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000 *................................* R354800 08
                                                        ABOVE LINE IS REPEATED
VA72E20   00000000 C4C1D3E3 00000000 00000000  00000000 00000000 00000000 00000000 *....DALT........................* R354E20 08
VA72E40   00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000 *................................* R354E40 08
                                                        ABOVE LINE IS REPEATED
```

A – DALTUSE field

B – Start of the DALT

Figure 5-37 (Part 5 of 5). ALLOCAS Dump

Debugging Hints

Hints for debugging specific problem areas are described here including:

- Allocation Serialization
- Subsystem Allocation Serialization
- Device Selection Problems (Non-Abend)
- Address Space Termination
- 0B0 Abend
- 0C4 Abend in IEFAB4FC
- Volume Mount and Verify (VM&V) Waiting Mechanism

Allocation Serialization

Allocation serializes on several types of resources. This has resulted in deadlocks between job steps when a programming change caused incorrect serialization.

Both dynamic allocation and JFCB housekeeping enqueue on data set names. Dynamic allocation enqueues on non-temporary data set names before calling JFCB housekeeping. JFCB housekeeping enqueues on real data set names when it finds via LOCATE, that the specified data set name is an alias; the fully-qualified names of GDG single requests (found via LOCATE); the individual names in a generation data group; and the data set names of temporary, non-VIO data sets. (The initiator enqueues all non-temporary names of JCL-specified data sets before a job starts.) Data set names are dequeued by unallocation, either batch or dynamic, in the last step in which the data set is referenced.

Common allocation enqueues on volume serials of all specific volume requests except for direct-access volumes, which are either permanently resident or reserved. This is done after the allocation of permanently resident or reserved direct-access volumes, that is, following fixed device allocation. The volume serials of demountable volumes allocated to non-specific volume requests are enqueued either when the volume is allocated (if the volume is already mounted) or when the volume is mounted (if allocation mounts and verifies it). (When there is a nonspecific request for tape, OPEN enqueues the tape-volume serial numbers because allocation only waits for direct-access volumes to be mounted.) Before actually allocating a device, common allocation serializes the status of devices by enqueuing on several resources all with the major name SYSIEFSD. The minor names and functions serialized are as follows:

1. Q4 - to serialize device allocation with VARY offline processing, which is actually done by common allocation
2. CHNGDEVS - to serialize device allocation with device unloading done by the UNLOAD operator command and JES3
3. DDRDA - to serialize device allocation with dynamic device reconfiguration (DDR) processing of direct access devices and
4. DDRTPUR - to serialize device allocation with DDR processing of tape and unit record devices

These four resources are enqueued for shared use by allocation and for exclusive use by the other functions. Within common allocation, these resources, with the exception of Q4, are dequeued when allocation must wait on an allocation recovery WTOR or on an allocation group.

Allocation serializes, via an internal mechanism, the processing of all devices except direct access devices containing non-demountable (permanently mounted or reserved) volumes. The serialization unit is an allocation group. This serialization is done to serialize the device allocation in one address space with that in another. Group serialization is exclusive, that is, it prevents an allocation in a given address space from considering the same device that an allocation in another address space is considering. All allocations serialize on groups in the same order; this order is specified at sysgen and is represented in the csect PREFTAB, which is part of the load module IEFEDTTB. PREFTAB is simply a list of generic device types.

To serialize changes to a specific UCB, allocation and unallocation always obtain the local and CMS locks before setting fields in the UCB.

Dynamic allocation serializes with itself so that only one dynamic allocation may proceed in an address space. This is done by an enqueue for exclusive use on major name SYSZTIOT, the minor name consisting of the 2-byte ASID and 4-byte address of the DSAB QDB.

Subsystem Allocation Serialization

Allocation does not serialize when processing subsystem data set requests, but provides the capability whereby a subsystem may serialize its own requests if it so desires. The mechanism to do this is the subsystem allocation sequence table (SAST). A skeletal SAST is built during subsystem interface initialization to define the order in which subsystems are to be invoked for the allocation of subsystem data sets. During common allocation processing the subsystem requests are sorted by subsystem. Using the sequence defined by the SAST, all requests for a given subsystem are passed to that subsystem for allocation before the next subsystems requests are processed. Thus a subsystem can serialize its allocation processing in order to prevent deadlocks.

Device Selection Problems (Non-Abend)

The device selection logic of common allocation is heavily dependent on the eligible devices table (EDT) which is built at SYSGEN. The EDT describes the unit eligibility of any unit name that may be specified either via JCL or dynamic request. Users have in the past tried to modify the EDT without doing either a full or an I/O SYSGEN. Modification of the EDT can result in incorrect allocation, for example, allocation of a 3330 request to a 2314, or failure of a request or job step with no error indicated. If such a device selection error occurs after modification of an EDT, the modification is suspect and should be carefully verified by consulting the EDT descriptions in the *OS/VS2 System Logic Library* section on Data Areas, and/or EDT mapping in *Data Areas* (microfiche), via mapping macro IEFZB421.

EDT Verification Routine

The eligible device table (EDT) can be verified by using the EDT verification routine IEFEB400. This module compares the EDT against the UCBs for a given system. The routine must execute on the system with the UCBs to be compared. For teleprocessing devices, only the device class is verified. For additional information about the EDT verification routine, refer to *Job Management* and *System Generation.*

Address Space Termination

When an address space is being abnormally terminated, the allocation end-of-memory resource manager, IEFAB4E5, gets control. This routing releases any allocation groups held by the address space and unallocates any units allocated to the address space. In the case where the allocation address space (ALLOCAS) is not active at the time IEFAB4E5 gets control, allocation cannot unallocate shareable, direct access units because they might be allocated to more than one address space.

The UCBASID field (at offset X'E') in the common UCB extension contains:

- The ASID of the address space that is allocated to the unit, or
- In the case of shareable direct access units, the ASID of the last address space that allocated the unit.

Cleanup: When an allocation routine abnormally terminates, the ESTAE routine IEFAB4ED receives control. Abnormal termination can result from an abend, cancel, or machine check. In each case, IEFAB4ED performs recovery processing, takes a dump (if there is a possibility that the error occurred in allocation), and routes control to other exit routines for more specific error recovery. If a cancel is issued during dynamic allocation, a dump is always taken and an X'x22' or X'x3E' abend occurs.

0B0 Abend

0B0 abends have occurred in allocation more than once. The code is issued by the SWA manager, which handles the reading, writing, and assigning of SWA records. Allocation requests all these functions of the SWA manager. Two situations cause allocation to receive a 0B0 abend from the SWA manager:

1. The address of a SWA record to be read or written, in behalf of allocation, has been overlaid. Allocation usually obtains a SWA virtual address (SVA) to read or write from another SWA record. When such an SVA has been overwritten by a scheduler sub-component, a 0B0 abend may occur.

2. A 0B0 abend will occur when allocation assigns an SVA for a record and then uses the SVA to attempt to read the record without first having written the record.

0C4 Abend in IEFAB4FC

This error always occurs when the device type in a UCB is changed from one generic type to another, and when a JCL statement or dynamic request specifies that particular unit. This error is usually the result of an improper IOGEN or the zapping of the EDT or UCB.

This error can be diagnosed by running the EDT verification routine (IEFEB400).

```
//EDT  JOB
//VERIFY  EXEC  PGM=IEFEB400
//EDT  DD  DSN=SYS1.LPALIB,DISP=SHR
//SYSPRINT  DD  SYSOUT=A
```

Volume Mount and Verify (VM&V) Waiting Mechanism

Volume mount and verify must wait for direct access volumes to be mounted so that the labels can be verified, and so that allocation can enqueue the volume serials for non-specific volume requests and obtain space (for new data set requests). In order to allow for several allocations to be waiting simultaneously, the control block structure shown in Figure 5-38 is set up by VM&V.

Each address space waiting for at least one direct access volume to be mounted has its own mount verification control area (MVCA), MVCA extension (MVCAX), one or more mount entries, and, in each mount entry, one or more UCB entries. Each MVCAX contains an ECB. When an allocation is waiting for a direct-access volume to be mounted, VM&V waits for this ECB in behalf of the allocation. The MVCA chain is anchored in the allocation/termination communication area (ATCA) in the nucleus. The ATCA is pointed to by location CVTQMWR in the CVT. All devices on which allocation waits for a device end (volume mount), will have the scheduler attention table index placed in their UCBs (at +3 in the common UCB extension). The index is X'0C'.

Any destruction of the MVCA/MVCAX structure causes one or more allocations to wait "permanently." The wait is not truly permanent, however, because VM&V also waits for (in a batch environment) the cancel ECB (in the CSCB-command scheduling control block), which is posted when the operator cancels a job. In a dynamic environment, VM&V waits for a WTOR ECB, in which case the operator can, via reply, cancel the single mount but not the job.

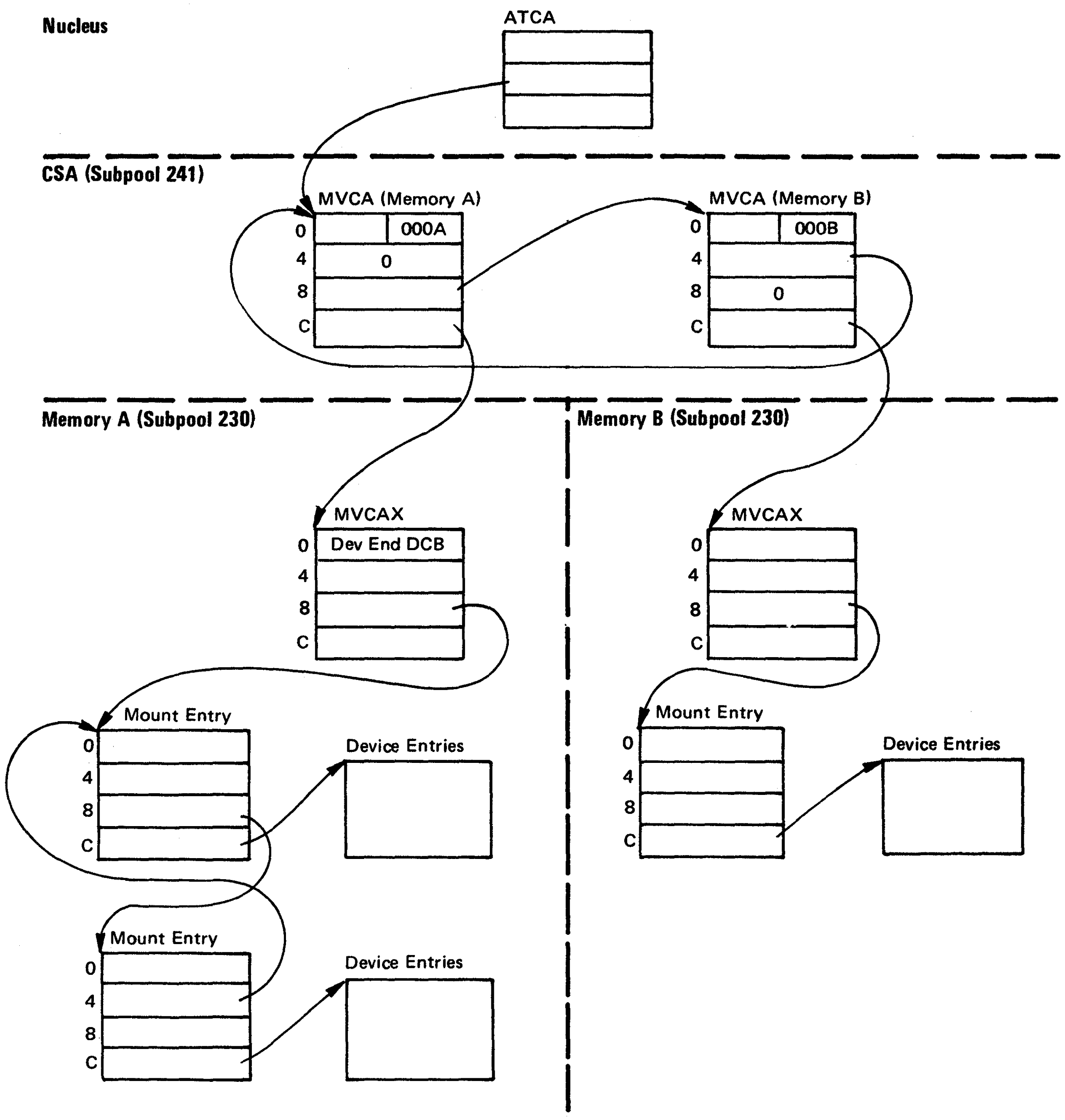


Figure 5-38. VM&V Control Block Structure

Allocation/Unallocation Reason Codes

The reason codes listed here are divided into three groups:

- Reason codes set by batch and common allocation modules and by JFCB housekeeping modules.
- Reason codes set by unallocation modules.
- Reason codes set by dynamic allocation modules.

Common and Batch Allocation and JFCB Housekeeping Reason Codes

The reason codes set by common and batch allocation and by JFCB housekeeping are divided into step-related reason codes and DD-related reason codes.

The following are DD-related error reason codes set by allocation and JFCB housekeeping modules and placed in the SIOTRSNC field of the SIOT. The reason codes serve as an index into message module IEFBB4M3. The prologue of IEFBB4M3 lists the modules which detect the error conditions.

Reason Code	Error Reason Code	Message	Meaning
1	1700	IEF212I	Data set not found.
2	0244	IEF371I	Telecommunication device is not accessible.
3	0210	IEF211I	Unable to ENQ on data set name.
4	020C	IEF211I	Unable to ENQ on data set name.
5	0458	IEF365I	Referenced data set name is GDG ALL.
6	0214	IEF702I	Unable to allocate.
7	*	IEF221I	Invalid backward reference to a step.
8	021C	IEF210I	Invalid UNIT parameter.
9	0480	IEF195I	Maximum number of devices for statement exceeded.
10	0224	IEF192I	Not enough eligible devices.
11	0398	IEF194I	Volume sequence number incorrect.
12	4714	IEF246I	Insufficient space on storage volumes.
13	*	IEF721I	Protection conflict in ISAM requests (SU 32 only).
14	*	IEF372I	VOL=REF to unresolved DD.
15	*	IEF318I	UNIT=AFF to new direct access data set.
16	47A8	IEF719I	Data set previously defined (SU 32 only).
17	47AC	IEF720I	User not authorized to define this data set (SU 32 only).
18	*	IEF688I	Nullfile and DSNAME conflict in ISAM concatenation.
19	reserved		
20	039C	IEF245I	Inconsistent unit name and volser.
21	0228	IEF474I	Unit or volume in use by system task.
22	4704	IEF253I	Duplicate data set name on direct access volume.
23	4708	IEF254I	Insufficient space in VTOC.
24	470C	IEF193I	Space not obtained because of I/O error.
25	4710	IEF256I	Absolute track not available.
26	4714	IEF257I	Space requested not available.
27	4718	IEF258I	Invalid record length in SPACE parameter.
28	*	IEF260I	Incorrect DSORG or DISP.
29	*	IEF261I	No prime area request for ISAM data set.
30	*	IEF262I	Prime area must be requested before overflow area.
31	*	IEF263I	Space request not on cylinder boundary.
32	*	IEF264I	Duplication of DSNAME element.
33	4734	IEF266I	Invalid JFCB or partial DSCB pointer.
34	4838	IEF140I	Directory space request too large.
35	reserved		
36	4740	IEF273I	Invalid user label request.

* - means that the error cannot be set in dynamic allocation.

37	reserved		
38	474C	IEF127I	No SPACE parameter or zero space request at ABSTR 0.
39	*	IEF128I	Invalid request for ISAM index.
40	*	IEF129I	Multivolume index request.
41	*	IEF130I	DSNAME element wrong.
42	*	IEF131I	Multivolume OVFLOW request.
43	*	IEF132I	CYL and ABSTR conflict in SPACE parameter.
44	*	IEF133I	CYL and CONTIG conflict in SPACE parameter.
45	*	IEF134I	Subparameter wrong in SPACE parameter.
46	476C	IEF135I	Zero primary space request.
47	*	IEF136I	Index area requested twice.
48	4780	IEF267I	Space request for directory larger than primary space request.
49	*	IEF145I	Space request not ABSTR for DOS volume.
50	*	IEF141I	Index request did not precede prime request.
51	*	IEF143I	Last concatenated DD card unnecessary or invalid.
52	035C	IEF366I	Relative GDG generation number contains syntax error.
53	0390	IEF219I	GDG group name exceeds 35 characters.
54	0394	IEF286I	DISP field incompatible with data set name.
55	*	IEF466I	Unable to recover from DADSM failure.
56	0218		Mounting required but not allowed.
57	0494	IEF704I	Can't access SYSCATLG data set on CVOL.
58	022C	IEF475I	Volume on ineligible permanently resident or reserved device.
59	0214	IEF467I	Units required not available - waiting not allowed.
60	0220	IEF710I	Volumes required not available - waiting not allowed.
61	4794	IEF476I	Data sets overlap in VTOC.
62	4798	IEF477I	DOS split cylinder data sets overlap.
63	479C	IEF478I	Possible VTOC error.
64	*	IEF479I	VTOC error on second or later volume of ISAM prime data set.
65	*	IEF481I	Same unit request twice - conflicts exist.
66	0230	IEF482I	Permanently resident or reserved volume on requested unit.
67	0488	IEF217I	Volume containing pattern DSCB not mounted.
68	048C	IEF218I	Pattern DSCB record not found in VTOC.
69	47A4	IEF703I	New data set requested on DOS stacked pack format volume.
70	0214		Can't wait for offline devices.
71	0240	IEF483I	Requested device is a console.
72	04B8	IEF726I	MSS not initialized.
73	04BC	IEF725I	MSS select error.
74	0234	IEF484I	More units required for demand request.
75	*	IEF493I	Invalid JOBCAT or STEPCAT parameters.
76	*	IEF492I	Invalid data set name for JOBCAT or STEPCAT.
77	*	reserved	
78	0470		Unauthorized requestor of subsystem data set.
79	046C	IEF480I	Invalid destination requested.
80	*	reserved	
81	0490	IEF701I	Error changing allocation assignments.
82	17FF	IEF213I	Error processing cataloged data set.
83	022C	IEF687I	Requested volume mounted on JES3-managed unit.
84	024C	IEF752I	The request for a subsystem data set was failed by the subsystem attempting to allocate the request.
85	0250	IEF752I	
86	03A0	IEF752I	
87	04A4	IEF752I	
88	0484	IEF752I	
89	7700	IEF752I	
90	04A8	IEF753I	A SUBSYS parameter specified a subsystem which does not support the allocation of subsystem data sets.
91	04AC	IEF754I	The subsystem requested on a SUBSYS parameter was not operational.

* - means that the error cannot be set in dynamic allocation.

92	04B0	IEF755I	The subsystem requested on a SUBSYS parameter does not exist.
93	7704	IEF756I	A system error occurred in allocating a subsystem data set.
94	reserved		
95	04B4	IEF740I	Data set/volume could not be RACF protected - RACF not active.
96	03A4	IEF741I	Protect request failed - invalid data set/ volume specification.
97	04C0	IEF741I	Data set/volume could not be RACF protected - user not defined to RACF.
98	0258	IEF198I	There are not enough unrestricted devices to satisfy the request. (Restricted devices are defined in *OS/VS2 System Programming Library: System Generation Reference.*)
99	47B0	IEF274I	A request for space was rejected by the installation exit.
100	47B4	IEF275I	A request for space cannot be satisfied on any eligible volume(s).
101	025C	IEF464I	The device is boxed and cannot be allocated.
102	47B8	IEF276I	RACF DEFINE request with modeling and the required model could not be found.

The following are step-related error reason codes set by allocation and JFCB housekeeping modules in an area pointed to by the allocation work area (ALCWA). With the exception of reason code 1, the reason codes serve as an index into message module IEFBB4M2. The prologue of IEFBB4M2 lists the modules which detect the error condition. Reason code 1 is set by IEFAB469 and is returned to dynamic allocation.

Reason Code	**Error Reason Code**	**Message**	**Meaning**
1	023C		Catalog not mounted.
2	0204	IEF180I	GETMAIN error.
3	0220	IEF713I	MSS volume not available.
4	*	reserved	
5	0484	IEF251I	Job cancelled.
6	0238	IEF240I	No space in TIOT or in TCTIOT.
7	0220	IEF485I	Volumes not available and waiting not allowed.
8	049C	IEF714I	MSS volume not defined.
9	0474	IEF473I	System Resources Manager error.
10	0248	IEF716I	Unable to mount MSS volume.
11	0450	IEF491I	Number of DDs exceeds 1635.
12	172C	IEF363I	Not enough storage for processing cataloged data set.
13	1718	IEF364I	Permanent I/O error processing cataloged data set.
14	670C	IEF367I	I/O error obtaining pattern DSCB.
15	0478	IEF465I	Unable to allocate subsystem data set.
16	047C	IEF456I	Error issuing ESTAE macro.
17	0214	IEF700I	Environment changed - no longer able to allocate.
18	0490	IEF701I	Error changing allocation assignments.
19	0468	IEF361I	Unable to allocate private catalog.
20	*	IEF362I	Unable to unallocate private catalog.
21	*	IEF202I	Step not run because of condition codes.
22	*	IEF202I	Step not run because of condition codes.
23	0498	IEF715	MSS volume inaccessible.
24	04A0	IEF717I	Specified virtual volume group (MSVGP) name does not exist.
25	*	IEF718I	Space or virtual volume group (MSVGP) required for nonspecific MSS request.
26	024C	IEF751I	The job was failed in allocation by a subsystem processing a request to allocate one or more subsystem data sets.
27	0250	IEF751I	
28	03A0	IEF751I	
29	04A4	IEF751I	
30	7700	IEF751I	

* - means that the error cannot be set in dynamic allocation.

Common and Batch Unallocation Reason Codes

The following reason codes are set by common and batch unallocation modules. Reason codes 1, 2, and 4 serve as an index into message module IEFBB4M5. Reason code 3 does not result in a message; it is returned to dynamic allocation.

Reason Code	Message	Meaning	Module Setting
1	IEF468I	GETMAIN error.	IEFBB410, IEFBB414, IEFBB416, IEFAB4A0
2	IEF469I	Data sets not released.	IEFAB4A0, IEFAB4A6
3	-------	Volumes not released. (Dynamic allocation only).	IEFAB4A0, IEFAB4A8
	IEF724I	Step catalogs not allocated. (Warm start only).	IEFAB4A2
4	IEF456I	Error issuing ESTAE macro.	IEFBB410, IEFAB4A0

In addition, IEFAB4A2 (disposition processor) receives return codes returned by the data management catalog and scratch functions (called by IEFAB4A2 to perform disposition processing). If the allocation is dynamic, these return codes are returned to dynamic allocation as reason codes in a field in the unallocation request block. For batch allocation, the return code is converted to a code for a disposition message.

Dynamic Allocation Reason Codes

For a description of dynamic allocation reason codes, refer to the topics "Informational Reason Codes" and "Error Reason Codes" in *OS/VS2 System Programming Library: Job Management.*

JES2

Note to Readers

Component information for JES2 is deleted from this book because it is duplicated in the JES2 logic book.

See the *JES2 Logic* book for diagnostic information for JES2.

Subsystem Interface (SSI)

In the course of HASP/ASP installation, hooks were put into OS and SVS operating systems to establish an interface. With the job entry subsystem (JES), an interface was designed to eliminate the need for these hooks.

The subsystem interface (SSI) is primarily used to communicate with the job entry subsystem (either JES2 or JES3), but is flexible enough to communicate with any subsystem.

System Initialization Processing

At system generation, the name of the primary job entry subsystem and secondary subsystems are listed on the SCHEDULR macro and put in the job entry subsystem names table (CSECT IEFJESNM). Subsystems may be specified in the subsystem names table (load module IEFJSSNT), and also, you can specify subsystems in the IEFSSNxx members of SYS1.PARMLIB. The PARMLIB suffixes are specified at IPL.

The master scheduler base initialization module (IEEVIPL) gives control to the subsystem interface initialization module (IEFJSINT). IEFJSINT calls the subsystem service routine (IEFJSBLD) to:

- Build a subsystem communication vector table (SSCVT) for each unique name in the JES names table and in the subsystem names table.
- Build and initialize the subsystem vector table (SSVT) for the master subsystem.
- Build the subsystem allocation sequence table (SAST) for later use in allocating subsystem data sets.

IEFJSINT then returns control to IEEVIPL.

The job entry subsystem builds and initializes its own SSVT when the system is initialized. All other subsystems must do likewise. A subsystem can be initialized as follows:

- By being started (for example: START JES2) or,
- By having an initialization routine specified in the subsystem names table, or in the IEFSSNxx SYS1.PARMLIB record.

Additional subsystem initialization is performed by module IEFJSIN2, which is invoked by IEEMB860. IEFJSIN2 calls the PARMLIB read routine (IEEMB878) to read each record from the IEFSSNxx members. All records read are passed to the positional parse routine (IEEMB882). IEFJSIN2 also calls IEFJSBLD to perform the following:

- Link to the subsystem initialization routines specified in the subsystem name table.

- Build a SSCVT for each unique subsystem specified in the IEFSSNxx members and link to the initialization routine if any is specified on the PARMLIB record.
- Build a hash table containing SSCVT addresses for use in processing SSI requests.
- Rebuild the SAST if additional subsystems were defined in SYS1.PARMLIB.

IEFJSBLD and IEFJSIN2 call the subsystem initialization message writer (IEFJSIMW) to issue operator messages. The text of the messages is contained in IEFJSIMM. The following messages are detected by IEFJSBLD:

- Message IEE730I indicates that a duplicate subsystem was specified for an SSCVT request.
- Message IEE859I is issued for each subsystem initialization routine that could not be found.
- Message IEF759I indicates that a GETMAIN failed, or that an abend occurred.

Note: It is the responsibility of the subsystem initialization routine to inform the operator of errors and to recover from errors in the initialization routine. If the subsystem initialization routine fails to recover from an error, message IEF759I is issued to indicate the abend and the next subsystem is processed. The failing subsystem might not be completely initialized.

The following messages are detected by IEFJSIN2:

- Message IEF758I is issued if the subsystem names table (IEFJSSNT) could not be found, and it is issued if any IEFSSNxx members could not be found in SYS1.PARMLIB. Message IEF758I is also issued to indicate an abend in IEFJSIN2 processing.
- Message IEF760I is issued when IEEMB882 (parse routine) finds a syntax error in a subsystem SYS1.PARMLIB record or when IEFJSIN2 finds an invalid subsystem name.

Subsystem Interface Major Control Blocks

Subsystem interface's major control blocks are the JES control table (JESCT), the subsystem allocation sequence table (SAST), the subsystem hash table (SHAS), the subsystem communications vector table (SSCVT), the subsystem vector table (SSVT), the subsystem information block (SSIB), the subsystem options block (SSOB), and the extension to the SSOB or function dependent area. The following table summarizes each of these control blocks.

Control Block	Created By	Subpool	Key	Size	Pointed To By	Function	Mapping Macro
JESCT	SYSGEN	NUCLEUS	0	108 bytes	CVT	Contains information needed by the Subsystem Interface and addresses of Scheduler routines.	IEFJESCT
SAST	IEFJSBLD	241	0	Note 1	JESCT	Defines the order in which subsystems will be invoked to allocation subsystem data sets.	IEFJSAST
SHAS	IEFJSBLD	241	0	76 bytes	JESCT	Contains a hash table of SSCVT addresses, used by subsystem interface to locate a specific subsystem.	IEFSSHSH
SSCVT	IEFJSBLD	241	0	36 bytes	JESCT	Identifies each subsystem defined to the system and points to the SSVT for each subsystem.	IEFJSCVT
SSVT	Subsystem owning the SSVT, at Initialization of subsystem.	Any - determined by the subsystem	Any	Note 2	SSCVT	Contains the indication of functions of a subsystem and the addresses of the routines that perform those functions.	IEFJSSVT
SSIB	The user of Subsystem Interface	User's Subpool	Any	36 bytes	SSOB, JSCB	Identifies the subsystem to the Subsystem Interface and passes information between the subsystem and its caller.	IEFJSSIB
SSOB	The user of Subsystem Interface.	User's Subpool	Any	20 bytes	SSWA, IEL	The parameter for the Subsystem Interface.	IEFJSSOB
Function Dependent Area	The user of Subsystem Interface	User's Subpool	Any	Variable	SSOB	Passes information to the function of the subsystem the user wishes to invoke.	IEFJSSOB

Notes:

1. *The SAST size is 8 bytes plus 12* (the number of subsystems in the SSCVT chain).*
2. *The SSVT size is 260 bytes plus 4* (the number of functions supported by the subsystem). Minimum size is 264 bytes, maximum - 1284 bytes.*

Control Block Usage is shown in Figure 5-39, Figure 5-40, and Figure 5-41.

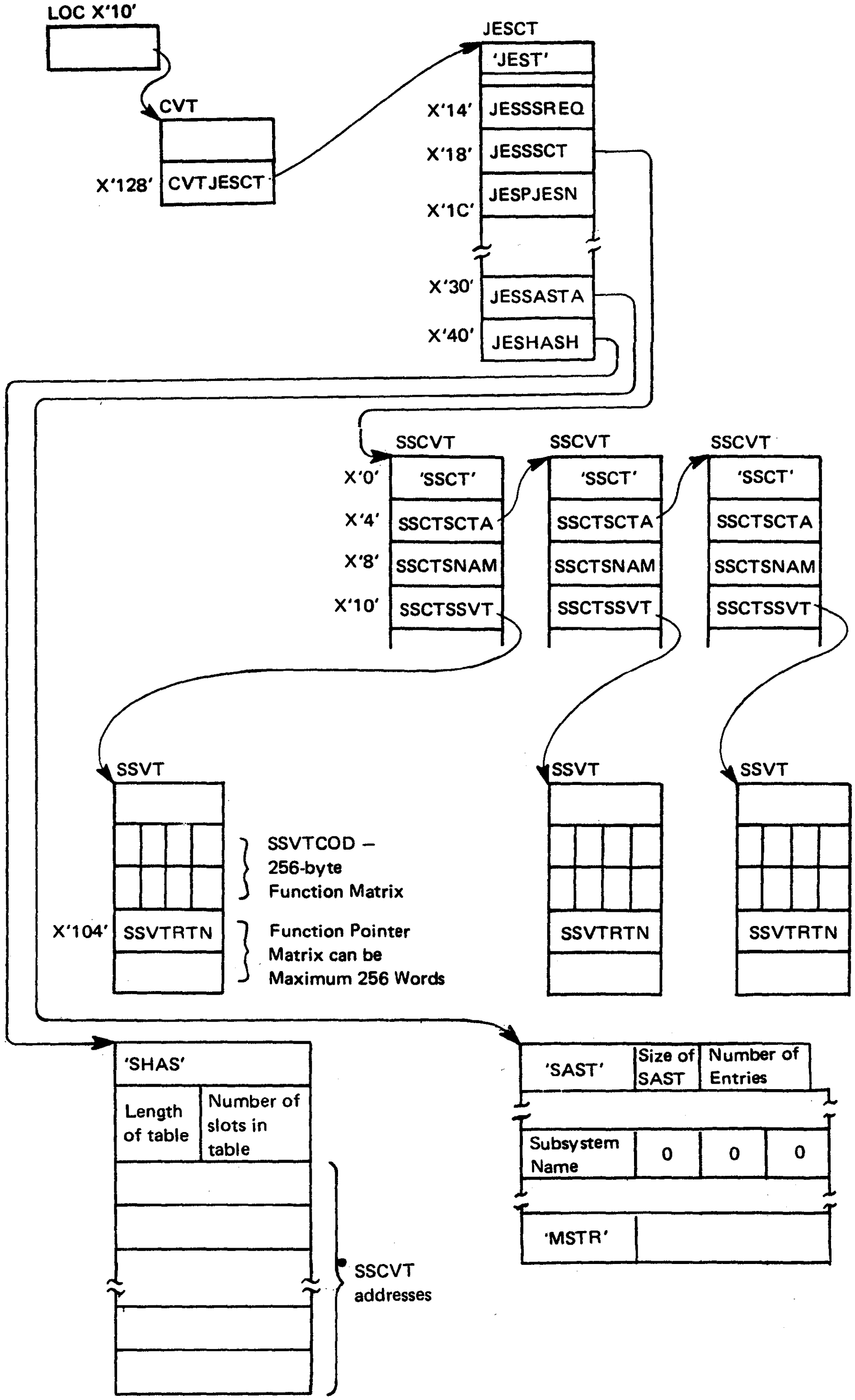


Figure 5-39. Subsystem Interface Control Block Usage

Because the hashing algorithm can produce synonyms, a synonym chain exits as shown in Figure 5-40. Field SSCTSYN is a synonym pointer.

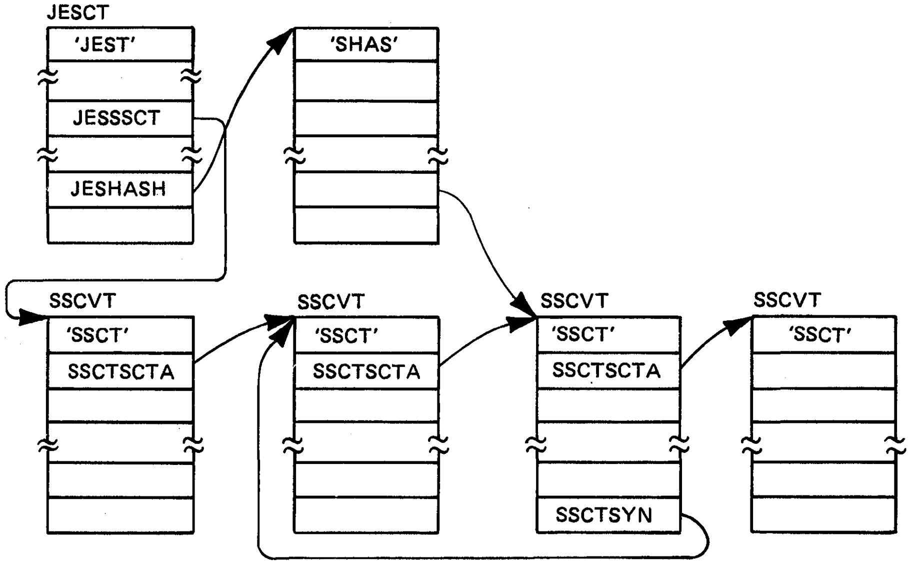


Figure 5-40. Control Block Usage With Synonyms

Requesting Subsystem Services

To request subsystem services, a system routine enters the correct function code (see Subsystem Interface Summary in *OS/VS2 System Logic Library)* in the subsystem options block (SSOB), and the name of the desired subsystem in the subsystem information block (SSIB). The IEFSSREQ macro is then issued, causing control to pass to the subsystem interface routine IEFJSREQ. The specified function code and subsystem name indicates to the interface routine the subsystem routine to receive control.

Invoking the Subsystem Interface

Storage is acquired for the SSIB, the SSOB, and the function dependent area of the SSOB if required. The following entries are made in the SSOB header:

SSOBID - 'SSOB'.

SSOBLEN - The length of the SSOB header.

SSOBFUNC - The function ID of the function to be invoked.

SSOBSSIB - The address of the SSIB or zero. Zero means that the life-of-job SSIB is to be used. Its address is in the active JSCB, field JSCBSSIB. The request will thus be directed to the subsystem that started the initiator under which the job is running. (See Figure 5-42).

SSOBINDV - The address of the function dependent area, or if not needed by the function, zero.

The following entries are made in the SSIB:

SSIBID - 'SSIB'.

SSIBLEN - The length of the SSIB.

SSIBFLG1 - The no-SVC flag (SSIBNSVC) must be set when SVCs cannot be issued by the subsystems invoked (such as SRB mode).

SSIBSSNM - The name of the subsystem to which the request is being made.

SSIBJBID - (If the function requires this field.)

SSIBDEST - (If the function requires this field.)

The entries made in the function dependent area are:

length - The length of the function dependent area (first halfword).

* - Any fields required by the function.

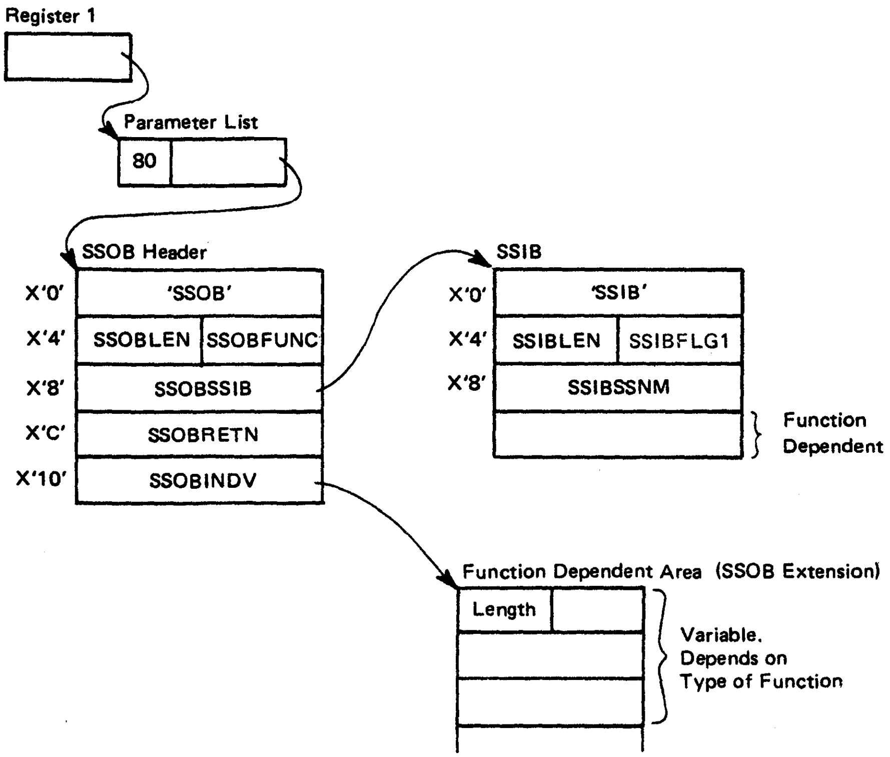
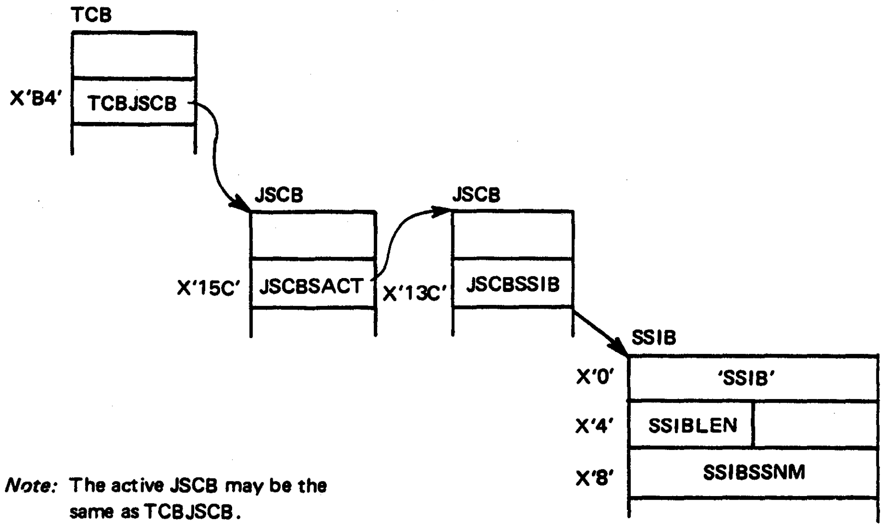


Figure 5-41. Control Block Structure for Invoking Subsystem Interface



Note: The active JSCB may be the same as TCBJSCB.

Figure 5-42. Finding the SSIB for a Job When SSOB Pointer is Zero

Register 1 points to a one-word parameter list which points to the SSOB. (See Figure 5-41).

Macro IEFSSREQ is invoked which passes control to routines which handle the Subsystem Interface request. The communications vector table (CVT) and the JES control table (JESCT) must be mapped if IEFSSREQ is invoked.

The subsystem interface returns a code in register 15. Possible return codes are:

0 - Successful completion - request went to subsystem
4 - Subsystem does not support this function
8 - Subsystem exists, but is not active
12 - Subsystem does not exist
16 - Function not completed - disastrous error
20 - Logical error (such as invalid SSOB format, incorrect length)

The field SSOBRETN in the SSOB contains a return code from the subsystem if the request was successful. The return code depends on the function being invoked (see the SSOB description in the *Debugging Handbook.*

Logic Flow Examples

This section provides an overall logic flow from a task making a request, through the subsystem interface to the subsystem, and then back to the task. Two examples are described.

Notifying a Single Subsystem

1. A task (TSO/cancel) wants to inform JES2 that a job is to be canceled.

2. The task creates an SSOB, SSIB, and a function dependent area.

   a. The SSOB is filled in. A function code of 2 is used. (See *OS/VS2 System Logic Library,* for a complete function code list.

b. The SSIB is filled in. The subsystem name is JES2.

c. The function dependent area is filled in with the necessary information that the subsystem needs for this type of request.

3. Macro IEFSSREQ is invoked which branches to module IEFJSREQ (IEFJSREQ's address is in the JESCT). Register 1 points to a parameter list which points to the SSOB.

4. IEFJSREQ checks:

   a. Are the pointers to the SSOB and SSIB valid? No, then return with a return code of 16 in register 15.

   b. Are the formats of the SSOB and SSIB correct? No, then return with a return code of 20 in register 15.

   c. Find the requested subsystem's SSCVT. If not found, return with a return code of 12 in register 15.

   d. Find the requested subsystem's SSVT. If not found, return with a return code of 8 in register 15.

   e. Is the requested function code valid? No, then return with a return code of 16 in register 15.

   f. Is the requested function code supported by the requested subsystem? No, then return with a return code of 4 in register 15.

   g. Index into the SSVT and get the address of the function routine.

   h. Branch to the function routine. Register 0 = Address of the SSCVT, register 1 = Address of the SSOB.

5. Module HASPSSSM at label HOSCANC receives control. It is the function routine for JES2 for function code 2 (CANCEL request).

   a. Process the request and place a return code in the SSOB (SSOBRETN).

   b. Return codes for this function code are as follows:

0 -	CANCEL completed.
4 -	Job name not found.
8 -	Invalid JOBNAME/JOB ID combination.
12 -	Job not canceled - Duplicate jobnames and no job ID given.
20 -	Job not canceled - Job is on output queue.
24 -	Job ID with invalid syntax for subsystem.
28 -	Invalid CANCEL request. Cannot cancel an active TSO user or a started task.

6. Control is then returned to the requesting task directly from the function routine. The task then examines register 15 and SSOBRETN and acts accordingly.

Notifying All Active Subsystems

1. A task wants to notify all active subsystems of a WTO message.

2. The task creates an SSOB and a function dependent area. No SSIB need be created if the task's life-of-job SSIB has the master subsystem's name (MSTR) in it. If it does not, and that SSIB is used, only one subsystem would be notified. The SSOB and the function dependent area are filled in. A function code of 9 is used. (A list of all function codes is in *OS/VS2 System Logic Library*.)

3. Macro IEFSSREQ is invoked which branches to IEFJSREQ (address is in the JESCT). Register 1 points to a parameter list which points to the SSOB.

4. IEFJSREQ checks:

   a. Are the pointers to the SSOB and SSIB valid? No, return with a return code of 16 in register 15.

   b. Are the formats of the SSOB and SSIB correct? No, return with a return code of 20 in register 15.

   c. Find the requested subsystem's SSCVT. If not found, return with a return code of 12 in register 15.

   d. Find the requested subsystem's SSVT. If not found, return with a return code of 8 in register 15.

   e. Is the requested function code valid? No, return with a return code of 16 in register 15.

   f. Is the requested function code supported? No, return with a return code of 4 in register 15.

   g. Index into the SSVT and get the address of the function routine.

   h. Branch to the function routine. Register 0 = address of the SSCVT. Register 1 = address of the SSOB.

5. If the SSIB contains the name of the master subsystem and the function code is defined as a broadcast code in the master scheduler's SSVT, then module IEFJRASP is the function routine that receives control.

   a. IEFJRASP makes a copy of the SSIB.

   b. For each SSCVT, IEFJRASP determines if the subsystem is active by checking for a nonzero subsystem vector table pointer (SSCTSSVT). For each active subsystem except the master subsystem, IEFJRASP copies the name of the subsystem into the SSIB copy and then invokes IEFSSREQ.

   c. The highest return code from the subsystems is placed in the requesting task's SSOB, and the lowest return code from the subsystem interface is put in register 15.

6. Control is then returned to the requesting task directly from the function routine. The task then examines register 15 and SSOBRETN and acts accordingly.

Debugging Hints

- Paging must be possible at the time subsystem interface is entered because the code for subsystem interface may not already be paged in at the time the call is made.

- For the same reason, the processor must not be physically disabled.

- The mapping macro IEFJSSVT maps the SSVT. Only the master subsystem's SSVT matches the mapping exactly. JES2 and JES3 SSVTs have additional material appended to the end of the area mapped by IEFJSSVT. For JES3, the mapping macro is IATYSVT. For the contents of the JES2 SSVT, refer to *OS/VS2 JES2 Logic.*

- Some functions requested at the master subsystem cause the function to be broadcast to every active subsystem. These function codes are:

4 -	Notify the subsystem of end-of-task.
8 -	Notify the subsystem of end-of-address space.
9 -	Notify the subsystem of a WTO message.
10 -	Notify the subsystem of an operator command.
14 -	Notify the subsystem of a delete operator message (DOM).
32 -	Notify the subsystem of a failing START command.
50 -	Notify, early, the subsystem of end-of-task.
63 -	Notify the subsystem of service processor damage.

- If the WTO broadcast count (field UCMBRDST in the UCM) is greater than zero, function codes 9 and 14 use an SSOB with the SSIB pointer containing the address of the SSIB for the master subsystem. Otherwise, the pointer to the SSIB is set to zero, causing the SSIB pointer in the JSCB to be used. If the resulting SSIB is for the master subsystem, the request is given to every active subsystem. If the SSIB is not for the master subsystem, the request is given to only the subsystem named in the SSIB.

- If a subsystem verification request (function code 15) is made to the master subsystem, (field SSIBSSNM in the SSIB contains 'MSTR'), and the name in the SSIBJBID field of the SSIB is not that of a job entry subsystem, then upon return from the subsystem interface, field SSIBSSNM will contain the name of the primary job entry subsystem. A job entry subsystem is defined as a subsystem that can provide its own sysout services. This is indicated by bit SSCTUPSS being off in the subsystem's SSCVT.

Event Notification Facility (ENF)

The event notification facility (ENF) provides a service for MVS system routines that allows them to:

- Signal the occurrence of an event
- Listen for the occurrence of an event
- Delete the listen request

A system routine that has detected a condition or performed a function (the signaller) signals the event to ENF. Other system routines (the listeners) can request that they receive control (at an exit routine) from ENF when the event has occurred. A routine that is listening can also delete the request to listen.

The events that are signalled and can be listened for are:

Event Code	Description
1	A vary online of a device is being performed.
2	A vary offline of a device is being performed.
3	A removable volume is being unloaded.
4	Free SQA storage that is not currently in use.
5	The communications task and TOD clock initialization completed.
12	A device pending is offline.

The system components (and modules) that signal and listen for these events are:

Event Code	Signallers	Listeners
1	Command Processing (IEE3103D) Allocation (IEFAB488)	IOS (IECVDPTH)
2	Allocation (IEFAB429)	IOS (IECVDPTH)
3	Allocation (IEFAB494)	IXVTOC/CVAF
4	SRM (IRARMSRV)	Media Manager
5	IEEVIPL	Global resource serialization
12	Allocation (IEFAB429)	Global resource serialization

To help you diagnose ENF problems, the following topics describe how system routines request ENF services, the ENF control blocks, and ENF initialization and processing.

Requests for ENF Services

A system routine makes a request to ENF by building an event parameter list (ENFPM), calling the ENF request router (IEFENFFX), and passing the address of the ENFPM to IEFENFFX. The address of IEFENFFX is contained in field ENFCFMOD in the ENF control table (ENFCT).

In the ENFPM, the requesting routine specifies the type of request (signal, listen, or delete listen), the event code, synchronous or asynchronous processing, the address of the exit routine to receive control, and other information.

The format of the ENFPM is shown in Figure 5-43; a description of the fields follows the figure.

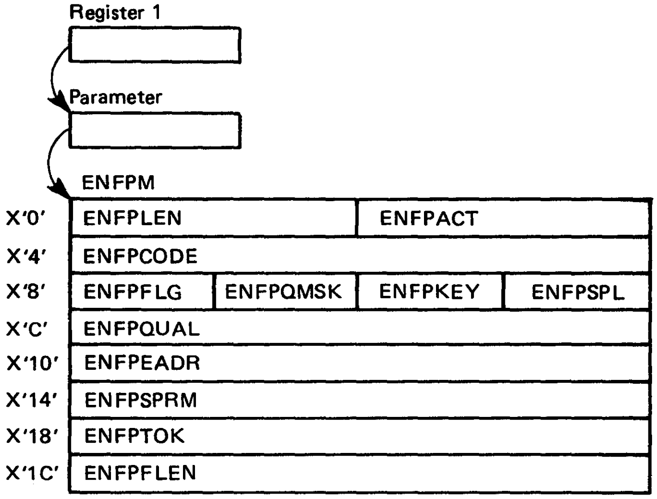


Figure 5-43. ENF Event Parameter List (ENFPM)

The fields in the event parameter list (ENFPM) are:

Offset	Length	Name	Description
0(0)	2	ENFPLEN	Length of the parameter list
2(2)	2	ENFPACT	Request: X'0001' - Signal request X'0002' - Listen request X'0003' - Delete listen request
4(4)	4	ENFPCODE	Event code: X'00000001' - A vary online of a device is being performed. X'00000002' - A vary offline of a device is being performed. X'00000003' - A removable volume is being unloaded X'00000004' - Free SQA storage that is not currently in use. X'00000005' - Communications task and TOD clock initialization completed. X'0000000C' - A device pending is offline.
8(8)	1	ENFPFLG	Flag field:
	1xxx xxxx	ENFPASN	Asynchronous processing
	0xxx xxxx		Synchronous processing
	xxxx 1xxx	ENFPFREE	Free signal parameter list
9(9)	1	ENFPQMSK	Qualifier mask field: For a listen request, specifies which bytes in the qualifier field (ENFPQUAL) are to be used during listen processing to filter the event: X'08' - First byte X'04' - Second byte X'02' - Third byte X'01' - Fourth byte X'00' - No filtering
10(A)	1	ENFPKEY	Key of signal parameter list to free.
11(B)	1	ENFPSPL	Subpool of signal parameter list to free.

12(C)	4	ENFPQUAL	Qualifier field: - For a signal request, specifies the qualifier bytes associated with the event. - For a listen request, specifies the qualifier bytes that must match with the signaller's qualifier bytes in order for the listener's exit routine to receive control. The listener specifies which qualifier bytes are to be matched via the ENFPQMSK field.
16(10)	4	ENFPEADR	Exit routine address: - For a listen request, specifies the address of the listener's exit routine that is to receive control when the event is signalled. - For a signal request, specifies the address of the signaller's exit routine that is to receive control when all listeners of an event have been notified and have completed processing.
20(14)	4	ENFPSPRM	For a signal request, specifies the address of a parameter list that the signaller passes to all listeners.
24(18)	4	ENFPTOK	Token value: - For a signal request, specifies the token value to be passed to the signaller's exit routine. - For a synchronous listen request, contains the token value returned by ENF for the event listener element (ENFLS). - For a delete listen request, specifies the token value of the event listener element (ENFLS) to be deleted.
28(1C)	4	ENFPFLEN	Length of signal parameter list to free.

Note: The system routines that use ENF services have agreed to pre-defined values for some fields in the ENFPM. These are the qualifier bytes used in the ENFPQUAL field and the content of the parameter list pointed to by the ENFPSPRM field. To determine these values, refer to the module listings of the signallers and listeners of events. The topic "ENF Logic Flow Examples" shows an example of how the qualifier bytes are used.

Listen and Signal Exit Routines

When a system routine makes a listen or signal request, the routine specifies the address of an exit routine to receive control in the event parameter list (field ENFPEADR). ENF mainline module IEFENFNM passes control to a listen exit routine when the specified event is signalled. When all listeners of the event have been notified and have completed processing, and if the signaller has specified the optional signal exit routine, IEFENFNM passes control to the signal exit routine.

ENF passes the following to a listen exit routine:

- Register 0 contains the event code.

- Register 1 contains the address of a fullword parameter, which contains the address of the parameter list passed by the signaller to the listeners of the event.

- Register 13 contains a save area address.

- Register 14 contains the return address.

ENF passes the following to a signal exit routine:

- Register 1 contains the address of a fullword parameter, which contains the address of the token value specified by the signaller on its signal request.
- Register 13 contains a save area address.
- Register 14 contains the return address.

Exit routines are entered enabled, in system key 0, supervisor state, and with no locks held.

ENF Control Blocks

The following control blocks are used during ENF processing. For the format of these control blocks, refer to the *Debugging Handbook,* Volume 2.

ENFCT ENF control table - contains ENF-related data and addresses of ENF routines and control blocks.

ENFVT ENF vector table - contains a pointer to the queue of listener elements (ENFLSs) for each event code.

ENFDS ENF process table - contains the event parameter lists (ENFPMs) to be processed for asynchronous requests.

ENFLS ENF listener element - contains the listener's exit routine address and optional qualifying (filtering) information for the listen request. For each listen request, ENF builds an ENFLS on the ENFLS queue of the specified event.

Figure 5-44 summarizes the ENF control blocks and Figure 5-45 shows the structure of the ENF control blocks.

Control Block	Created by	Subpool	Key	Size in Bytes	Pointed to by	Mapping Macro
ENFCT	SYSGEN	Nucleus	0	44	CVT	IEFENFCT
ENFVT	IEAVNPA7	231	0	Note 1	ENFCT	IEENFVT
ENFDS	IEAVNP47	239	0	1604	ENFCT	IEFENFDS
ENFLS	IEFENFNM	241	0	28	ENFVT ENFLS	IEFENFLS
Note 1. Four bytes plus eight bytes for each event code.						

Figure 5-44. ENF Control Block Summary

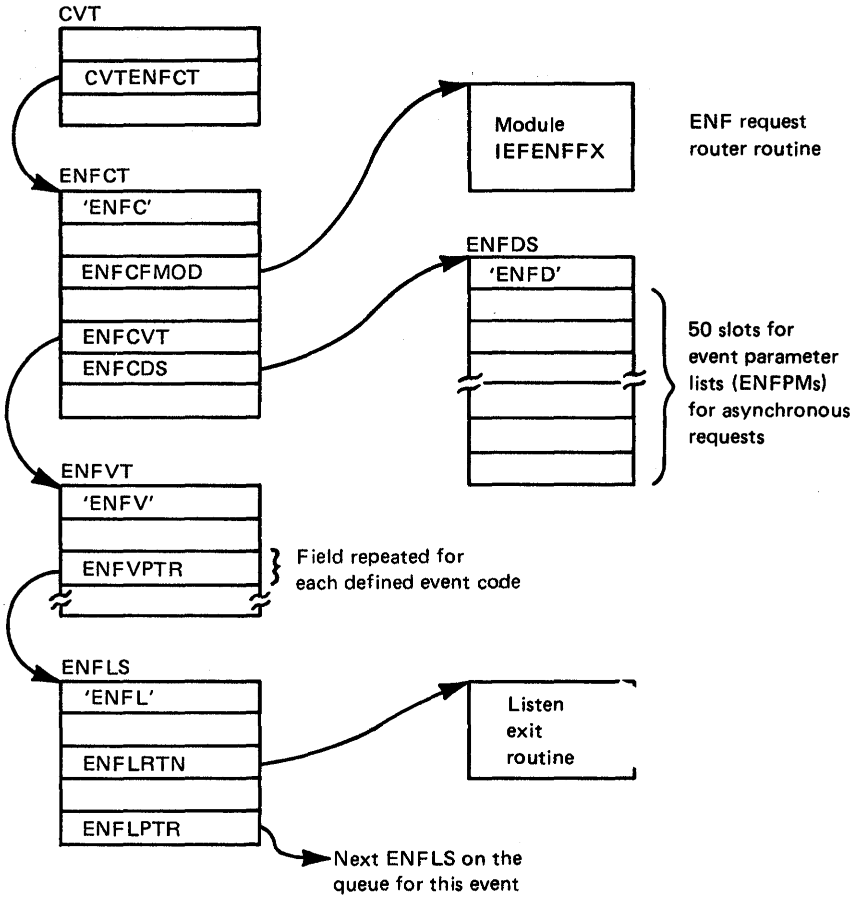

CVT
CVTENFCT
ENFCT
'ENFC'
ENFCFMOD
ENFCVT
ENFCDS
Module IEFENFFX
ENF request router routine
ENFDS
'ENFD'
50 slots for event parameter lists (ENFPMs) for asynchronous requests
ENFVT
'ENFV'
ENFVPTR
Field repeated for each defined event code
ENFLS
'ENFL'
ENFLRTN
ENFLPTR
Listen exit routine
Next ENFLS on the queue for this event

Figure 5-45. ENF Control Block Structure

ENF Initialization

Module IEAVNP47 initializes ENF during nucleus initialization (NIP) processing, which includes:

- Initializing the ENF control table (ENFCT)
- Creating the ENF vector table (ENFVT) for the event codes specified in the ENFCT.
- Creating the ENF process table (ENFDS), which is used for asynchronous requests.

Also, the master scheduler region initialization module (IEEMB860) attaches the ENF wait routine (IEFENFWT), which waits to process asynchronous requests.

ENF Processing

When a request is made for ENF services, module IEFENFFX (request router) pre-processes the request.

If synchronous processing is requested in the event parameter list (ENFPM), IEFENFFX calls the ENF mainline module (IEFENFNM) to process the request.

If asynchronous processing is requested in the ENFPM, IEFENFFX stores the ENFPM in the ENF process table (ENFDS) and posts the ENF wait routine (IEFENFWT). IEFENFWT runs in the master address space and calls IEFENFNM to process the request.

For a description of the program logic for the ENF modules, refer to *OS/VS2 System Logic Library*.

The following topics describe the return codes returned by ENF to the caller, and show examples of logic flow.

ENF Return Codes

ENF returns the following return codes in register 15 to its caller.

Return Code		
Dec	**(Hex)**	**Description**
0	(0)	The request was processed successfully.
4	(4)*	A duplicate listen request was detected.
8	(8)	The ENFDS control block is full (ENF cannot process the asynchronous request).
12	(C)	An error was detected in the caller's event parameter list (ENFPM).
16	(10)	ENF is not available.
20	(14)	ENF is not initialized.
24	(18)*	Storage cannot be obtained for the listen request.
28	(1C)*	An invalid token value was detected on a delete listen request.
32	(20)*	An error occurred in the signal exit routine for a signal request.
44	(2C)	The freemain for the signal parameter list failed.

*Returned for synchronous requests only. (For asynchronous requests, these conditions result in a zero return code because the conditions have not yet been checked by ENF.)

ENF Logic Flow Examples

Examples of the logic flow for a signal, listen, and delete listen request are described in the following topics.

Listen for an Event

A system routine (for example, IOS initialization) needs to listen for the "vary online" event, and wants to limit the listen request to tape devices.

1. The routine builds an event parameter list (ENFPM) and fills in the following fields:

ENFPLEN	X'001C' (28 bytes)
ENFPACT	X'0002' (listen request)

ENFPCODE	X‘00000001’ (vary online event)
ENFPFLG	X‘00’ (synchronous processing)
ENFPQMSK	X‘02’ (qualifier mask)
ENFPQUAL	X‘00008000’ (qualifier field)
ENFPEADR	address of the listen exit routine that is to receive control when the event (vary online) occurs
ENFPSPRM	zero
ENFPTOK	zero (ENF returns the ENFLS token value in this field)

2. The routine calls the ENF request router (IEFENFFX) with register 1 pointing to a fullword parameter which points to the event parameter list (ENFPM).

3. IEFENFFX checks the validity of the ENFPM and, if valid, calls the ENF mainline routine (IEFENFNM).

4. IEFENFNM adds a listener element (ENFLS) to the listen queue for the vary online event.

5. IEFENFNM returns to IEFENFFX, which returns to the caller and passes back the listen token value for the ENFLS in field ENFPTOK of the caller's ENFPM.

The listen exit routine (specified in field ENFPEADR) will receive control when a vary online event occurs and the third byte of the signaller's qualifier field matches the third byte of the listener's qualifier field (X‘80’). As shown in the next example, the signaller of the vary online event puts the UCBTYP field in its qualifier field (where X‘80’ indicates a tape device). Therefore, the listen exit routine is entered for a vary online of tape devices.

Signal an Event

A system routine (for example, vary device processing) is performing a vary online function and signals the event to ENF. In this example, the routine puts the UCBTYP field in the ENFPQUAL field. This allows listeners to filter their listen requests to specific device types or characteristics.

1. The routine builds an event parameter list (ENFPM) and fills in the following fields:

ENFPLEN	X‘001C’ (28 bytes)
ENFPACT	X‘0001’ (signal request)
ENFPCODE	X‘000000001’ (vary online event)
ENFPFLG	X‘00’ (synchronous processing)
ENFPQMSK	zero (not used)
ENFPQUAL	X‘02208001’ (qualifier field)

ENFPEADR optional (address of a signal exit routine to receive control when all listeners have been notified and have completed processing)

ENFPSPRM optional (address of a parameter list to be sent to all listeners)

ENFPTOK optional (the token value to be passed to the signal exit routine)

2. The routine calls the ENF request router (IEFENFFX) with register 1 pointing to a fullword parameter which points to the event parameter list (ENFPM).

3. IEFENFEX checks the validity of the ENFPM and, if valid, calls the ENF mainline routine (IEFENFNM).

   Note: For asynchronous processing, IEFENFFX stores the ENFPM in the ENFDS (process table) and posts the ENF wait routine (IEFENFWT), which calls IEFENFNM.

4. IEFENFNM processes the request as follows:

   a. Searches the listen queue (ENFLSs) of the vary online event and, if the listener had specified qualifier bytes, looks for a match of the signaller's qualifier bytes to the listener's qualifier bytes.

   b. Calls each listener's exit routine and passes to each exit the event code and, if specified, the address of the signaller's parameter list (ENFPSPRM).

   c. If a signal exit routine address was specified in field ENFPEADR of the signaller's ENFPM, passes control, with the token value, to the signaller's exit routine.

   d. Returns control to IEFENFFX.

5. IEFENFFX returns to the caller.

Delete a Listen Request

A system routine that has been listening for a specific event (such as vary online processing) no longer needs to know when the event occurs.

1. The routine builds an event parameter list (ENFPM) and fills in the following fields:

ENFPLEN	X‘001C’ (28 bytes)
ENFPACT	X‘0003’ (delete listen request)
ENFPCODE	X‘000000001’ (vary online event)
ENFPFLG	X‘00’ (synchronous processing)
ENFPQMSK	zero
ENFPQUAL	zero
ENFPEADR	zero
ENFPSPRM	zero
ENFPTOK	token value of the ENFLS that ENF returned on the listen request for the event.

2. The routine calls the ENF request to router (IEFENFFX) with register 1 pointing to a fullword parameter which points to the event parameter list (ENFPM).

3. IEFENFFX checks the validity of the ENFPM and, if valid, calls the ENF mainline routine (IEFENFNM).

4. IEFENFNM processes the delete listen request as follows:

   a. Searches the listen queue (ENFLS) for a matching token value for the event code specified.

   b. Deletes the appropriate listener element (ENFLS) by marking the ENFLS available for reuse. (See Note).

   c. Returns control to IEFENFFX.

5. IEFENFFX returns to the caller.

   Note: If the ENFLS is in use, indicated by a use count of one or more in field ENFLUSE, ENF sets the high-order bit (ENFLDEL) of field ENFLUSE to one to indicate that a delete request is pending. ENF does not call the listen exit routine for any additional signal requests to this ENFLS. After any existing listen exit routines that are in process are serviced, ENF marks the ENFLS available for reuse when the next signal request for the event is processed.

   The ENFLUSE field has the following meaning:

High-order Bit (ENFLDEL)	Remainder of Field	Meaning
1	0	The ENFLS has been deleted and is available for reuse.
1	>0	A delete request is pending for the ENFLS.
0	>0	The ENFLS is active and is in use.
0	0	The ENFLS is active but not in use.

ENF Recovery Routines

If an error occurs during ENF processing, ENF recovery routines issue the SETRP macro (with RECORD=YES) to record errors to the SYS1.LOGREC data set and issue the SDUMP macro to dump virtual storage.

The FRR and ESTAE recovery routines (FRRRTN and ESTAERTN) in modules IEFENFFX and IEFENFNM put the following data in the variable recording area (SDWAVRA) of the SDWA in a key-length format:

- Component ID (BB131)
- Product level
- Module footprints (FOOTPRNT)
- ESTAE or FRR parameter list

The ESTAE recovery routine (ESTAEXIT) in module IEFENFWT does not put data in the SDWAVRA.

Recovery Termination Manager (RTM)

The recovery termination manager (RTM) cleans up system resources when a task oı address space terminates, either normally or abnormally.

Functional Description

Logically, RTM consists of four related processes.

1. *RTM1* attempts recovery for software or hardware errors; it is entered via the CALLRTM macro instruction issued by supervisory routines. Functional recovery routines (FRRs) are processed in this logical phase.

2. *RTM2* performs normal and abnormal task termination for both system and problem program routines. The ABEND macro (SVC 13) requests RTM2 services.

3. Address space termination provides normal and abnormal address space termination for supervisory routines. The CALLRTM macro instruction is used to request this function.

4. RTM support functions such as error recording, formatting of dumps (see note), and creating recovery control blocks for error exit processing.

Note: RTM generates an error id that ensures that information recorded in SYS1.LOGREC concerning a problem, can be readily correlated with SVC dump information concerning the same problem. See 'Error Id' later in this topic.

Work Areas

For details of RTM work areas see "Use of Recovery Work Areas for Problem Solving" in Section 2.

Major RTM Modules

RTM1, which is part of the nucleus, comprises seven modules:

1. IEAVTRT1 - RTM entry point processor
2. IEAVTRTM - RTM1 mainline
3. IEAVTRTS - system recovery manager
4. IEAVTRTR - RTM1 recovery routines
5. IEAVTRS0 - RTM1 subroutines
6. IEAVTSR1 - ITERM processor
7. IEAVTRMC - RMGRCML preprocessor

RTM2, which resides in the link pack area (LPA), is entered via SVC 13. The mainline for RTM2 comprises the following three modules:

1. IEAVTRT2 - initialization
2. IEAVTRTC - controller
3. IEAVTRTE - exit handler

Other important RTM2 modules are:

- IEAVTAS1 - pre-exit processing
- IEAVTAS2 - post-exit processing
- IEAVTAS3 - control recovery
- IEAVTSKT - task termination purges
- IEAVTMRM - RTM2 resource manager
- IEAVTRML - installation resource manager list

Process Flow

The following charts depict the process flow for:

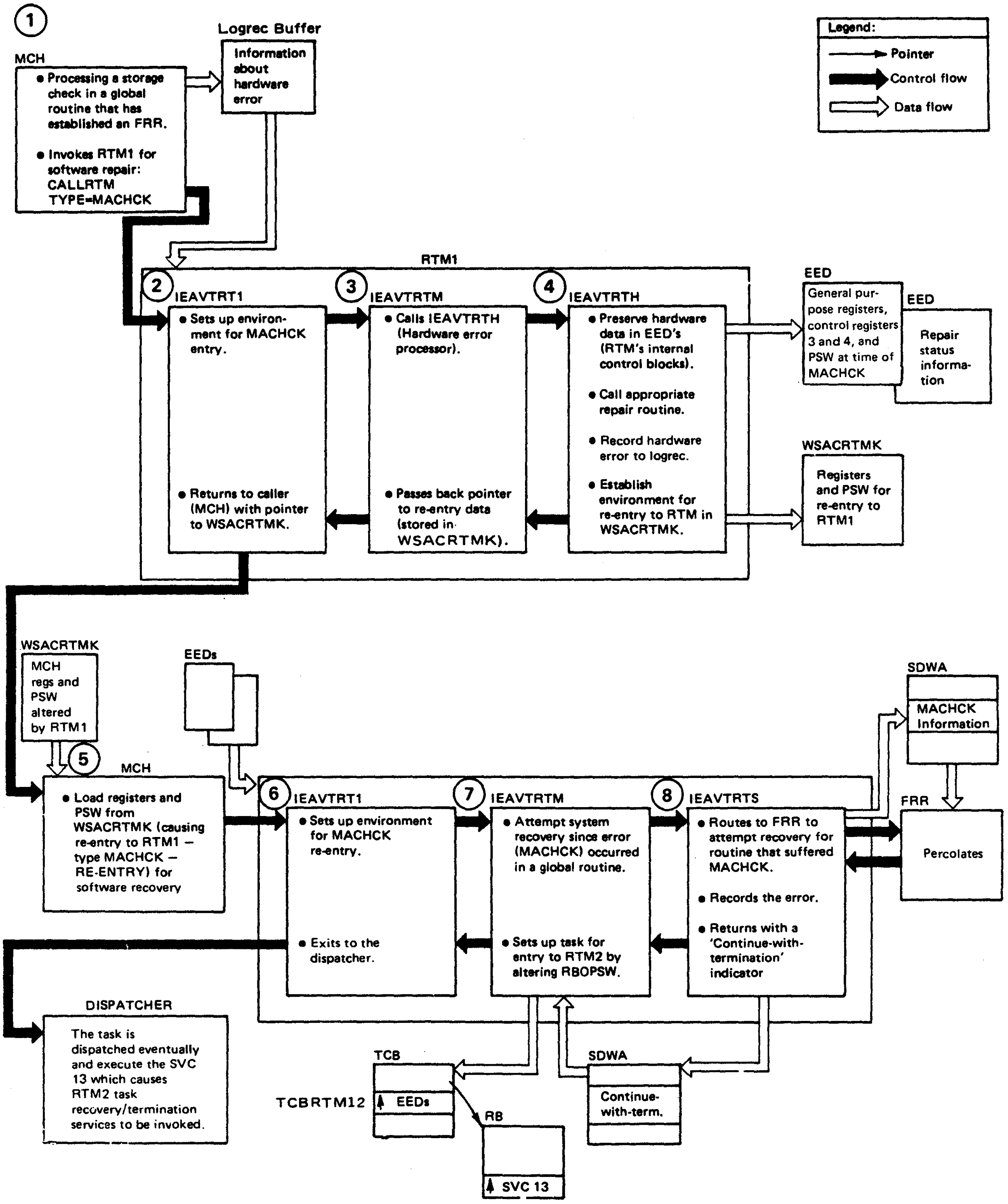
- Hardware error processing
- Normal end-of-task termination
- Abnormal end-of-task termination
- Retry
- Cancel
- Address-space termination
- PER activation/deactivation

Hardware Error Processing

Depicted in the following diagram is the processing for a hard type machine check in a global routine that has FRR recovery. It shows the interfaces and control flow between the machine check handler and RTM1 for both hardware error processing and the resulting software recovery attempt by the FRR. It indicates that software recovery continues in task mode because, in this example, the FRR does not recover the error.

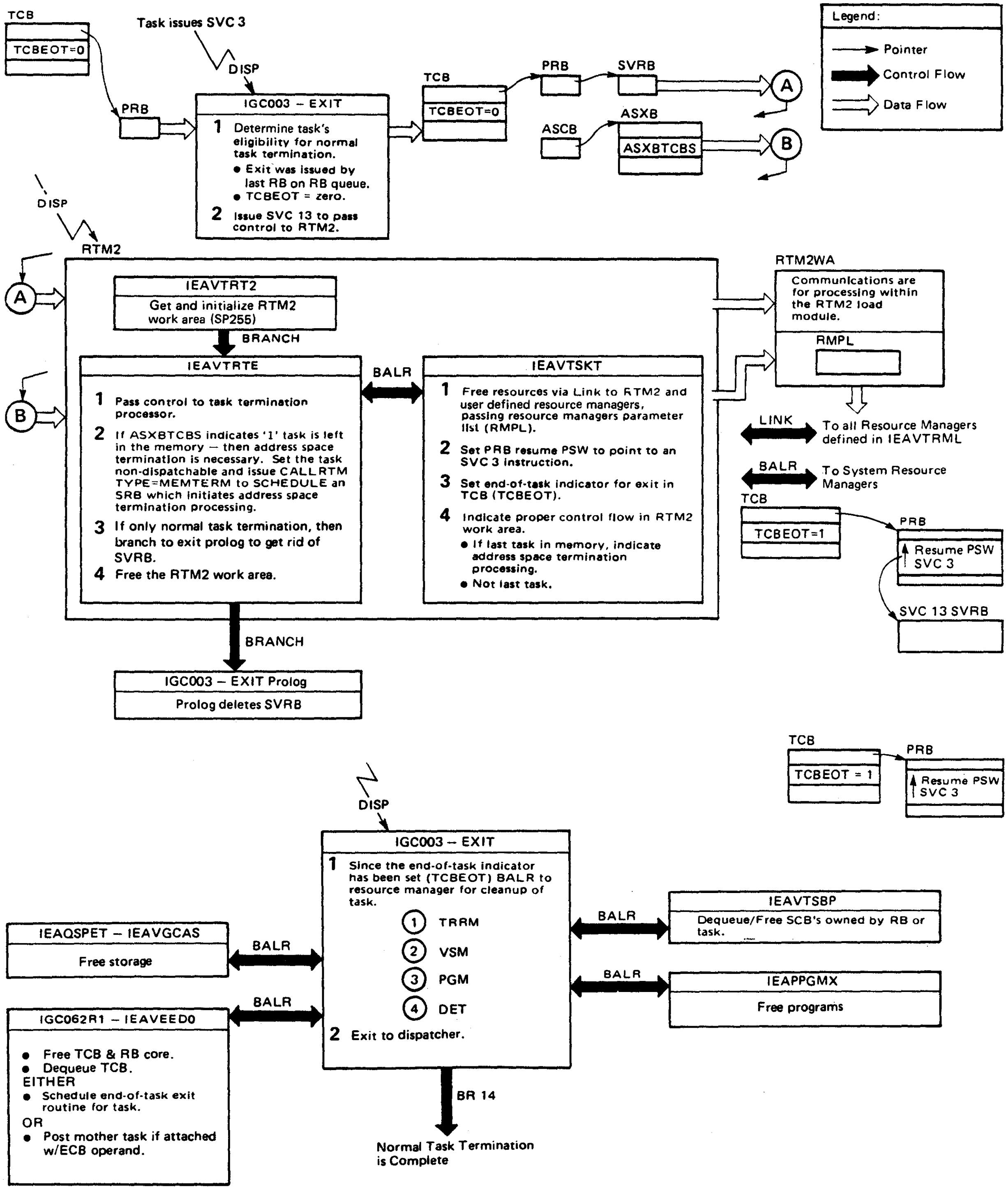
The use of extended error descriptors (EEDs) allows the LOGREC buffer to be available for further possible machine checks and is the mechanism for passing information to RTM1 and RTM2. The information in the global system diagnostic work area (SDWA) used by RTM1 recovery was obtained from the EEDs. RTM2 obtains an SDWA, but also uses the EEDs as its source of error data to be passed to recovery routines.

RTM1 uses the RTM processor-related work save area (WSACRTMK) to alter the registers and the PSW that MCH reloads, thereby determining whether MCH resumes the interrupted process (soft error), or reenters RTM1 for software recovery (or hard error).


1
MCH
• Processing a storage check in a global routine that has established an FRR.
• Invokes RTM1 for software repair: CALLRTM TYPE=MACHCK
Logrec Buffer
Information about hardware error
Legend:
Pointer
Control flow
Data flow
RTM1
2
IEAVTRT1
• Sets up environment for MACHCK entry.
• Returns to caller (MCH) with pointer to WSACRTMK.
3
IEAVTRTM
• Calls IEAVTRTH (Hardware error processor).
• Passes back pointer to re-entry data (stored in WSACRTMK).
4
IEAVTRTH
• Preserve hardware data in EED's (RTM's internal control blocks).
• Call appropriate repair routine.
• Record hardware error to logrec.
• Establish environment for re-entry to RTM in WSACRTMK.
EED
General purpose registers, control registers 3 and 4, and PSW at time of MACHCK
EED
Repair status information
WSACRTMK
Registers and PSW for re-entry to RTM1
WSACRTMK
MCH regs and PSW altered by RTM1
5
MCH
• Load registers and PSW from WSACRTMK (causing re-entry to RTM1 – type MACHCK – RE-ENTRY) for software recovery
EEDs
6
IEAVTRT1
• Sets up environment for MACHCK re-entry.
• Exits to the dispatcher.
7
IEAVTRTM
• Attempt system recovery since error (MACHCK) occurred in a global routine.
• Sets up task for entry to RTM2 by altering RBOPSW.
8
IEAVTRTS
• Routes to FRR to attempt recovery for routine that suffered MACHCK.
• Records the error.
• Returns with a 'Continue-with-termination' indicator
SDWA
MACHCK Information
FRR
Percolates
DISPATCHER
The task is dispatched eventually and execute the SVC 13 which causes RTM2 task recovery/termination services to be invoked.
TCB
TCBRTM12
EEDs
RB
SVC 13
SDWA
Continue-with-term.

Normal Task Termination

EXIT and parts of RTM2 make up this function as shown in the following diagram. The flow shows how EXIT is entered and then reentered to complete task termination; it also provides a perspective of RTM2 functions related to normal termination of a task.



IGC003 – EXIT

1 Determine task's eligibility for normal task termination.
- Exit was issued by last RB on RB queue.
- TCBEOT = zero.

2 Issue SVC 13 to pass control to RTM2.

IEAVTRT2

Get and initialize RTM2 work area (SP255)

IEAVTRTE

1 Pass control to task termination processor.

2 If ASXBTCBS indicates '1' task is left in the memory – then address space termination is necessary. Set the task non-dispatchable and issue CALLRTM TYPE=MEMTERM to SCHEDULE an SRB which initiates address space termination processing.

3 If only normal task termination, then branch to exit prolog to get rid of SVRB.

4 Free the RTM2 work area.

IEAVTSKT

1 Free resources via Link to RTM2 and user defined resource managers, passing resource managers parameter list (RMPL).

2 Set PRB resume PSW to point to an SVC 3 instruction.

3 Set end-of-task indicator for exit in TCB (TCBEOT).

4 Indicate proper control flow in RTM2 work area.
- If last task in memory, indicate address space termination processing.
- Not last task.

RTM2WA

Communications are for processing within the RTM2 load module.

LINK To all Resource Managers defined in IEAVTRML

BALR To System Resource Managers

IGC003 – EXIT Prolog

Prolog deletes SVRB

IGC003 – EXIT

1 Since the end-of-task indicator has been set (TCBEOT) BALR to resource manager for cleanup of task.

① TRRM

② VSM

③ PGM

④ DET

2 Exit to dispatcher.

IEAQSPET – IEAVGCAS

Free storage

IGC062R1 – IEAVEEDO

- Free TCB & RB core.
- Dequeue TCB.

EITHER
- Schedule end-of-task exit routine for task.

OR
- Post mother task if attached w/ECB operand.

IEAVTSBP

Dequeue/Free SCB's owned by RB or task.

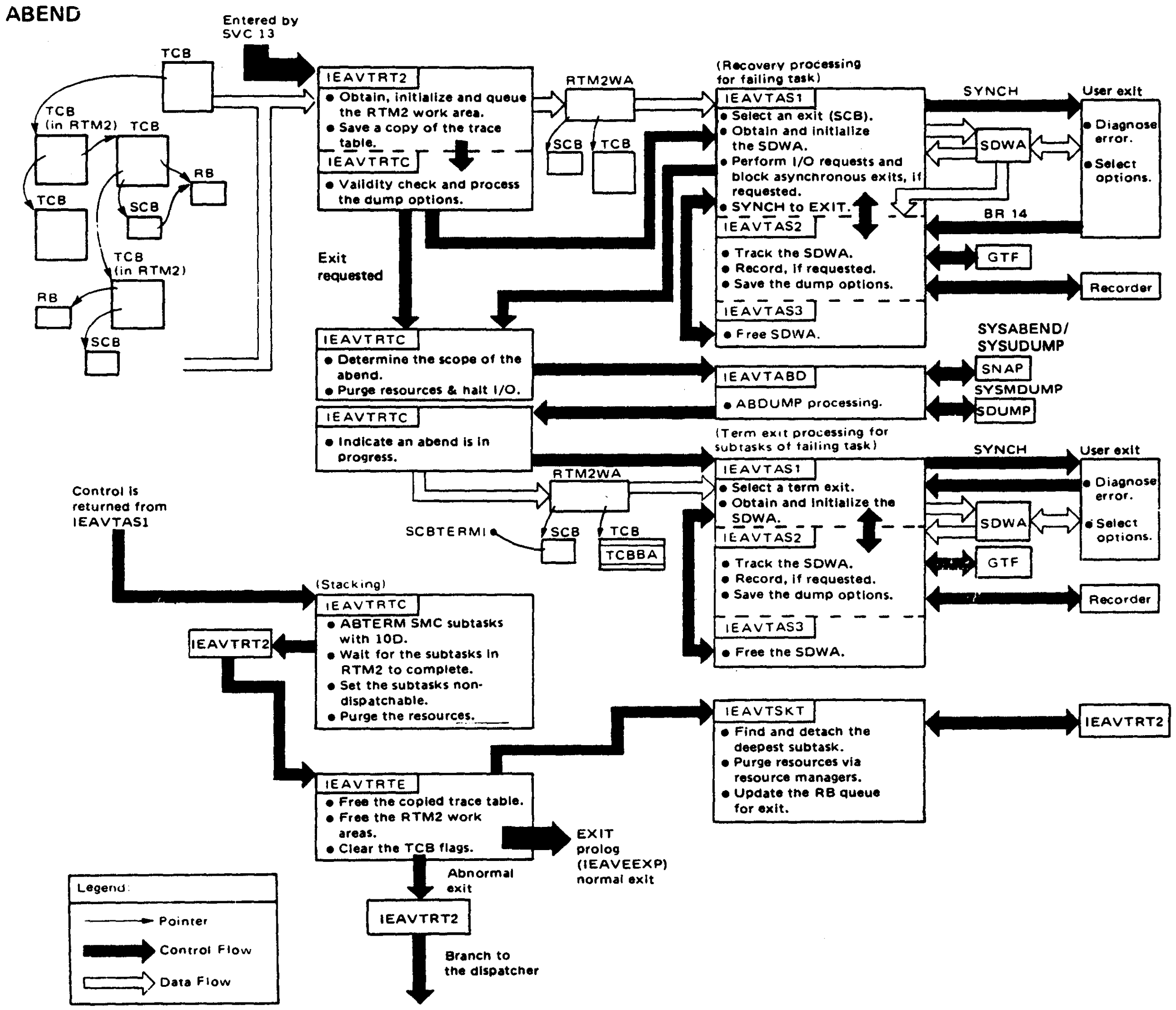
IEAPPGMX

Free programs

Normal Task Termination is Complete

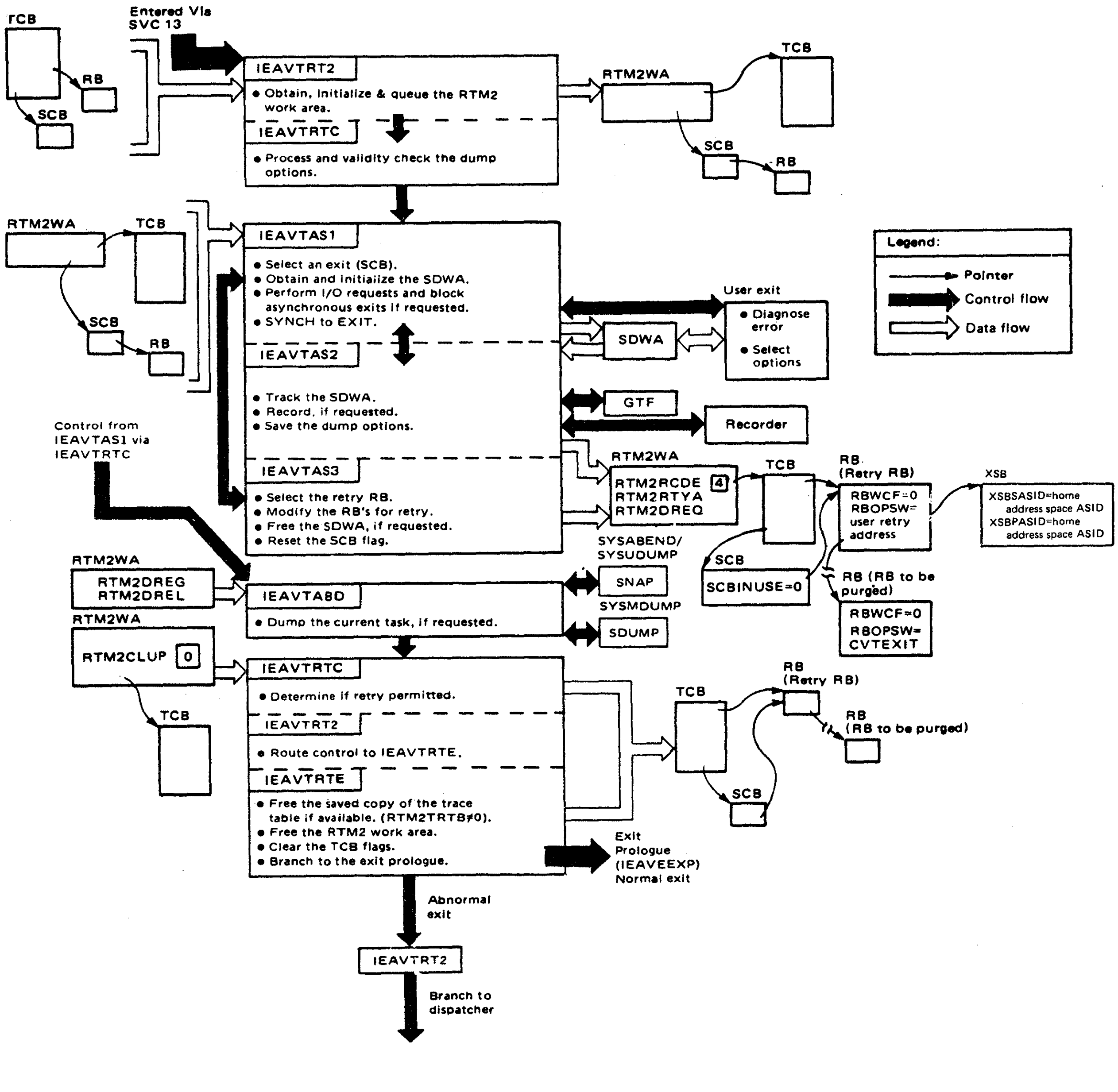
Abnormal Task Termination

Shown in the following diagram is the logic flow during abnormal termination of a non-critical nature. If the error is not recoverable at a particular task level, that task and its subtasks are removed. If the scope of the abend is "Step," then the entire job step is removed. Optionally, serviceability information (dumps and software error records) is supplied to the user.


ABEND
Entered by SVC 13
TCB
TCB (in RTM2)
TCB
RB
TCB
SCB
TCB (in RTM2)
RB
SCB
IEAVTRT2
• Obtain, initialize and queue the RTM2 work area.
• Save a copy of the trace table.
IEAVTRTC
• Validity check and process the dump options.
RTM2WA
SCB
TCB
(Recovery processing for failing task)
IEAVTAS1
• Select an exit (SCB).
• Obtain and initialize the SDWA.
• Perform I/O requests and block asynchronous exits, if requested.
• SYNCH to EXIT.
IEAVTAS2
• Track the SDWA.
• Record, if requested.
• Save the dump options.
IEAVTAS3
• Free SDWA.
SYNCH
User exit
SDWA
• Diagnose error.
• Select options.
BR 14
GTF
Recorder
Exit requested
IEAVTRTC
• Determine the scope of the abend.
• Purge resources &amp; halt I/O.
IEAVTABD
• ABDUMP processing.
SYSABEND/ SYSUDUMP
SNAP
SYSMDUMP
SDUMP
IEAVTRTC
• Indicate an abend is in progress.
(Term exit processing for subtasks of failing task)
RTM2WA
IEAVTAS1
• Select a term exit.
• Obtain and initialize the SDWA.
IEAVTAS2
• Track the SDWA.
• Record, if requested.
• Save the dump options.
IEAVTAS3
• Free the SDWA.
SYNCH
User exit
SDWA
• Diagnose error.
• Select options.
GTF
Recorder
Control is returned from IEAVTAS1
SCBTERMI
SCB
TCB
TCBBA
(Stacking)
IEAVTRTC
• ABTERM SMC subtasks with 10D.
• Wait for the subtasks in RTM2 to complete.
• Set the subtasks non-dispatchable.
• Purge the resources.
IEAVTRT2
IEAVTSKT
• Find and detach the deepest subtask.
• Purge resources via resource managers.
• Update the RB queue for exit.
IEAVTRT2
IEAVTRTE
• Free the copied trace table.
• Free the RTM2 work areas.
• Clear the TCB flags.
EXIT prolog (IEAVEEXP) normal exit
Abnormal exit
IEAVTRT2
Branch to the dispatcher
Legend:
Pointer
Control Flow
Data Flow

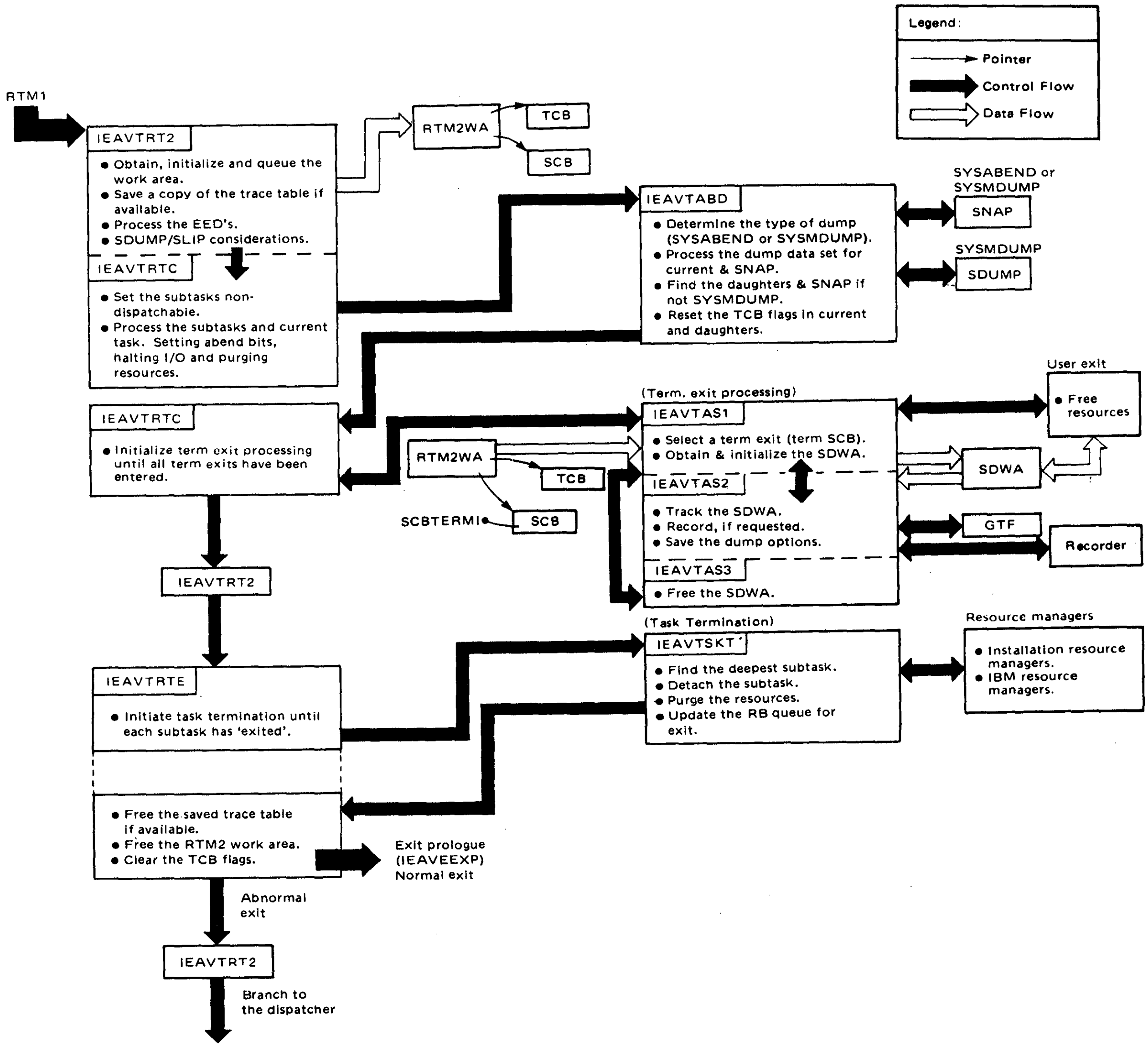
Retry

Shown in the following diagram is the flow through RTM2 when processing a potentially recoverable error. The recovery exit is supplied environmental data that describes the error (for example, completion code, register contents, PSW, system state at time of error) to aid in diagnosing the error. To retry, the resume PSW in each request block (RB) up to and including the retry RB is modified. The retry address supplied by the exit is placed in the resume PSW field of the retrying RB, and all RBs between the retry RB and the RTM2 RB have their resume PSW set to either exit prologue or SVC 3. To ensure that the retry routine runs in the home address space, the RBOPSW S-bit is zeroed and the ASIDs of the primary and secondary address spaces in the XSB of the retrying RB are set to the ASID of the home address space. When RTM2 eventually returns to the system, supervisor-assisted linkage will cause the retry address in the retry RB to be given control.



Cancel

Shown in the following diagram is the flow of control through RTM when a job is cancelled. The CANCEL request is indicated by specific completion codes set in the TCB by RTM1 (code='X22'). The CANCEL process is distinctive in that it is considered a strictly unrecoverable situation. Normal termination procedures are abandoned in favor of creating an express path through termination. However, term exits are given control.



Legend:
Pointer
Control Flow
Data Flow
RTM1
IEAVTRT2
• Obtain, initialize and queue the work area.
• Save a copy of the trace table if available.
• Process the EED's.
• SDUMP/SLIP considerations.
IEAVTRTC
• Set the subtasks non-dispatchable.
• Process the subtasks and current task. Setting abend bits, halting I/O and purging resources.
RTM2WA
TCB
SCB
IEAVTABD
• Determine the type of dump (SYSABEND or SYSMDUMP).
• Process the dump data set for current & SNAP.
• Find the daughters & SNAP if not SYSMDUMP.
• Reset the TCB flags in current and daughters.
SYSABEND or SYSMDUMP
SNAP
SYSMDUMP
SDUMP
IEAVTRTC
• Initialize term exit processing until all term exits have been entered.
RTM2WA
TCB
SCBTERMI
SCB
(Term. exit processing)
IEAVTAS1
• Select a term exit (term SCB).
• Obtain & initialize the SDWA.
IEAVTAS2
• Track the SDWA.
• Record, if requested.
• Save the dump options.
IEAVTAS3
• Free the SDWA.
User exit
• Free resources
SDWA
GTF
Recorder
IEAVTRT2
(Task Termination)
IEAVTSKT
• Find the deepest subtask.
• Detach the subtask.
• Purge the resources.
• Update the RB queue for exit.
Resource managers
• Installation resource managers.
• IBM resource managers.
IEAVTRTE
• Initiate task termination until each subtask has 'exited'.
• Free the saved trace table if available.
• Free the RTM2 work area.
• Clear the TCB flags.
Exit prologue (IEAVEEXP) Normal exit
Abnormal exit
IEAVTRT2
Branch to the dispatcher

FORCE Command

The FORCE command is designed to remove a job or TSO user from the system *after* the CANCEL command has failed to do so. For example, a job is writing to a DASD unit when the unit is suddenly made unavailable to the system; in this case, the CANCEL command is frequently unable to remove the job. If CANCEL does fail to remove the job, then FORCE can be used. However, FORCE does not use normal termination or normal cleanup routines, and is intended to be used only as an alternative to another IPL.

When FORCE is issued, the job's address space is terminated and any task running in the address space is terminated. If a job is running under an initiator, the initiator is also terminated.

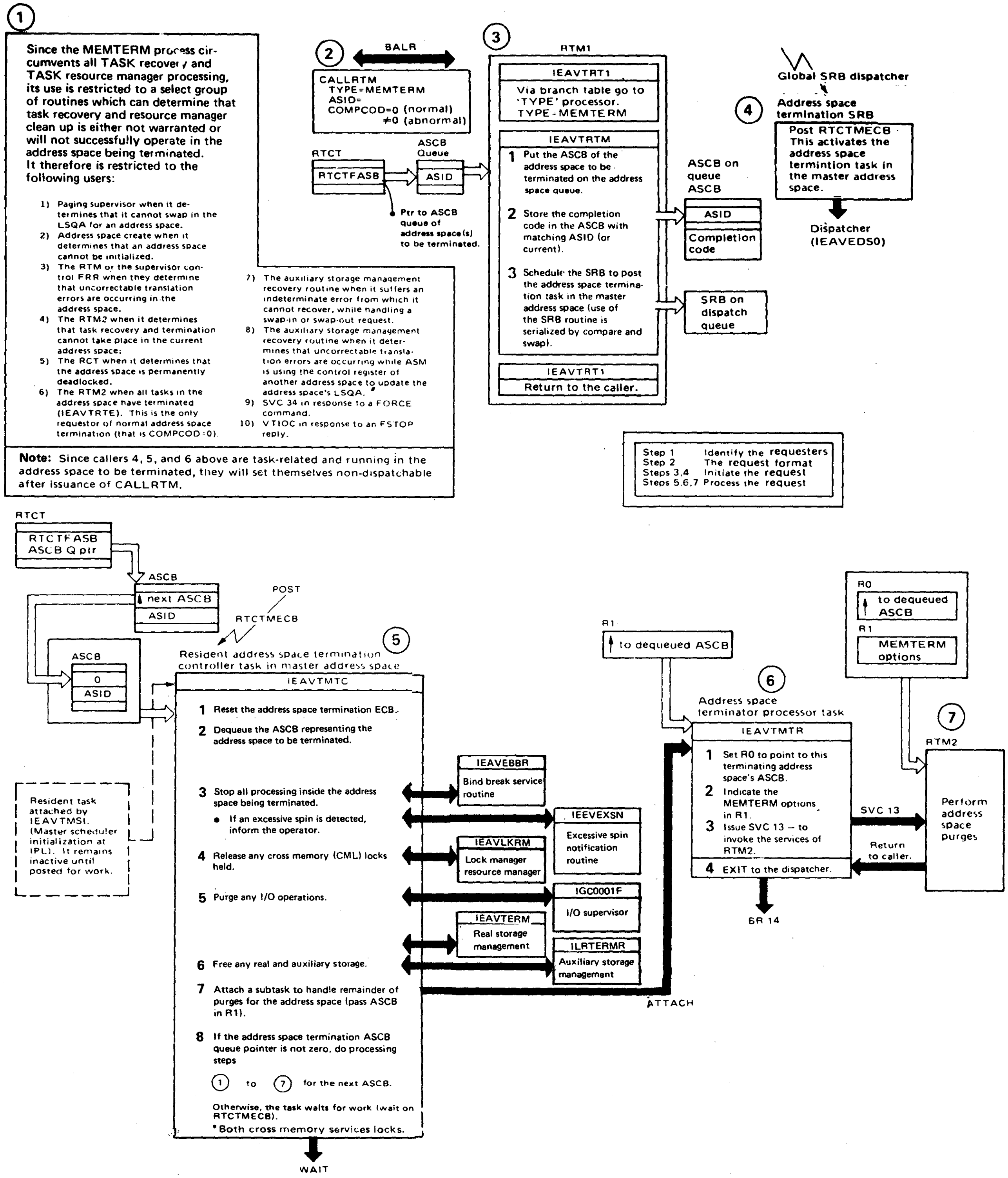
FORCE processing is dependent on the recovery termination manager (RTM), and on the command scheduling control block (CSCB), which contains a new bit definition in CHAFORCE. When the FORCE command is entered on a console having system authority, control is given to the CANCEL-FORCE processor which verifies that the command syntax is correct. The processor then scans the CSCB chain to see if the job exists and is cancelable. A bit in the job's CSCB is then checked to see if a CANCEL has been issued for this job. If not, a call is made to the message module to issue message IEE838I - 'CANCELABLE - ISSUE CANCEL BEFORE FORCE', and control is returned to the system. If CANCEL has been issued, a CALLRTM TYPE=MEMTERM is issued. The message module is called to issue message IEE301I - 'FORCE COMMAND ACCEPTED', and control is then returned to the system. If an error is found in the command syntax, or the job was either not found or was non-cancelable, the message module issues an appropriate message and control is returned to the system.

The FORCE processor uses the current CANCEL serialization code. The CSCB chain resource is serialized via ENQUEUE on SYSIEFSD Q10. Because the holder of CSCB Q10 must be non-swappable while holding the resource, the FORCE command processor issues a SYSEVENT DONTSWAP before issuing the ENQUEUE on Q10.

For additional information concerning the use of FORCE, see *Operator's Library: System Commands.*

Address-Space Termination

The process of terminating an address space (memory) is one that cannot be isolated to one task, module, or logical unit of code. The many parts of the process, depicting control flow and data flow, are shown in the following diagram.



1

Since the MEMTERM process circumvents all TASK recovery and TASK resource manager processing, its use is restricted to a select group of routines which can determine that task recovery and resource manager clean up is either not warranted or will not successfully operate in the address space being terminated. It therefore is restricted to the following users:

1) Paging supervisor when it determines that it cannot swap in the LSQA for an address space.
2) Address space create when it determines that an address space cannot be initialized.
3) The RTM or the supervisor control FRR when they determine that uncorrectable translation errors are occurring in the address space.
4) The RTM2 when it determines that task recovery and termination cannot take place in the current address space;
5) The RCT when it determines that the address space is permanently deadlocked.
6) The RTM2 when all tasks in the address space have terminated (IEAVTRTE). This is the only requestor of normal address space termination (that is COMPCOD=0).
7) The auxiliary storage management recovery routine when it suffers an indeterminate error from which it cannot recover, while handling a swap-in or swap-out request.
8) The auxiliary storage management recovery routine when it determines that uncorrectable translation errors are occurring while ASM is using the control register of another address space to update the address space's LSQA. *
9) SVC 34 in response to a FORCE command.
10) VTIOC in response to an FSTOP reply.

Note: Since callers 4, 5, and 6 above are task-related and running in the address space to be terminated, they will set themselves non-dispatchable after issuance of CALLRTM.

2

```
CALLRTM
  TYPE=MEMTERM
  ASID=
  COMPCOD=0 (normal)
         ≠0 (abnormal)
```

3 RTM1

IEAVTRT1: Via branch table go to 'TYPE' processor. TYPE=MEMTERM

IEAVTRTM

1 Put the ASCB of the address space to be terminated on the address space queue.

2 Store the completion code in the ASCB with matching ASID (or current).

3 Schedule the SRB to post the address space termination task in the master address space (use of the SRB routine is serialized by compare and swap).

IEAVTRT1: Return to the caller.

4 Address space termination SRB

Post RTCTMECB · This activates the address space termintion task in the master address space.

Step 1 Identify the requesters
Step 2 The request format
Steps 3,4 Initiate the request
Steps 5,6,7 Process the request

Resident task attached by IEAVTMSI. (Master scheduler initialization at IPL). It remains inactive until posted for work.

5 Resident address space termination controller task in master address space

IEAVTMTC

1 Reset the address space termination ECB.

2 Dequeue the ASCB representing the address space to be terminated.

3 Stop all processing inside the address space being terminated.
- If an excessive spin is detected, inform the operator.

4 Release any cross memory (CML) locks held.

5 Purge any I/O operations.

6 Free any real and auxiliary storage.

7 Attach a subtask to handle remainder of purges for the address space (pass ASCB in R1).

8 If the address space termination ASCB queue pointer is not zero, do processing steps 1 to 7 for the next ASCB.

Otherwise, the task waits for work (wait on RTCTMECB).

*Both cross memory services locks.

6 Address space terminator processor task

IEAVTMTR

1 Set R0 to point to this terminating address space's ASCB.

2 Indicate the MEMTERM options in R1.

3 Issue SVC 13 – to invoke the services of RTM2.

4 EXIT to the dispatcher.

7 RTM2

Perform address space purges

PER Activation/Deactivation

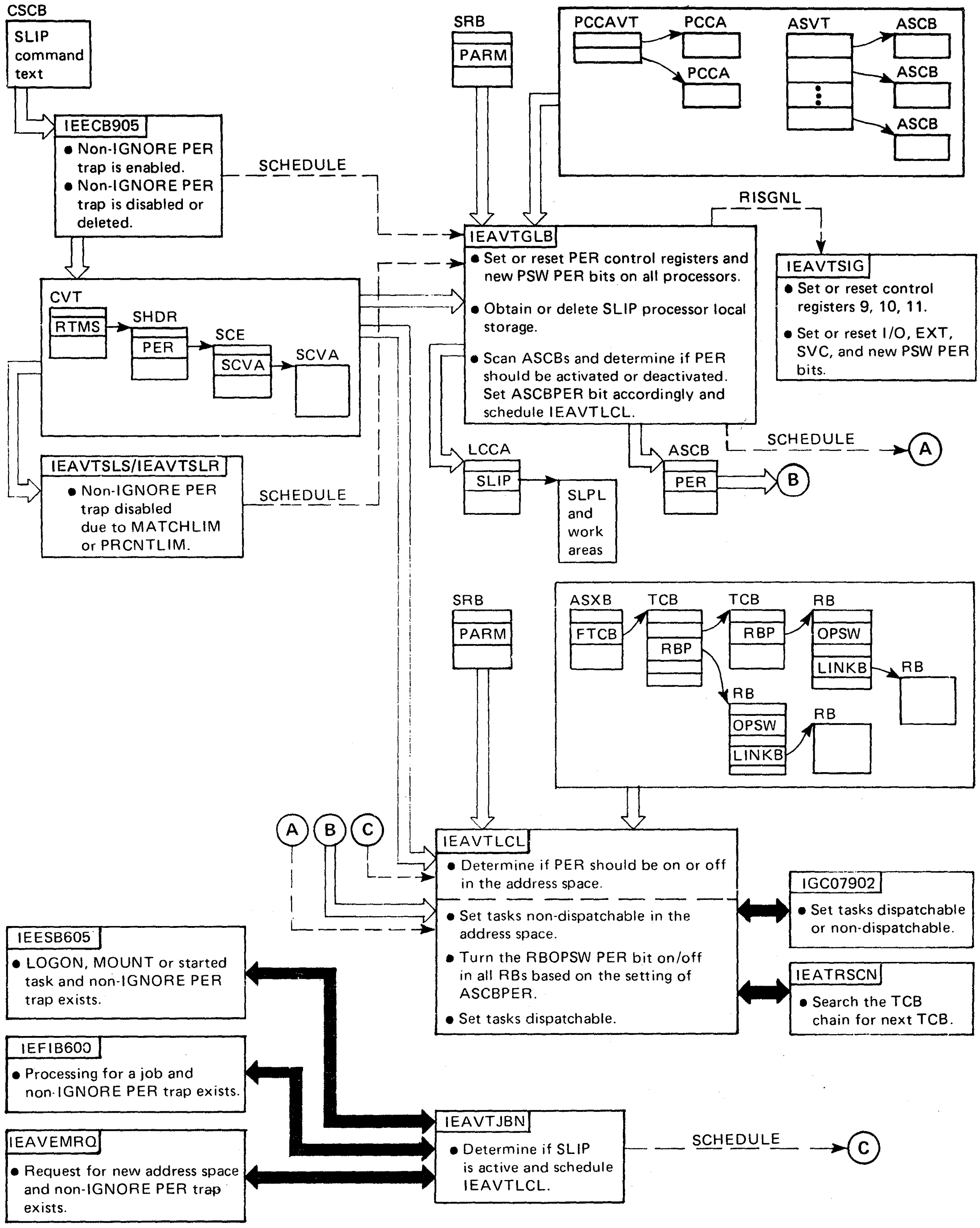
The following diagram shows the normal module flow that results when PER (program event recording) is activated or deactivated on behalf of a SLIP PER trap.

Activation can occur as a result of a SLIP command being entered which enables or sets a non-IGNORE SLIP PER trap. Deactivation can occur as a result of a SLIP command being entered which disables or deletes a non-IGNORE SLIP PER trap. Deactivation can also occur as a result of a non-IGNORE PER trap being automatically disabled by the SLIP processor (IEAVTSLS) or its recovery routine (IEAVTSLR). Whether activating or deactivating PER, the SLIP PER global activation/deactivation routine (IEAVTGLB) is scheduled as an SRB with global priority to run in the master address space.

IEAVTGLB manages the global resources associated with PER monitoring such as the control registers on all processors, processor local work and save areas, I/O, EXT, and SVC new PSW PER bits, and the ASCBPER bit in each address space.

The status (enabled or disabled) of the non-IGNORE PER trap will determine whether PER should be activated or deactivated. If activating or deactivating PER in an address space, IEAVTGLB schedules IEAVTLCL to run in that address space with local priority.

IEAVTLCL manages the address space local resources that affect PER monitoring. The PSW PER bits in saved PSWs are set to reflect that status of PER monitoring in the address space as determined by the ASCBPER bit. IEAVTLCL may also be scheduled by IEAVTJBN while PER is active in the system. This happens when a new address space is created or a change of jobs occurs in an address space. When scheduled by IEAVTJBN, IEAVTLCL determines whether or not PER should be active in the address space and sets the ASCBPER bit accordingly in addition to setting the PSW PER bit in saved PSWs.


CSCB
SLIP command text
IEECB905
● Non-IGNORE PER trap is enabled.
● Non-IGNORE PER trap is disabled or deleted.
SCHEDULE
SRB
PARM
PCCAVT
PCCA
PCCA
ASVT
ASCB
ASCB
ASCB
RISGNL
IEAVTGLB
● Set or reset PER control registers and new PSW PER bits on all processors.
● Obtain or delete SLIP processor local storage.
● Scan ASCBs and determine if PER should be activated or deactivated. Set ASCBPER bit accordingly and schedule IEAVTLCL.
IEAVTSIG
● Set or reset control registers 9, 10, 11.
● Set or reset I/O, EXT, SVC, and new PSW PER bits.
CVT
RTMS
SHDR
PER
SCE
SCVA
SCVA
IEAVTSLS/IEAVTSLR
● Non-IGNORE PER trap disabled due to MATCHLIM or PRCNTLIM.
SCHEDULE
LCCA
SLIP
SLPL and work areas
ASCB
PER
SCHEDULE
A
B
SRB
PARM
ASXB
FTCB
TCB
RBP
TCB
RBP
RB
OPSW
LINKB
RB
RB
OPSW
LINKB
RB
A
B
C
IEAVTLCL
● Determine if PER should be on or off in the address space.
● Set tasks non-dispatchable in the address space.
● Turn the RBOPSW PER bit on/off in all RBs based on the setting of ASCBPER.
● Set tasks dispatchable.
IGC07902
● Set tasks dispatchable or non-dispatchable.
IEATRSCN
● Search the TCB chain for next TCB.
IEESB605
● LOGON, MOUNT or started task and non-IGNORE PER trap exists.
IEFIB600
● Processing for a job and non-IGNORE PER trap exists.
IEAVEMRQ
● Request for new address space and non-IGNORE PER trap exists.
IEAVTJBN
● Determine if SLIP is active and schedule IEAVTLCL.
SCHEDULE
C

Error ID

Error ID ensures that problem information recorded in SYS1.LOGREC, can be easily correlated with message IEA911A and SVC dump information concerning the same problem. The error id function is invoked whenever the RTM is entered to process an error condition. The RTM determines if the entry is to process a recursive or a new error, a new error being one unrelated to a previous error.

If an error occurs during the processing of a previous error, the error id has the same sequence number as the original error, but is given a new time stamp. In this way, the sequence numbers show that the errors are related, and the time stamps show the history of error processing.

The RTM generates an entirely new error id:

1. Upon entry to RTM1 for a machine-check error (module IEAVTRTH).
2. Upon entry to RTM1 in SLIH mode for non-recursive error processing.
3. Upon entry to RTM2 when there has been no previous error processing in RTM1. Control passed to RTM2 by RTM1 does not result in a new error id if RTM1 has already generated one.

RTM1 maintains the error id in either the SDWA or an EED. RTM2 maintains the error id in its work area, RTM2WA. At an appropriate point in the error processing, the error id is moved to the SDUMP work area (pointed to by RTCTSDWK), where it is stored until processed by SDUMP. The correct error id is passed to SYS1.LOGREC when a software or hard machine check error record is written by RTM. Soft machine check error records do not contain an error id because no subsequent software recovery takes place following a "soft" error. IFCEREP1 recognizes and prints the error id in the LOGREC software or machine check record. AMDPRDMP recognizes and prints the error id as part of the header information for an SVC dump, formatted as follows:

```
ERRORID FOR THIS DUMP = SEQyyyyy CPUzz ASIDaaaa
TIMEhh.mm.ss.t
```

where:

yyyyy represents the sequence number of the error id
zz represents the error id logical processor
aaaa represents the error id ASID
hh.mm.ss.t represents the error id time stamp converted to read as hours, minutes, seconds, and tenths of seconds

If an error id is not available for a dump (indicated to AMDPRDMP by zeroes in the error id header field), the message "NO ERRORID ASSOCIATED WITH THIS DUMP" is printed where the error id is normally found.

To further increase the usefulness of error id, message IEA911A is changed to include the error id when an SVC dump is taken. The message reads:

```
IEA911A COMPLETE/PARTIAL DUMP
ON SYS1.DUMPxx/UNIT=ddd
ERRORID=SEQyyyy CPUzz ASIDaaaa TIMEhh.mm.ss.t
```

where xx and ddd have the same meaning as in the current message.

SVC Dump Debugging Aids

The SVC dump function of RTM is invoked when the SDUMP macro is issued. SVC dump produces dumps of system errors on a SYS1.DUMPxx or user-defined data set. SVC dump also produces abend dumps requested by SYSMDUMP DD statements.

Items that are important for you to understand when debugging errors in SVC dump processing are described in the following topics:

- Important SVC Dump Entry Points
- SVC Dump Error Conditions
- SYS1.LOGREC Entries Produced for SVC Dump Errors
- Control Blocks Used to Debug SVC Dump Errors
- Resource Cleanup for SVC Dump

Important SVC Dump Entry Points

The BRANCH= parameter on the SDUMP macro determines the SVC dump entry points and mainline processing to be used.

BRANCH=YES Option

Entry point IEAVTSDX is used for branch-entry SVC dumps. IEAVTSDX creates a summary dump in a real storage buffer (if the SUMDUMP option is requested on the SDUMP macro), schedules one or more SRBs to invoke dump task (IEAVTSDT) processing for the requested address spaces, and then returns control to the caller.

The branch-entry option is requested by many FRR routines and some ESTAE routines. This option is also requested when ACTION=SVCD is specified on the SLIP command.

BRANCH=NO Option

Entry point IEAVAD00 is used for the SVC entry to SVC dump. For scheduled dump requests (ASID, ASIDLST, LISTA, TYPE=XMEM, or TYPE=XMEME is specified on the SDUMP macro), IEAVAD00 calls IEAVTSDX which schedules one or more SRBs to invoke dump task (IEAVTSDT) processing for the requested address spaces, and then returns control to the caller. For synchronous dump requests (ASID, ASIDLST, LISTA, TYPE=XMEM, and TYPE=XMEME are not specified on the SDUMP macro), IEAVAD00 processes the dump and then returns to the caller.

The SVC entry is requested by many ESTAE routines. It is also requested by the DUMP command (as a scheduled dump), and by the abend dump processor (IEAVTABD) for SYSMDUMP DD statements (as a synchronous dump).

Determining the SVC Entry Point

The title line produced for SVC dumps that are formatted by the print dump service aid indicates whether a scheduled or synchronous SVC dump was requested, as follows:

- If "MODULE IEAVTSDT" is in the title line, then the dump was scheduled (SDUMP BRANCH=YES, or SDUMP BRANCH=NO with the ASID, ASIDLST, LISTA, TYPE=XMEM, or TYPE=XMEME keyword).
- If "MODULE SVCDUMP" is in the title line, then the dump was synchronous (SDUMP BRANCH=NO without the ASID or ASIDLST keyword).
- If another module name is in the title line, then the dump was synchronous. SVC dump was able to determine which load module invoked it and insert the name in the title.

SVC Dump Error Conditions

If the SVC dump function encounters an unexpected abend during its processing, it produces a software SYS1.LOGREC record and, if possible, continues taking the dump.

Expected program checks can occur when SVC dump is checking whether a virtual page that is to be dumped is valid and assigned. These program checks do not result in SYS1.LOGREC entries.

SVC dump issues abends 133 and 233 if it detects an unauthorized caller or invalid input parameter. In these cases, LOGREC entries are not created and retry is not attempted.

SVC dump issues a C0D abend for some unexpected errors during its processing. In this case, retry is attempted.

SYS1.LOGREC Entries Produced for SVC Dump Errors

The best starting place for debugging SVC dump problems is the SYS1.LOGREC entries contained in the in-storage SYS1.LOGREC buffer or in the SYS1.LOGREC data set, because a dump of the SVC dump problem is generally not available. (SVC dump does not take a dump of its own problems.)

Many SVC dump problems can be debugged from the SYS1.LOGREC entries alone. However, more complex problems may require a stand-alone dump that can be taken after a SLIP trap with ACTION=WAIT has matched. These problems include loops and failures to free critical system resources

Fixed Data

The fixed data that SVC dump places in the system diagnostic work area (SDWA) for recording on SYS1.LOGREC is:

SDWAMODN - Load module name (generally IGC0005A, which is the SVC 51 load module in SYS1.LPALIB).

SDWACSCT - CSECT (microfiche) name, which can be any SVC dump module name. For details of SVC dump module functions, interfaces, and flow, refer to *OS/VS2 System Logic Library*.

SDWAREXN - Recovery routine CSECT name.

SDWARRL - Recovery routine name, which is given as a label. This label is not always within the failing CSECT shown in SDWACSCT.

The following table shows the label of the recovery routine, the microfiche name of the containing CSECT, and a description of the recovery processing.

Label	CSECT	Description
DISRBFRR	IEAVTSDX	FRR routine for the SRB that creates the timer disabled interrupt exit (DIE) used to free the real storage buffer that is used with summary dumps for for branch-entry SVC dumps. This FRR is established by the SCHDISRB routine in IEAVTSDX.
DTESTAE1	IEAVTSDT	ESTAE routine for scheduled SVC dumps that are executing under the dump task (IEAVTSDT) in the requested address space. IEAVTSDT also establishes SDESTAEX which can percolate to DTESTAE1.
ESTAEXIT	IEEMB879	ESTAE routine for the master trace exit routine invoked by SVC dump module IEAVTSDU. This ESTAE is established by IEEMB879.
IEAVTSDR	IEAVTSDR	FRR for SDUMP SRBs. This FRR starts the dump task in the requested address space.
SCHFRR	IEAVTSDX	FRR routine for branch-entry to SVC dump, for part of scheduled SVC dump initialization, and for the timer disabled interrupt exit (DIE) used to free SVC dump's real storage buffer. This FRR is established by IEAVTSDX.
SDESTAEX	IEAVAD00	ESTAE routine for mainline SVC dump processing. This ESTAE is established by IEAVAD00 and IEAVTSDT.
SDFRRRTN	IEAVAD00	FRR routine for mainline SVC dump processing. This FRR is established by SVC dump modules when a lock is held and a retry is needed in the locked state. IEAVAD00 and IEAVTSDT are the main users of this FRR.
SRBFRR	IEEMB880	FRR routine for IEEMB880 which is the SRB routine that does data moves for IEEMB879. IEEMB879 is is the master trace exit for SVC dump. This FRR is established by IEEMB880.
SUMFRFRR	IEAVTSSD	FRR routine for SUMFRR routine processing. This FRR is established by the SUMFRR routine.
SUMFRR	IEAVTSSD	FRR routine for summary dump processing invoked for branch-entered SVC dumps. This FRR is is established by IEAVTSSD. If SUMFRR abends, SUMFRFRR receives control; and if it percolates, SCHFRR receives control.

Variable Data

The variable data that SVC dump places in the SDWAVRA field of the SDWA for recording to SYS1.LOGREC is:

- The 24-byte recovery routine parameter area if DTESTAE1, SDESTAEX, SDFRRRTN, SUMFRFRR, or SUMFRR is the recovery routine name in SDWARRL. This area contains bits that indicate the resources held, other status bits, the retry address, the base register value, and the address of the

SVC dump work area (ERRWKADR at X'8'). The contents of the parameter area are mapped by the IHASDERR macro. The common name of the work area is ERRWORK.

To obtain the offset into the failing module, subtract the base register field in ERRWORK (ERBASE1 at X'C') from the address in the failing PSW (found in the SDWANXT1 field at X'6C').

The ERRWORK data is mapped as follows:

Offset	Length	Name	Description
0(0)	1	ERRFLGS1	Flags:
		1...	FRR protection is active.
		.1..	ESTAE protection is active.
		..1.	SVC dump lock is set on (high-order bit in RTCTSDPL).
		...1	SVC dump 4K SQA buffer lock is set on (high-order bit in CVTSDBF).
		.... 1...	TCBs in the address space are set non-dispatchable.
		.... .1..	HOOK macro was issued for GTF.
		.... ..1.	Local lock is held.
		1	SALLOC lock is held.
1(1)	1	ERRFLGS2	Flags:
		1...	Error occurred during I/O - no retry.
		.1..	SVC dump has had a terminating error.
		..1.	IEAVAD00 is fixed in real storage.
		...1	The SVC dump work area is fixed in real storage.
		.... 1...	CMS lock is held.
		.... .1..	SVC dump has successfully initialized.
		.... ..1.	LOGREC recording is not needed (expect program checks during storage validation).
		1	At least one record has been successfully written to the dump data set.
2(2)	2	ERRUBSW	Register prime flags used for SETRP retry.
4(4)	4	ERRADDR	Retry address to be used by the error recovery routine. After a retry, the retry routine changes this to the address of ERRWORK.
4(4)	4	ERRRETRY	Another name for ERRADDR.
8(8)	4	ERRWKADR	Address of SVC dump work area (reg 7).
12(C)	4	ERRBASE1	First base register (reg 9).
16(10)	1	ERRFLAG3	Flags:
		1...	A term ESTAE exit was not established. Do not risk retry.
		.1..	Switch used to end DO loop processing.
		..1.	ENQ performed on the SVC dump resource (doing I/O to a dump data set).
		...1	IEAVTSDT called this module.
		.... 1...	All dump tasks scheduled to other address spaces should DEQ and quit because of a terminating error.
		.... .1..	Dispatcher lock is held.
		.... ..1.	SRB being scheduled.
		1	Reserved.
17(11)	1	ERRSAVE	One-byte general save space.
18(12)	2	Reserved.	
20(14)	4	ERRBASE2	Second base register.

- For DTESTAE1, the RTCT array of information about address spaces being dumped (RTCTASO) follows the 24-byte recovery routine parameter area in the SDWAVRA.

- For ESTAEXIT (IEEMB879) and SRBFRR (IEEMB880), the variable data consists of as much of the SVC dump exit parameter list (SDEXPARM) as will fit in the SDWAVRA. The SDEXPARM data is mapped (by the IHASDEXP macro) as follows:

Offset	Length	Name	Description
0(0)	4	SDEXECB	ECB used to synchronize SVC dump processing.
4(4)	4	SDEXASCB	ASCB address used to post the ECB.
8(8)	2	SDEXWAID	ASID used to post the ECB.
10(A)	2		Reserved.
12(C)	4	SDEXBFAD	Buffer address for the data moved by the exit.
16(10)	4	SDEXBFLN	Length of the data buffer.
20(14)	4	SDEXORAD	Address of the output routine used to process the buffer.
24(18)	2	SDEXKEYS	Two one-byte keys - for first and second 2K of storage in the buffer.
26(1A)	2	SDEXASID	ASID of the data that was last moved to the buffer.
28(1C)	4	SDEXNDAD	Address of the next data to move to the buffer.
32(20)	4	SDEXCDAD	From address of the data that was last moved to the buffer.
36(24)	4	SDEXBTTA	Start address of the master trace table (if the master trace exit is executing).
40(28)	4	SDEXTTLN	Length of the master trace table (if the master trace exit is executing).
44(2C)	72	SDEXSAVE	Save area for exit use.

Control Blocks Used to Debug SVC Dump Errors

The following control blocks contain key information that can be used to debug problems in SVC dump routines.

- Address Space Control Block (ASCB)
- Recovery Termination Control Table (RTCT)
- RTCT Extension (RTSD)
- SVC Dump Work Area (SDWORK)
- Summary Dump Work Area (SMWK)

Address Space Control Block (ASCB)

The ASCB contains the address of the TCB for the SVC dump task (IEAVTSDT) in the ASCBDUMP field (at offset X'60'). In this TCB, the TCBEXSVC bit (low-order bit at X'CC') is set on while the SVC dump task is executing. The ASCB is mapped by the IHAASCB macro.

Recovery Termination Control Table (RTCT)

The RTCT is pointed to by the CVTRTMCT field (at X'23C') in the CVT. It contains SVC dump information including status bits, an array that describes the SYS1.DUMPxx data sets, and an array that contains information for the address spaces to be dumped. The RTCT is mapped by the IHARTCT macro.

RTCT Extension (RTSD)

The RTSD is pointed to by the RTCTRTSD field (at X'16C') in the RTCT. It contains storage that SDUMP uses to save user parameters and internal variables. The RTSD is mapped by the IHARTSD macro.

SVC Dump Work Area (SDWORK)

The SDWORK is pointed to by the RTCTSDWK field (at X'DC') in the RTCT and the CVTSDBF field (at X'24C') in the CVT. It contains most of the reentrant storage used by SVC dump including register save areas, CCWs, and the I/O buffer that contains the 4104-byte SVC dump records before they are written to the dump data set. The SDWORK work area is mapped by the IHASDWRK macro.

Summary Dump Work Area (SMWK)

The SMWK is pointed to by the RTCTSDSW field (at X'B4') in the RTCT and contains fields used when a summary SVC dump was requested or defaulted (via the SUMDUMP option on the SDUMP macro). It includes counter fields that show how many real frames are used for the real storage buffer that holds the summary dump created for branch-entry callers of SVC dump. The count of real frames held (field SMWKFRHD at X'C6') is zeroed after the summary dump is written to the dump data set and the frames returned to RSM. The SMWK work area is mapped by the IHASMWK macro.

Resource Cleanup for SVC Dump

Resource cleanup performed by SVC dump includes: setting the system dispatchable, setting tasks dispatchable, freeing the summary dump real storage buffer, deleting the TQE for the real storage buffer, restarting the system trace, writing end-of-file on the dump data set, dequeueing the dump data set, and turning off indicators that an SVC dump is in progress. These resources are cleaned up by SVC dump's mainline processing or recovery routines. In special cases, the following routines also perform resource cleanup.

If an address terminates during SVC dump processing, SVC dump's MEMTERM exit (IEAVTSDR) cleans up the resources related to that address space. If the address space was the last to be processed, then all resources are cleaned up and the SVC dump in-progress indicators (high-order bits in the CVTSDBF (at X'24C') and RTCTSDPL (at X'9C') fields) are turned off so that additional dumps can be taken.

SVC dump also uses a timer DIE exit that is contained in module IEAVTSDX at label SCHDIE. This exit ensures that the SVC dump real storage buffer is returned to RSM if SVC dump encounters an error during processing (such as a loop).

SLIP Processor Debugging Aids

The SLIP function of RTM allows you to establish SLIP traps in the system via the SLIP command. When a trap has been established, the SLIP processor is driven by the events that occur in the system.

Entering a SLIP command activates the SLIP command processor and possibly the PER activation/deactivation function (depending on the type of trap). The occurrence of a SLIP event (a system-detected error or a PER interrupt) activates the SLIP processor to check existing traps.

Because a considerable amount of recovery processing is built into the SLIP function, some portion of that recovery is executed if an error occurs. Consequently, when trying to debug the SLIP function, you should have an idea of what the recovery processing is attempting to do. This section discusses the recovery philosophy and provides details for the major SLIP functions.

SLIP Command Processor Recovery

Module IEECB906 provides recovery processing for the SLIP command processor (primarily IEECB905). Most errors encountered in the command processor affect only the command that is issued and not the rest of the system. However, if a PER trap is involved, an error in the command processor could potentially affect the system.

If a PER trap is being disabled or deleted and an error is encountered, IEECB906 disables the non-IGNORE PER trap and schedules IEAVTGLB to deactivate PER. If a PER trap is being set or enabled and an error occurs after SHDRPER has been updated but before IEAVTGLB has been scheduled, IEECB906 tries to schedule IEAVTGLB to activate PER. Additionally, whenever an error occurs, the command processor recovery routine checks to make sure the double-threaded SCE chain is properly chained. Forward and backward pointers found to be in error are repaired if possible. Recovery for the SLIP command processor does not return to mainline processing but requests percolation in the event of an error.

Diagnostic information concerning errors that occur in the command processor is available in a software LOGREC record and a dump. The ESTAE parameter list (mapped by IEEZB906) is part of the LOGREC record. The ESTAE parameter list and SHDR data area along with other information are available in a dump for the error.

SLIP Processor Recovery

Recovery for the SLIP processor is designed to handle both expected and unexpected errors.

Errors which are considered "expected" are:

- A page fault occurs while examining or retrieving the instruction that caused a PER interrupt.
- A page fault occurs while retrieving user-defined data.
- A page fault occurs while processing in IEAVTADR.
- A page fault occurs while examining the instruction that caused a PER interruption in the ASIDSA subroutine of IEAVTSL2.
- The CMSET SSARTO function fails in the DATA subroutine of IEAVTSL2.
- The CMSET SSARTO function fails in the ADRCMSET subroutine in IEAVTSLS.

When the above error conditions are recognized, the SLIP processor attempts to retry at an appropriate point. In general, the retry allows normal trap processing to continue. You may eventually receive an indication that an error has occurred while examining a trap (for example, the data unavailable counter has been incremented). SYS1.LOGREC recording does not occur for these expected errors.

When an unexpected error (none of those shown above) occurs, SLIP processor recovery gathers information concerning the error, cleans up any resources being used by the SLIP processor, and then retries at a point which will terminate processing for the event that caused the SLIP processor to receive control. Diagnostic information concerning the error can be found in the dump taken by the SLIP processor recovery routine (IEAVTSLR). The summary dump usually contains:

- The FRR parameter list (IHASLFP).

  Note: Bits in the SLFPFLGS portion of the FRR parameter list provide an indication of what portion of the SLIP processor encountered the error.

  Note: The FRR parameter list is also recorded as part of the software LOGREC record for the error.

- The SHDR data area.
- The SCE/SCVA data areas being processed at the time of the error.
- The SLIP parameter list (IHASLPL).
- SLIP work areas (IHASLWAs).
- The SLIP register save area.
- The SCE/SCVA data areas representing the enabled non-IGNORE PER trap (if they exist).

Further information concerning the error is included in the software LOGREC record for the error.

PER Activation/Deactivation Recovery

The PER activation/deactivation function is performed primarily by SLIP modules IEAVTGLB, IEAVTSIG, IEAVTLCL, and IEAVTJBN. (Refer also to the process flow for PER activation/deactivation described in a previous topic of this chapter.) In general, if an error is encountered at any point in the PER activation/deactivation process, these modules try to deactivate PER completely. Recovery processing for these modules is described in the following topics.

IEAVTGLB Recovery

If an error is encountered by IEAVTGLB, the recovery for this module gathers information concerning the error, frees the resources held by the mainline code, disables the non-IGNORE PER trap, and then retries at a point in the module which attempts to completely deactivate PER. Diagnostic information concerning the error is recorded in a software LOGREC record and a dump. The information available in the summary dump includes some or all of the following (depending on when the error occurred).

- The FRR parameter list (mapped by FRRWA in module IEAVTGLB).

  Note: The FRR parameter list is also recorded as part of the software LOGREC record for the error.

- The CVT data area.

- The SHDR data area.

- The SCE/SCVA data areas for the non-IGNORE PER trap.

- The model PSA data area.

- The PCCAVT data area.

- The ASCB being processed by IEAVTGLB.

- The name of the job running in the address space being processed by IEAVTGLB.

- The PCCA data area.

- The PER control registers (9, 10, and 11).

If a recursive error is encountered by IEAVTGLB, message IEA414I is sent to the operator and percolation is requested.

IEAVTLCL Recovery

If an error is encountered by IEAVTLCL, the recovery for this module sets tasks dispatchable in the address space, gathers information concerning the error, frees the resources held by the mainline code, and then percolates the error. Diagnostic information concerning the error is available in a software LOGREC record and a dump. The information in the summary dump includes some or all of the following (depending on when the error occurred).

- The FRR parameter list (mapped by FRRPARMS in module IEAVTLCL).

- The CVT data area.

- The SHDR data area.

- The SCE/SCVA data areas for the non-IGNORE PER trap.

- The ASCB for the address space in which IEAVTLCL was running when the error occurred.

- The name of the job in the address space.

Recovery processing for IEAVTLCL is accomplished by message IEA415I.

IEAVTJBN Recovery

If an error is encountered by IEAVTJBN, the recovery for this module gathers information concerning the error, notifies the SLIP user that the status of PER in the system is uncertain (via message IEA422I), and then returns to mainline processing where control is returned to the caller of IEAVTJBN. Diagnostic information concerning the error is available in a software LOGREC record and a dump.

Control Blocks Used by SLIP

The following control blocks contain key information that can be used to debug problems in SLIP routines.

- System Control Blocks
  - Address Space Control Block (ASCB)
  - Logical Configuration Communication Area (LCCA)
  - Prefixed Save Area (PSA)
  - Request Block (RB)
  - Task Control Block (TCB)
- SLIP Control Blocks
  - SLIP Control Element (SCE)
  - SLIP Control Element Variable Area (SCVA)
  - SLIP Header (SHDR)
  - SLIP TSO Element (STE)

ASCB

The ASCBPER bit in the ASCB indicates the status of PER in the address space (1 - PER is active, 0 - PER is inactive).

LCCA

The LCCA contains information related to a PER interrupt processed by SLIP. The PSW (in field LCCAPPSW) and the interrupt code (in field LCCAPERC) for a PER interrupt are saved in the LCCA. The LCCA also contains a pointer (field LCCASLIP) to an area of storage used by the SLIP processor to process a PER interrupt. This area (512 bytes) is split into a parameter list, a work area, and a register save area by IEAVTPER before calling the SLIP processor.

PSA

The PSA contains the external, SVC, and I/O new PSWs. Each PSW has a bit reflecting the status of PER. The PSA also contains the PSASLIP super bit used by SLIP for recursion control.

RB

The RB contains a save area for a PSW (in field RBOPSW). The saved PSW contains a bit reflecting the status of PER.

TCB

The TCB contains secondary non-dispatchability bits. The non-dispatchability bit for SLIP is used while PER is being activated or deactivated in an address space.

SCE/SCVA

The SCE/SCVA pair of data areas is the internal representation of a SLIP trap. In addition to bits which define what was specified for the trap, other indicators provide information relative to the status of the trap. Some important indicators are:

SCEDSABL - a bit that indicates if the trap is enabled (0) or disabled (1).

SCEMATCH - a bit that indicates if the trap has matched at least once since it was enabled (1).

SCVAMLNO - a field that indicates the number of times a trap has matched since it was enabled. (Present only if MATCHLIM was specified or defaulted for the trap.)

SCVADAUN - a field that indicates the number of times data was unavailable for comparison for the trap. Present only if DATA was specified for the trap.

SHDR

The SHDR provides the anchor for the chain of SCE/SCVA control blocks. It is pointed to by CVTRTMS. The SHDRFWD field points to the first SCE on the chain and the SHDRBKWD field points to the last SCE. Other fields in the SHDR are of interest when debugging a SLIP error. They are:

SHDRPFC - this field is zero when there are no enabled SLIP traps. It is one when the SLIP processor and associated routines have been page-fixed by the command processor and no processing is taking place on behalf of any trap. It is two or more when the SLIP processor or portions of IEAVTGLB are running.

SHDRSRBR - this bit is turned on to indicate that IEAVTGLB needs to be scheduled. This bit is usually turned on when IEAVTGLB tries to get a resource (primarily SHDRSEQ) to perform some service but the resource is not available. When on, it indicates that IEAVTGLB will be scheduled to perform the service later. The SLIP command processor (IEECB905) may also set this bit and examines this bit when the sequence word is released.

SHDRPER - this field points to the enabled non-IGNORE PER trap or is zero.

SHDRSEQ - this word is used as a lock to serialize access to the SCE chain for the SLIP command processor (IEECB905), PER activation/deactivation routine (IEAVTGLB), local PER activation/deactivation routine (IEAVTLCL), and the SLIP display processor (IEECB907). The contents of the word indicate the owner of the word as follows:

'CMD' -	IEECB905
'DSP' -	IEECB907
'GLB' -	IEAVTGLB
'Lxx' -	IEAVTLCL (where xx indicates the ASID in which IEAVTLCL is running).

STE

The STE is used to communicate between the SLIP command processor running in the master address space and a TSO user who issued the SLIP command. The STE is created when the TSO user issues the SLIP command and is deleted when SLIP command processing is completed. The STE chain is pointed to by the RTCTSTE field in the RTCT.

Communications Task

The communications task (comm task) handles communications between console operators and the system (user programs and system routines).

The types of communications that the communications task handles are:

- Operator commands from a console as a result of an attention interrupt (on local devices).
- Output to the operator caused by the write-to-operator (WTO), write-to-operator with reply (WTOR), and delete-operator-message (DOM) macro instructions.
- External interrupts that are caused by the operator pressing the interrupt key on the operator control panel. The communications task switches the master console's function to an alternate console.
- Automatic console switching from a failing console to its alternate when an unrecoverable I/O error occurs.
- Console switching as the result of the VARY CHANNEL, VARY CPU, or VARY MSTCONS command.
- Console switching as a result of a processor failure in a multiprocessing system as a part of alternate CPU recovery (ACR).

Before processing WTO, WTOR, and DOM macro requests, the communications task passes control to the job entry subsystem (JES) responsible for the job issuing the request. The JES exit routine may suppress the message, or modify the message text or routing code.

Multiple console support (MCS) is a standard feature that supports up to 99 consoles. With MCS, messages can be routed to up to 15 different functional areas, according to the type of information in the message.

Device independent display operator console support (DIDOCS) is an optional feature that provides uniform console services for various display consoles.

Functional Description

The wait service routine (IEAVMQWR) determines the functions to be performed by the communications task. It is given control by the dispatcher (supervisor control routine IEAVEDS0) after one of the communications task's event control blocks (ECBs) has been posted.

Upon each entry to the wait service routine, the entire list of communications task's ECBs is tested from top to bottom in priority sequence. The posted ECB identifies the service that will be performed by the communications task. As each service is completed, control is returned to the wait service routine and the entire list of ECBs is again tested for an active ECB. When no active ECBs are found, the wait service routine issues the WAIT macro which places the communications task in the wait state until the next communications task ECB is posted.

In addition to testing for posted ECBs, the wait service routine checks other indicators (represented by control bits). The communications task ECBs and control bits are located in the unit control module (UCM) and unit control module entries (UCMEs).

Figure 5-46 lists and describes the ECBs and control bits in the sequence that the wait service routine makes the tests.

ECB or Control Bit	**Function**
UCMARECB (in UCM)	*Alternate CPU recovery* - the process of switching from one processor to another in multiple processor configurations. The communications task switches consoles as required.
UCMXECB (in UCM)	*External interrupt* - switches the master console functions from the current master console to the next available alternate console. This results when the operator presses the interrupt key on the console control panel.
UCMAECB (in UCM)	*Attention interrupt* - prepares the console (from which the interrupt was received) to accept operator input.
UCMECB (in UCME)	*I/O processing complete* - indicates a message has been sent to or received from a console. Results from the interrupt that an I/O device causes after performing each I/O operation.
UCMPF (bit in UCME)	*Console output pending* - indicates a message is queued and ready for some console. Results if (1) one message is queued for several consoles, or (2) a console is busy when a WTO or WTOR message is queued for that console.
UCMSYSJ (bit in UCM)	*Hardcopy output pending* - indicates a message is queued for hardcopy output and ready to be sent to a data set.
Note: Before the following ECB is processed, the communications task tests the WQEs and may issue message IEA405E (80% of WQEs in use), IEA404A (limit of WQEs reached), or IEA406I (shortage relieved).	
UCMOECB (in UCM)	*Queue message for output* - prepares the message posted by the WTO or WTOR macro for output to the appropriate consoles.
UCMSYSI (bit in UCM)	*Cleanup WQE chain* - eliminates WQEs marked for deletion by system functions (such as task termination).
UCMDECB (in UCM)	*Delete operator message* - indicates that a DOM macro has been issued to (1) delete a WTOR message that the operator has not responded to, or (2) delete a WTO message when the issuer has determined that the requested action was performed.
UCMNPECB (in UCM prefix)	*Write NIP messages to buffer* - indicates that NIP messages stored during nucleus initialization can be written.

Figure 5-46. Sequence of Communications Task Processing

Communications Task Control Blocks

The following control blocks are used by the communications task:

UCM *Unit control module* - created at system generation. Contains pointers to the control blocks and routines that support the communications task.

UCME *Unit control module entry* - created at system generation for each generated device. Contains information about the device including attributes, pointer to the UCB, I/O ECB and message queue for the device.

WQE* *Write queue element* - created for each WTO or WTOR request. Contains information about the request including message text and routing code.

ORE *Operator reply element* - created for each WTOR. Contains information about the reply portion of a WTOR request including the buffer to receive the reply.

CQE* *Console queue element* - created for each console that is to receive a message. Contains information about messages queued to particular consoles.

EIL *Event indication list* - created at system generation. Contains pointers to the various ECBs in the UCB and UCME.

RDCM *Resident display control module* - created at system generation. Contains information about a display console.

TDCM* *Pageable display control module* - created at system generation. Contains DIDOCS work and save areas, pointers to related modules, and the screen image.

CXSA *Communications extended save area* - used to communicate among communications task modules.

*These control blocks are located in the private address space of the communications task.

Refer to Figure 5-47 for the relationship of these control blocks.

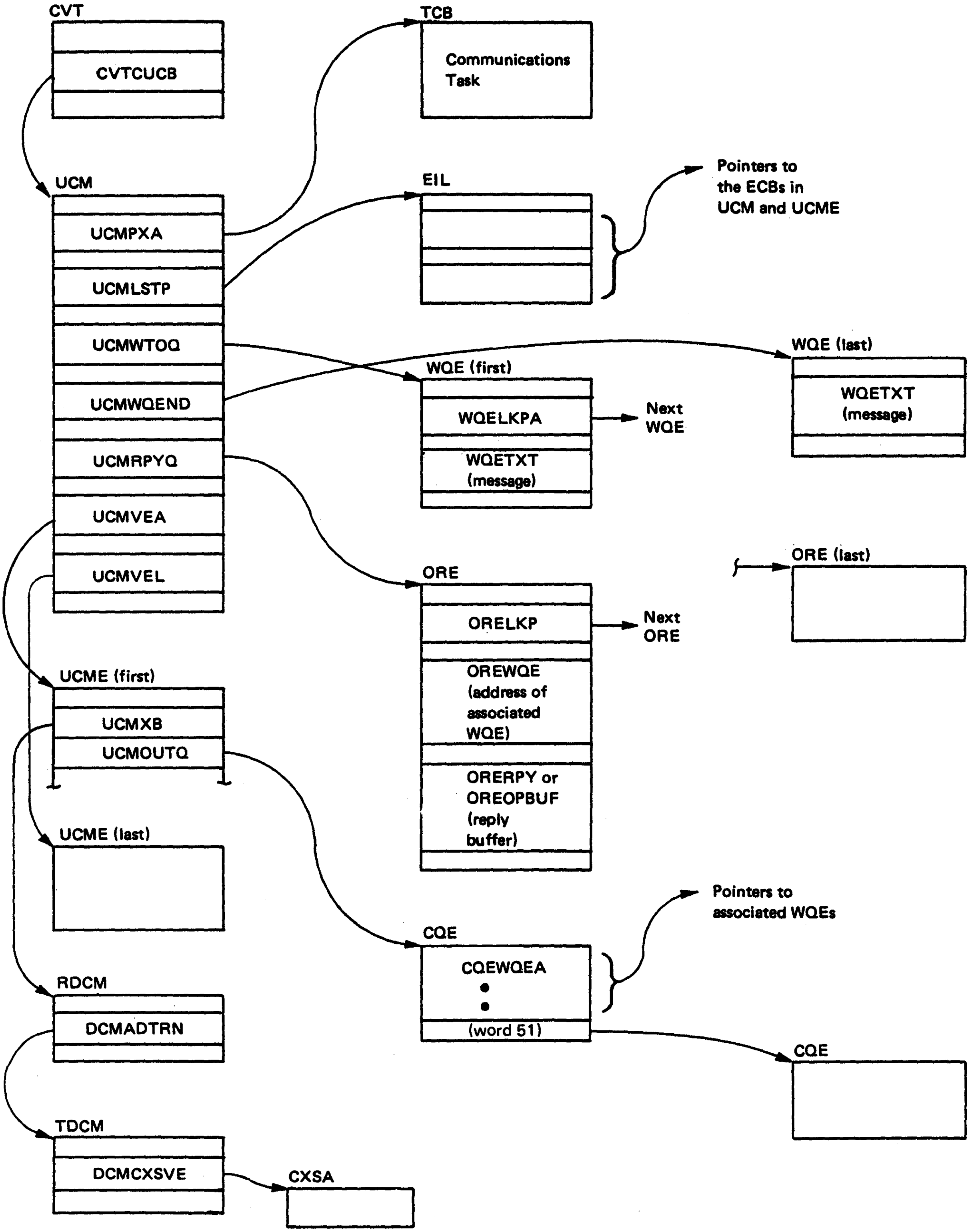


Figure 5-47. Communications Task Control Block Structure

Debugging Hints

Hints for debugging various problems are described in this topic.

Console Not Responding to Attention

If a console is not responding to an attention interrupt, check the following:

- The console attention processor (IEAVVCRA) may not be posting the attention ECB (UCMAECB) in the UCM. The communications task will not process the attention interrupt until the attention ECB is posted. This normally occurs when the console is inactive (UCMUF indicator in the UCME is off), a CLOSE is pending for the device (UCMCF indicator in the UCME is on), or the device does not support interrupts (UCMIF indicator in the UCME is off).

- The attention processor may not be setting the attention pending indicator (UCMAF in the UCME) on for the console causing the interrupt. It is turned on when the attention ECB (UCMAECB) is posted.

- If the attention pending (UCMAF) and busy (UCMBF) indicators in the UCME are both on, the attention interrupt will not be processed until an I/O processing complete interruption is received from the console. I/O processing is performed by a specific device support processor (DSP). The busy indicator (UCMBF) is turned on while the console is waiting for the completion of an I/O operation and is turned off when the I/O completion operation is processed.

Enabled Wait State

If the communications task is in an enabled wait state, check the following:

Normal Case: The communications task has no work to do; that is, no communications task ECBs have been posted. Check the following ECBs (see Figure 5-46 for descriptions and locations of the ECBs).

UCMXECB UCMAECB UCMOECB UCMDECB
UCMARECB UCMNPECB UCMECB

WQE Limit Reached: The system limit for WQEs or OREs has been reached (indicated by message IEA404A).

- Check the following fields in the UCM:

  UCMWQNR - indicates the current number of WQEs in the system.
  UCMWQRSV - indicates how many WQEs are reserved for WTOs being built.
  UCMWQLM - indicates how many WQEs can be built.
  UCMRQNR - indicates the current number of OREs in the system.
  UCMRQLM - indicates how many OREs can be built.

- Check the following indicators in the UCM prefix:

  UCMSYSI - indicates that cleanup of the WQE chain is needed; that is, eliminate WQEs marked for deletion. This indicator is checked by the wait service (IEAVMQWR) and device service (IEAVMDSV) routines; and it is set on by the DOM processing (IEAVMDOM), wait service (IEAVMQWR), console queueing

(IEAVMWSV), multiple-line processing (IEAVMWTO), and WTO/WTOR processing (IEAVVWTO) routines.

UCMSYSJ - indicates that at least one message needs to be sent to the hardcopy log. Possibly, the WQE space is filled with WQEs (messages) that need to be sent to the hardcopy log. This indicator is referenced by the wait service (IEAVMQWR) and device service (IEAVMDSV) routines, and it is set on by the wait service (IEAVMQWR) and console switching (IEAVSWCH) routines.

UCMSYSM - indicates a failure in a composite console. This indicator is used by the console switching (IEAVSWCH) routine.

UCMSYSO - indicates a dummy attention interrupt. This indicator is checked by the wait service (IEAVMQWR) routine. It is set on by the WTO/WTOR processing (IEAVVWTO) routine when the system log is not available and a WTL (write-to-log) is changed to a WTO macro.

Disabled Wait State

The communications task issues only one wait state code, code 007. this code is issued during nucleus initialization when a master console is not available to the system. See wait state code 007 in *OS/VS2 System Initialization Logic.*

Messages or Replies Lost

Messages and replies can be lost or routed incorrectly if the WQE, ORE, or CQE control blocks are not chained correctly.

- To ensure that the WQE chain is intact, check the following:
  - In the UCM, check fields:

    UCMWTOQ - points to the first WQE on the chain.
    UCMWQEND - points to the last WQE on the chain.

  - In each WQE, check:

    WQELPKA - points to the next WQE on the chain.
    WQEORE - indicates that an ORE exists for this WQE.

- To ensure that the ORE chain is intact, check the following:
  - In the UCM, the UCMRPYQ field points to the first ORE.
  - In each ORE, check:

    ORELKP - points to the next ORE on the chain.
    ORERWQE - points to the WQE associated with this ORE.

- To ensure that the CQE chain is intact, check the following:
  - In the UCME (for each console), the UCMOUTQ field points to the first group of 51 CQEs.
  - In each group of CQEs, the CQEWQEA field in the last CQE points to the next group of CQEs on the chain.

  Note: Each CQE is one word; one byte for control bits, and three bytes for a pointer. The CQEs are built in groups of 51. The first 50 CQEs point to WQEs and the last points to the next group of CQEs.

  - In each group of CQEs, the CQEWQEA field in the first 50 CQEs point to their associated WQEs.

  - In each CQE, the CQEFLAG byte contains the control bits.

No Messages on One Console

If messages are not being received on a specific console, check the following:

- The device busy indicator (UCMBF) in the failing console's UCME may be on. A message is not processed until an I/O processing complete interruption is received from the console. I/O processing is performed by a specific device support processor (DSP). The busy indicator (UCMBF) is turned on while the console is waiting for the completion of an I/O operation and is turned off when the I/O completion operation is processed.

- If the console is not busy, ensure that the CQE chain for the console is intact. (Refer to the previous topic "Messages or Replies Lost.")

- If the CQE chain is valid, then check for unusual status in the failing console's UCME and UCB.

Messages Routed to Wrong Console

The console queueing routine (IEAVMWSV) queues messages for specific consoles and builds the CQE chain. If messages are routed to the wrong console, then:

- Ensure that the CQE chain is correct for the failing console. (Refer to the previous topic, "Messages or Replies Lost.")

- Check the routing codes for each console. The UCMRTCD field in each console's UCME defines the routing codes for the respective consoles.

- Check the routing codes for the messages that are being incorrectly routed:

  - In the WTO/WTOR WQE, the WQEROUT field contains the routing codes for the message.

  - In a major multiple-line WQE (for MLWTO), the WMJMRTC field contains the routing codes.

Truncated Messages

If message text is being truncated (the length of the message text is shortened), then:

- The message may exceed the maximum allowable bytes for console messages.

- The console operator may have requested that time stamps and/or job names appear with the messages. Check the following indicators in the UCME for the failing console:

  UCMDISPI - indicates that messages are to appear with both time stamps and job names.
  UCMDISPJ - indicates that only job names are to appear with messages.

Console Switching

Console switching is performed by the IEAVSWCH routine for the following error conditions:

- An I/O error occurs on a console. The failing console's function is automatically switched to its alternate (or, if none available, to the master console). Check the I/O interrupt ECB (UCMECB) in the failing console's UCME. Note that successful I/O completion is indicated by X'7F' in the first byte of the ECB.

- An abnormal termination in the device support processor (DSP) that services the failing console. The failing console's function is automatically switched to its alternate (or, if none available, to the master console). Check the appropriate DSP in load module IGC0007B.

- A processor failure in a multiprocessing environment as a part of alternate CPU recovery (ACR). Consoles are switched as required. Check the alternate CPU recovery ECB (UCMARECB) in the UCM.

Action Message Retention Facility Debugging Aids

If an error occurs in the action message retention facility, pointers to queues and other fields are saved before they are cleared. In the UCM, the following fields and bits are helpful in diagnosing problems.

Field at time of error	**or**	**If zero, use this field**	**Meaning**
UCMPAMRQ		UCMFAMRQ	Pointer to the retained message queue.
UCMPIAMQ		UCMFIAMQ	Pointer to the retained immediate action message queue.
UCMPEAMQ		UCMFEAMQ	Pointer to the retained eventual action message queue.
UCMPAMRN		UCMFAMRN	Number of messages that have been retained.
UCMPRQSD		UCMFRQSD	If on, the retained message queue was successfully scanned before the error occurred.
UCMPIQSD		UCMFIQSD	If on, the retained immediate action message queue was successfully scanned before the error occurred.
UCMPEQSD		UCMFEQSD	If on, the retained eventual action message queue was successfully scanned before the error occurred.
UCMAMRFF		--	If on, the action message retention facility suffered an error.
UCMAMRFS		--	If on, the action message retention facility was active at the time of error and an attempt to restart the facility should be made.

UCMAMRFR	--	If on and another error occurs in the action message retention facility, the fields are not copied again and the facility is not restarted.
UCMAMRFA	--	If on, the action message retention facility is active.

DIDOCS Trace Table

A DIDOCS trace table exists in the pageable DCM (display control module IEETDCM) beginning at field DCMTRACE. The trace table contains the identifiers of up to 16 of the last DIDOCS modules to receive control on the console represented by the pageable DCM.

After each DIDOCS module receives control, it places a two-byte identifier in the trace table. The first byte of the identifier states whether the module is an "E" module such as IEECV*E*TA) or an "F" module (such as IEECV*F*FTA). The second byte of the identifier is the last character in the module name. For example, the identifier for IEECVETA is "EA" and the identifier for IEECVFT1 is "F1." (An exception to this rule occurs during DIDOCS recovery processing. Entries to the ESTAE routine in IEECVET1 are indicated by the identifier "ES.")

When DIDOCS is entered for the first time to perform an operation, the first DIDOCS module to receive control (module IEECVET1) places two bytes of asterisks in the trace table before it stores its identifier. The asterisks signal the beginning of a DIDOCS operation.

DIDOCS-In-Operation Indicator

At offset X'11F' in a console's pageable DCM (display control module IEETDCM) is a field labeled DCMMCSST. When DIDOCS is processing, bit DCMUSE (X'80') in DCMMCSST is set on. This bit remains on during any SVC processing initiated by DIDOCS (SVC34, GETMAIN, FREEMAIN, and EXCP). DIDOCS turns the bit off when DIDOCS exits (via BR14).

DIDOCS Locking

DIDOCS uses two fields (CSAXB and CSAXC) in the communications extended save area (CXSA) to control locking during DIDOCS operations.

The two fields are used as follows:

- When the lock is available:
  - Field CSAXB contains the address of the subroutine that obtains the lock.
  - Field CSAXC contains the address of a BR14 instruction.
- After a DIDOCS module obtains the lock, the subroutine that obtains the lock:
  - Sets field CSAXB to the address of a BR14 instruction.
  - Sets field CSAXC to the address of the subroutine that releases the lock.

- After the DIDOCS module releases the lock, the subroutine that releases the lock:
  - Resets field CSAXB to the address of the subroutine that obtains the lock.
  - Resets field CSAXC to the address of a BR14 instruction.

When the lock is already held by a DIDOCS module (field CSAXB contains the address of a BR14), any attempt by another DIDOCS module to obtain the lock results in a no-operation (NOP).

K Q Command Debugging Aids

The K Q command processor (IEE8B03D) provides the following debugging aids to assist you in diagnosing problems in the command processor.

FRR Work Area

The FRR work area is initialized after syntax processing is complete and IEE8B03D is starting to execute the K Q command. The FRR work area is initialized as follows:

4 bytes -	Module base register.
4 bytes -	Dynamic work area of the module.
8 bytes -	Name of the procedure in control.
1 byte -	ID of the last function that has executed within the procedure.

The last two items are a footprint that identifies the last function that has executed before the error occurred. This information helps to lead you to where the error occurred. Refer to the IEE8B03D module listing for the definitions of the IDs and their use.

Module/Function Trace

The module dynamic area contains procedure/function trace bits. This area is prefixed with 'TRACEBIT' in the dynamic area. Refer to the IEE8B03D module listing for a definition of these bits and their use.

The trace bits are associated in pairs. The first bit indicates if a function was ever entered and the second bit indicates if the function is actively being used. Both bits are turned on when a function is entered and the second bit is turned off on exit from it.

SYS1.LOGREC Data

When an error occurs, the SYS1.LOGREC entry is initialized with data from the variable recording area (SDWAVRA). The data identifies the entry and locates information contained in the module's (IEE8B03D) dynamic area. The information is:

8 bytes -	Load module name IGC0003D.
8 bytes -	Module name IEE8B03D.
8 bytes -	Footprint procedure name.
4 bytes -	Footprint ID in the procedure.
4 bytes -	Character header 'UCM'.
1 byte -	UCMFLG1 flags from UCM at time of error.
3 bytes -	Reserved (initialized to blanks).

4 bytes - Character header 'UCME'.
1 byte - UCMSTS flag from UCME at time of error.*
1 byte - UCMSDS5 flags from UCME at time of error.*
2 bytes - Reserved (initialized to blanks).
4 bytes - UCMWLAST pointer from UCME at time of error.*
4 bytes - UCMMLAST pointer from UCME at time of error.*
4 bytes - Character header 'LCON'.
4 bytes - Pointer to L= console UCME.
4 bytes - Character header 'RCON'.
4 bytes - Pointer to R= console UCME.
4 bytes - Character header 'MCON'.
4 bytes - Pointer to master console UCME.
4 bytes - Character header 'ICON'.
4 bytes - Pointer to issuing console UCME.
4 bytes - Character header 'CQE'.
4 bytes - Pointer to CQE in process.
4 bytes - Character header 'DBUG'.
4 bytes - Pointer to module trace bits.

*These fields contain Xs if cleanup had already occurred which initialized the fields.

Master Trace Debugging Aids

The master trace facility (modules IEEMB808, IEEMB809, and IEEMB816) provides the following debugging aids to assist you in diagnosing problems in master trace.

Copy of Master Trace Table

The communications task maintains a copy of the master trace table header in the CSA (subpool 231). It builds a new table entry and updates the header in this copy. Then the updated header and entry are copied into the master trace table located in the master scheduler's address space.

MSRDA Area

The master scheduler resident data area (MSRDA) contains three bytes of status flags that indicate checkpoints in master trace processing. The CVTMSER field points to this data area. The fields are:

BAMTCNTL - Indicates module in control.
BAMTRECF - Notes possible error recursion.
BAMTITFL - Master trace processing flags.

FRR Work Area

Each master trace module contains data in the FRR work area. The data consists of pointers and individual module processing flags.

Each module has two flags associated with a major function. When a major function is entered, a flag is set on. On exit from the function, another flag is set on. This pair of flags indicates the functions that are invoked and executed.

For IEEMB808, the FRR work area contains:

4 bytes - Pointer to the caller's parameter list.
4 bytes - Pointer to the caller's save area.

The module processing flags are contained in the master trace table in field MTTPFLAG (at X'20').

For IEEMB809, the FRR work area contains:

4 bytes - Pointer to the caller's parameter list.
4 bytes - Pointer to the caller's save area.
4 bytes - Pointer to the module's dynamic work area.
5 bytes - Processing flags.

For IEEMB816, the FRR work area contains:

4 bytes - Pointer to the caller's parameter list.
4 bytes - Pointer to the caller's save area.
4 bytes - Pointer to the IEEMB809 dynamic work area.
4 bytes - Processing flags.
3 bytes - MSRDA processing flags (described earlier).

SDWAVRA Area

In the event of an error in master trace processing, module IEEMB816 (FRR routine) supplies the following data for the variable recording area (SDWAVRA) in the SDWA. The information is:

- Header for master scheduler's resident data area information - 'IEEBASEA'
- Address of IEEBASEA.
- Return code to be passed to the caller of master trace.
- Reason code to be passed to the caller of master trace.
- Component ID - 'SC1B8'.
- Copy of the FRR parameter area passed by the CSECT in error. (For a description of the area, see the previous topic "FRR Work Area.")
- Function in error - 'MTRACE'.
- FMID.
- Header for master trace table data area - 'IEEZB806'.
- Address of the master trace table.
- Address of the old master trace table.
- Header for master trace table entry work area - 'IEEZB806'.
- Address of master trace table entry work area.

Recovery Management Support (RMS)

This section is divided into five parts. They contain diagnostic aids information for the following components of recovery management support (RMS).

- **The machine check handler (MCH)** - which alerts the control program of hardware failures that affect system operation. MCH analyzes machine check interruptions and provides RTM with a diagnostic record describing the hardware failure.

- **The Power Warning Feature (PWF) Support** - which, along with its supporting hardware, prevents the loss of data in real storage when a utility power disturbance occurs by dumping real storage to disk.

- **The channel check handler (CCH)** - which supports IOS in handling channel errors. When a channel error occurs, IOS invokes CCH to aid in the recording of the error.

- **Dynamic device reconfiguration (DDR)** - which supports I/O processing by making data on a malfunctioning device available to processing programs. DDR responds to operator- and system-initiated requests for device swapping.

- **The missing interruption handler (MIH)** - which monitors UCBs at installation-specified time intervals for pending mounts, device swaps, and I/O interruptions.

MCH Diagnostic Aids

This topic contains the following diagnostic aids information that can be used to diagnose problems in the machine check handler (MCH).

- A list of return codes set by MCH modules
- An explanation of the processor work area (PWA)

For additional information related to machine checks, refer to the topic "Debugging Machine Checks" in Section 2 of this publication.

MCH Return Codes

Return codes are set by:

IGFPMMSG -	which schedules error messages.
IGFPMPFX -	which converts a failing address to an absolute address.

The MCH return codes are:

Module	Location	Code	Meaning
IGFPMMSG	register 15	0	Indicates a message is scheduled.
		4	Indicates the message buffer is full. (The message is lost.)
IGFPMPFX	register 15	0	The failing storage address is returned.
		≠0	The failing storage address is invalid.

Processor Work Area (PWA)

When MCH receives control it stores the contents of storage location 0 through 231 in the PWA field PWASFLC. The MCH register save areas are PWA fields PWASA1, PWASA2, PWASA3, and PWASA4. An explanation of the PWA can be found in *Data Areas.*

When a machine check, program check, or restart interruption occurs in MCH, the following PWA fields are used:

- PWASOSW holds the PSW at the time of the interruption.
- PWAINTC holds the machine check or program interruption code.
- PWAFRRCD holds one of the following codes indicating the type of interruption.

X'00000001' -	recursive machine check or threshold exceeded.
X'00000023' -	program interruption.
X'00000024' -	restart interruption.
X'00000025' -	system damage.
X'00000026' -	a zero machine check interrupt code.

PWF Diagnostic Aids

This topic contains the following information that can be used to diagnose problems in the Power Warning Feature (PWF) Support.

- A list of return codes set by PWF
- A description of the data areas that PWF uses
- A dump footprint table
- An appendage footprint table
- A description of LOGREC recording

PWF Return Codes

The PWF return codes are:

Module	Location	Code	Meaning
ICFBDF00	register 15	0	Power warnings are to be disabled on the MCH exit.
User-written routines	register 15	0 4	Continue executing with PWF. Return to the MCH.

PWF Data Areas

This section describes the major data areas used by the Power Warning Feature Support. These data areas include:

- Unit control block (UCB), one byte at offset 17.
- Communications vector table (CVT), one word at offset: 244.
- PWF communications area.

For description of other fields in the UCB and in the CVT see:

- *Data Areas*
- *Debugging Handbook*

UCB Unit Control Block

Offset	Size/Bits Length	Name	Description
17(11)	1	UCBTBYT2	
	.... ..1.	UCB2OPT6	Volume attribute. This volume must be mounted on a device supported by UPS.
	1	UCB2OPT7	Device attribute. This is a device supported by UPS, and is turned on at SYSGEN by specifying AP = YES in the IODEVICE macro.

CVT Communications Vector Table

Offset	Size/Bits Length	Name	Description
244(F4)	4	CVTVOLM2	Address of the PWF communications area
	1	CVTVOLF2	PWF Flag
	1... ...	CVTVOLI2	PWF not initialized
	.xxx xxx		Reserved, set to zero
	3	CVTVOLT2	PWF time delay parameter. This field is set with the WARN = parameter in the CTRLPROG MACRO (0 is the default).

PWF Communication Area

Common Name: PWF Communications Table

Macro ID: ICFWORK

Created by: ECFBIF00

Size: 2048 Bytes

Pointed to by: CVTVOLM2 field of the CVT data area

Function: To provide common information required by the Power Warning Feature Support routines.

Offsets	Length	Name	Description
0(0)	4	ICFADR1	Pointer to footprint table
4(4)	4	ICFADR2	VS2-2 - real address of PCCA vector table
8(8)	4	ICFADR3	VS2-2 - real address of CSD
12(C)	4	ICFADR4	Pointer to ICFBIE00
16(10)	8	ICFSEK00	Seek CCW track 00
24(18)	8	ICFSRC00	Search for track 00
32(20)	8	ICFIC00	TIC CCW track 00
40(28)	8	ICFWRD00	Write data track 00
48(30)	8	ICFSEK01	Seek CCW track 01

56(38)	8	ICFSRC01	Search for track 01
64(40)	8	ICFTIC01	TIC CCW track 01
72(48)	8	ICFWRD01	Write data track 01
80(50)	8	ICFSEK02	Seek CCW track 02
88(58)	8	ICFSRC02	Search for track 02
96(60)	8	ICFTIC02	TIC CCW track 02
104(68)	8	ICFWRD02	Write data track 02
112(70)	8	ICFSEK03	Seek CCW track 03
120(78)	8	ICFSRC03	Search for track 03
128(80)	8	ICFTIC03	TIC CCW track 03
136(88)	8	ICFWRD03	Write data track 03
144(90)	8	ICFSEK04	Seek CCW track 04
152(98)	8	ICFSRC04	Search for track 04
160(A0)	8	ICFTIC04	TIC CCW track 04
168(A8)	8	ICFWRD04	Write data track 04
176(B0)	8	ICFSEK05	Seek CCW track 05
184(B8)	8	ICFSRC05	Search for track 05
192(C0)	8	ICFTIC05	TIC CCW track 05
200(C8)	8	ICFWRD05	Write data track 05
208(D0)	8	ICFSEK06	Seek CCW track 06
216(D8)	8	ICFSRC06	Search for track 06
224(E0)	8	ICFTIC06	TIC CCW track 06
232(E8)	8	ICFWRD06	Write data track 06
240(F0)	8	ICFSEK07	Seek CCW track 07
248(F8)	8	ICFSRC07	Search for track 07
256(100)	8	ICFTIC07	TIC CCW track 07
264(108)	8	ICFWRD07	Write data track 07
272(110)	8	ICFSEK08	Seek CCW track 08
280(118)	8	ICFSRC08	Search for track 08
288(120)	8	ICFTIC08	TIC CCW track 08
296(128)	8	ICFWRD08	Write data track 08
304(130)	8	ICFSEK09	Seek CCW track 09
312(138)	8	ICFSRC09	Search for track 09
320(140)	8	ICFTIC09	TIC CCW track 09
328(148)	8	ICFWRD09	Write data track 09
336(150)	8	ICFSEK10	Seek CCW track 10
344(158)	8	ICFSRC10	Search for track 10
352(160)	8	ICFTIC10	TIC CCW track 10
360(168)	8	ICFWRD10	Write data track 10
368(170)	8	ICFSEK11	Seek CCW track 11
376(178)	8	ICFSRC11	Search for track 11
384(180)	8	ICFTIC11	TIC CCW track 11
392(188)	8	ICFWRD11	Write data track 11
400(190)	8	ICFSEK12	Seek CCW track 12

408(198)	8	ICFSRC12	Search for track 12
416(1A0)	8	ICFTIC12	TIC CCW track 12
424(1A8)	8	ICFWRD12	Write data track 12
432(1B0)	8	ICFSEK13	Seek CCW track 13
440(1B8)	8	ICFSRC13	Search for track 13
448(1C0)	8	ICFTIC13	TIC CCW track 13
456(1C8)	8	ICFWRD13	Write data track 13
464(1D0)	8	ICFSEK14	Seek CCW track 14
472(1D8)	8	ICFSRC14	Search for track 14
480(1E0)	8	ICFTIC14	TIC CCW track 14
488(1E8)	8	ICFWRD14	Write data track 14
496(1F0)	8	ICFSEK15	Seek CCW track 15
504(1F8)	8	ICFSRC15	Search for track 15
512(200)	8	ICFTIC15	TIC CCW track 15
520(208)	8	ICFWRD15	Write data track 15
528(210)	8	ICFSEK16	Seek CCW track 16
536(218)	8	ICFSRC16	Search for track 16
544(220)	8	ICFTIC16	TIC CCW track 16
552(228)	8	ICFWRD16	Write data track 16
560(230)	8	ICFSEK17	Seek CCW track 17
568(238)	8	ICFSRC17	SRC FOR TRK 17
576(240)	8	ICFTIC17	TIC CCW track 17
584(248)	8	ICFWRD17	Write data track 17
592(250)	8	ICFSEK18	Seek CCW track 18
600(258)	8	ICFSRC18	Search for track 18
608(260)	8	ICFTIC18	TIC CCW track 18
616(268)	8	ICFWRD18	Write data track 18
624(270)	8	ICFSEK19	Seek CCW track 19
632(278)	8	ICFSRC19	Search for track 19
640(280)	8	ICFTIC19	TIC CCW track 19
648(288)	8	ICFWRD19	Write data track 19
656(290)	4	ICFWADEV	Device address of primary data set
660(294)	4	ICFWAUCB	UCB address of primary data set
664(298)	7	ICFWACHR	Start of primary extent
671(29F)	1	ICFFLAGA	Flag A field
1...		ICFINOP	PWF function inoperative
.1..		ICFCMTDM	Commit to dump
..1.		ICFUSRC4	User set return code of 4
...x xxxx			Reserved, set to zero
672(2A0)	4	ICFWBDEV	Device address of alternate data set
676(2A4)	4	ICFWBUCB	UCB address of alternate data set
680(2A8)	7	ICFWBCHR	Start of alternate extent

687(2AF)	1	ICFFLAGB	Flag B field
...1		ICFMVT	System type - MVT
...1 ..1.		ICFSVM	System type - VS2 R1
...1 ..11		ICFMVM	System type - VS2 R2
..1. ..1.		ICFVS1	System type - VS1 R3
xx.. xx..			Reserved, set to zero
688(2B0)	4	ICFTRSIZ	Number of bytes per track
692(2B4)	4	ICFTPC	Number of tracks per cylinder
696(2B8)	8	ICFCHR00	Seek/search address for track 00
704(2C0)	8	ICFCHR01	Seek/search address for track 01
712(2C8)	8	ICFCHR02	Seek/search address for track 02
720(2D0)	8	ICFCHR03	Seek/search address for track 03
728(2D8)	8	ICFCHR04	Seek/search address for track 04
736(2E0)	8	ICFCHR05	Seek/search address for track 05
744(2E8)	8	ICFCHR06	Seek/search address for track 06
752(2F0)	8	ICFCHR07	Seek/search address for track 07
760(2F8)	8	ICFCHR08	Seek/search address for track 08
768(300)	8	ICFCHR09	Seek/search address for track 09
776(308)	8	ICFCHR10	Seek/search address for track 10
784(310)	8	ICFCHR11	Seek/search address for track 11
792(318)	8	ICFCHR12	Seek/search address for track 12
800(320)	8	ICFCHR13	Seek/search address for track 13
808(328)	8	ICFCHR14	Seek/search address for track 14
816(330)	8	ICFCHR15	Seek/search address for track 15
824(338)	8	ICFCHR16	Seek/search address for track 16
832(340)	8	ICFCHR17	Seek/search address for track 17
840(348)	8	ICFCHR18	Seek/search address for track 18
848(350)	8	ICFCHR19	Seek/search address for track 19
856(358)	4	ICFSTSIZ	Storage size
860(35C)	4	ICFTME00	Original time value (in MSEC)
864(360)	8	ICFTME01	Original time in TOD units
872(368)	8	ICFTOD00	Time at entry to MCH appendage
880(370)	8	ICFTOD01	Time at inner warning signals
888(378)	8	ICFTOD99	Time to commit to dump
896(380)	4	ICFLRDAT	Date of dump for LOGREC
900(384)	4	ICFLRTIM	Time of dump for LOGREC
904(388)	8	ICFLRCPU	Processor ID for LOGREC
912(390)	8	ICFLRCHA	Channel assignment for LOGREC
920(398)	16	ICFRSVD1	Reserved
936(3A8)	4	ICFPXREG	Prefix register contents at dump time
940(3AC)	4	ICFTRMSA	Trace flags for MSI appendage
944(3B0)	4	ICFTRMCA	Trace flags for MCH appendage
948(3B4)	4	ICFTRDMP	Trace flags for dump
952(3B8)	72	ICFIOMAP	Work area for IOSGEN
1024(400)	512	ICFCNTRK	Buffer for control record
1024(400)	4	ICFCTID	Control track identifier

1028(404)	128	ICFCTCF	Cylinder flags for 128 cylinders
1156(484)	1	ICFCTFLA	Control track flag A
....		ICFCTEMP	Data set is empty
1...		ICFCTFUL	Data set contains valid dump
.... 1...		ICFCTINV	Data set contains invalid dump
1... .1..		ICFCTFBT	Data set contains valid dump but one PWF or more tracks gave I/O errors - PWF alternate track(s) are in use
1160(488)	4	ICFCTTS	Number of bytes per track
1164(48C)	4	ICFCTAWA	Address of PWF work area
1168(490)	4	ICFCTB11	Start of storage block address
1172(494)	4	ICFCTB12	Track addr at which storage blk begins
1176(498)	4	ICFCTB13	End of storage block address
1180(49C)	4	ICFCTB14	Track addr at which storage block ends
1184(4A0)	4	ICFCTB21	Start of storage block address
1188(4A4)	4	ICFCTB22	Track addr at which storage blk begins
1192(4A8)	4	ICFCTB23	End of storage block address
1196(4AC)	4	ICFCTB24	Track addr at which storage block ends
1200(4B0)	4	ICFCTB31	Start of storage block address
1204(4B4)	4	ICFCTB32	Track addr at which storage blk begins
1208(4B8)	4	ICFCTB33	End of storage block address
1212(4BC)	4	ICFCTB34	Track addr at which storage block ends
1216(4C0)	4	ICFCTB41	Start of storage block address
1220(4C4)	4	ICFCTB42	Track addr at which storage blk begins
1224(4C8)	4	ICFCTB43	End of storage block address
1228(4CC)	4	ICFCTB44	Track addr at which storage block ends
1232(4D0)	4	ICFCTB51	Start of storage block address
1236(4D4)	4	ICFCTB52	Track addr at which storage blk begins
1240(4D8)	4	ICFCTB53	End of storage block address
1244(4DC)	4	ICFCTB54	Track addr at which storage block ends
1248(4E0)	4	ICFCTB61	Start of storage block address
1252(4E4)	4	ICFCTB62	Track addr at which storage blk begins
1256(4E8)	4	ICFCTB63	End of storage block address
1260(4EC)	4	ICFCTB64	Track addr at which storage block ends
1264(4F0)	4	ICFCTB71	Start of storage block address
1268(4F4)	4	ICFCTB72	Track addr at which storage blk begins
1272(4F8)	4	ICFCTB73	End of storage block address
1276(4FC)	4	ICFCTB74	Track addr at which storage block ends
1280(500)	4	ICFCTB81	Start of storage block address
1284(504)	4	ICFCTB82	Track addr at which storage blk begins
1288(508)	4	ICFCTB83	End of storage block address
1292(50C)	4	ICFCTB84	Track addr at which storage block ends
1296(510)	8	ICFCTST	TOD at entry to MCH appendage
1304(518)	8	ICFCIED	TOD at end of dump
1312(520)	4	ICFCTTPC	Number tracks per cylinder
1316(524)	4	ICFCTRDA	Device address for restore

1320(528)	4	ICFCTPXR	Prefix register contents at dump time
1324(52C)	4	ICFCTSTS	Highest storage address for LOGREC
1328(530)	4	ICFCTDAT	Date of dump for LOGREC
1332(534)	4	ICFCTTIM	Time of dump for LOGREC
1336(538)	8	ICFCTCPU	Processor ID for LOGREC
1344(540)	8	ICFCTCHA	Channel assignment for LOGREC
1352(548)	184	ICFCTRSV	Reserved
1536(600)	16	ICFRSVD2	Reserved
1552(610)	64	ICFSAVE	Register save area for user exit
1616(650)	8	ICFSNMCP	SA for his MCNPSW
1624(658)	8	ICFMCOPS	SA for his MCOPSW
1632(660)	160	ICFDMPWA	Work area for dump routine
1792(700)	256	ICFRSVD3	Reserved

Name	Offsets/EQU Value	Name	Offsets/EQU Value	Name	Offsets/EQU Value
ICFADR1	0(0)	ICFCTFLA	1156(484)	ICFSRC16	536(218)
ICFADR2	4(4)	ICFCTFUL	1156 X'80'	ICFSRC17	568(238)
ICFADR3	8(8)	ICFCTID	1024(400)	ICFSRC18	600(58)
ICFADR4	12(C)	ICFCTINV	1156 X'08'	ICFSRC19	632(278)
ICFCHR00	696(2B8)	ICFCTPXR	1320(528)	ICFSTSIZ	856(358)
ICFCHR01	704(2C0)	ICFCTRDA	1316(524)	ICFSVM	687 X'12'
ICFCHR02	712(2C8)	ICFCTRSV	1352(548)	ICFTIC00	32(20)
ICFCHR03	720(2D0)	ICFCTST	1296(510)	ICFTIC01	64(40)
ICFCHR04	728(2D8)	ICFCTSTS	1324(52C)	ICFTIC02	96(60)
ICFCHR05	736(2E0)	ICFCTTIM	1332(534)	ICFTIC03	128(80)
ICFCHR06	744(2E8)	ICFCTTPC	1312(520)	ICFTIC04	160(A0)
ICFCHR07	752(2F0)	ICFCTTS	1160(488)	ICFTIC05	192(C0)
ICFCHR08	760(2F8)	ICFDMPWA	1632(660)	ICFTIC06	224(E0)
ICFCHR09	768(300)	ICFFLAGA	671(29F)	ICFTIC07	256(100)
ICFCHR10	776(308)	ICFFLAGB	687(2AF)	ICFTIC08	288(120)
ICFCHR11	784(310)	ICFINOP	671 X'80'	ICFTIC09	320(140)
ICFCHR12	792(318)	ICFIOMAP	952(3B8)	ICFTIC10	352(160)
ICFCHR13	800(320)	ICFLRCHA	912(390)	ICFTIC11	384(180)
ICFCHR14	808(328)	ICFLRCPU	904(388)	ICFTIC12	416(1A0)
ICFCHR15	816(330)	ICFLRDAT	896(380)	ICFTIC13	448(1C0)
ICFCHR16	824(338)	ICFLRTIM	900(384)	ICFTIC14	480(1E0)
ICFCHR17	832(340)	ICFMCOPS	1624(658)	ICFTIC15	512(200)
ICFCHR18	840(348)	ICFMVM	687 X'13'	ICFTIC16	544(220)
ICFCHR19	848(350)	ICFMVT	687 X'10'	ICFTIC17	576(240)
ICFCMTDM	271 X'40'	ICFPXREG	936(3A8)	ICFTIC18	608(260)
ICFCNTRK	1024(400)	ICFRSVD1	920(398)	ICFTIC19	640(280)
ICFCTAWA	1164(48C)	ICFRSVD2	1536(600)	ICFTME00	860(35C)
ICFCTB11	1168(490)	ICFRSVD3	1792(700)	ICFTME01	864(360)
ICFCTB12	1172(494)	ICFSAVE	1552(610)	ICFTOD00	872(368)
ICFCTB13	1176(498)	ICFSEK00	16(10)	ICFTOD01	880(370)
ICFCTB14	1180(49C)	ICFSEK01	48(30)	ICFTOD99	888(378)
ICFCTB21	1184(4A0)	ICFSEK02	80(50)	ICFTPC	692(2B4)
ICFCTB22	1188(4A4)	ICFSEK03	112(70)	ICFTRDMP	948(3B4)
ICFCTB23	1192(4A8)	ICFSEK04	144(90)	ICFTRMCA	944(3B0)
ICFCTB24	1196(4AC)	ICFSEK05	176(B0)	ICFTRMSA	940(3AC)
ICFCTB31	1200(4B0)	ICFSEK06	208(D0)	ICFTRSIZ	688(2B0)
ICFCTB32	1204(4B4)	ICFSEK07	240(F0)	ICFUSRC4	671 X'20'
ICFCTB33	1208(4B8)	ICFSEK08	272(110)	ICFVS1	687 X'22'
ICFCTB34	1212(4BC)	ICFSEK09	304(130)	ICFWACHR	664(298)
ICFCTB41	1216(4C0)	ICFSEK10	336(150)	ICFWADEV	656(290)
ICFCTB42	1220(4C4)	ICFSEK11	368(170)	ICFWAUCB	660(294)
ICFCTB43	1224(4C8)	ICFSEK12	400(190)	ICFWBCHR	680(2A8)
ICFCTB44	1228(4CC)	ICFSEK13	432(1B0)	ICFWBDEV	672(2A0)
ICFCTB51	1232(4D0)	ICFSEK14	464(1D0)	ICFWBUCB	676(2A4)
ICFCTB52	1236(4D4)	ICFSEK15	496(1F0)	ICFWRD00	40(28)
ICFCTB53	1240(4D8)	ICFSEK16	528(210)	ICFWRD01	72(48)
ICFCTB54	1244(4DC)	ICFSEK17	560(230)	ICFWRD02	104(68)
ICFCTB61	1248(4E0)	ICFSEK18	592(250)	ICFWRD03	136(88)
ICFCTB62	1252(4E4)	ICFSEK19	624(270)	ICFWRD04	168(A8)
ICFCTB63	1256(4E8)	ICFSNMCP	1616(650)	ICFWRD05	200(C8)
ICFCTB64	1260(4EC)	ICFSRC00	24(18)	ICFWRD06	232(E8)
ICFCTB71	1264(4F0)	ICFSRC01	56(38)	ICFWRD07	264(108)
ICFCTB72	1268(4F4)	ICFSRC02	88(58)	ICFWRD08	296(128)
ICFCTB73	1272(4F8)	ICFSRC03	120(78)	ICFWRD09	328(148)
ICFCTB74	1276(4FC)	ICFSRC04	152(98)	ICFWRD10	360(168)
ICFCTB81	1280(500)	ICFSRC05	184(B8)	ICFWRD11	392(188)
ICFCTB82	1284(504)	ICFSRC06	216(D8)	ICFWRD12	424(1A8)
ICFCTB83	1288(508)	ICFSRC07	248(F8)	ICFWRD13	456(1C8)
ICFCTB84	1292(50C)	ICFSRC08	280(118)	ICFWRD14	488(1E8)
ICFCTCF	10281(404)	ICFSRC09	312(138)	ICFWRD15	520(208)
ICFCTCHA	1344(540)	ICFSRC10	344(158)	ICFWRD16	552(228)
ICFCTCPU	1336(538)	ICFSRC11	376(178)	ICFWRD17	584(248)
ICFCTDAT	1328(530)	ICFSRC12	408(198)	ICFWRD18	616(268)
ICFCTED	1304(518)	ICFSRC13	440(1B8)	ICFWRD19	648(288)
ICFCTEMP	1156 X'00'	ICFSRC14	472(1D8)		
ICFCTFBT	1156 X'84'	ICFSRC15	504(1F8)		

Dump Footprint Table

The dump footprint table is a 4-byte record of the steps performed by the dump routine. In case the dump routine fails, a printout of this table will aid in diagnosing the failure. This table is located in the PWF communication area. If the dump routine is in control, register 14 contains the address of the footprint table for the WARNA data set. The footprint table for WARNA is located at ICFTRDMP, and the footprint table for WARNB is located at ICFTRDMP+2.

First Byte

1...	Dump routine was started.
.1..	Initialization (PWF) complete.
..1.	Control track read and erased.
...1	Storage scan complete.
.... 1...	One or more cylinders written from real storage.
.... .1..	Transfer of real storage complete.
.... ..1.	Control track written.
....1	Storage protect keys read: transfer from real storage completed.

Second Byte

1...	One or more unit checks occurred.
.1..	One or more uncorrectable storage errors validated.
..1.	Retrying write using spare track.
...1	Retrying write while in I/O subroutine.
.... 1...	Retrying write while in sense subroutine.
.... .1..	Two track errors occurred on one cylinder.
.... ..1.	Failed to write on control track.
....1	Channel checks or device was inoperative.

Appendage (ICFBDF00) Footprint Table

The machine check appendage footprint table is a 4 byte record of the steps performed by the machine check appendage. In case of a failure within this routine, a storage print out of this table will aid in diagnosing the failure. This table is located in the PWF communications area with register 14 containing the address when this routine is in control.

First Byte

1...	Function entered.
.1..	Function inoperative bit is set.
..1.	Function is operative.
...1	First store clock is successful.
.... 1...	Dump immediate flag is not set.
.... .x..	Reserved.
.... ..1.	The power disturbance was transient.
....1	Normal return to machine check handler after transient warning.

Second Byte

1...	Commit to dump, power disturbance is not transient.
.1..	Commit to dump, user routine has been entered.
..1.	User routine returned control with a code of 0 (go dump).
...1	Control has been transferred to dump routine.
.... 1...	User routine returned control with a code of 4 (return to system).
.... .xxx	Reserved, set to zero.

Third Byte

(This byte relates to the availability of WARNA)

1... .... IOSGEN map complete (information to provide the online paths to the device).
.1.. .... At least one path is online.
..1. .... At least one path is clear.
...1 .... Path check routine has been entered.
.... 1... At least one path is available after clear.
.... .1.. Sense I/O has been accepted (code 0 was returned).
.... ..1. CE/DE (channel end/device end) returned form sense I/O.
....1 WARNA processing complete.

Fourth Byte

(This byte relates to the availability of WARNB.) Same as the third byte.

LOGREC Recording

During initialization of Power Warning Feature Support an indication that a power disturbance has occurred is placed in SYS1.LOGREC. When SYS1.LOGREC is printed, the indication of a power disturbance will appear as the first of two IPL records. The format of the IPL record is shown in *OS/VS2 System Programming Library: SYS1.LOGREC Error Recording*

CCH Diagnostic Aids

This topic contains the following information that can be used to diagnose problems in the channel check handler (CCH).

- CCH message IGF002I information
- PCCA fields used to trace the activity of CCH

Message IGF002I

There is one message issued by CCH:

```
IGF002I  CHANNEL DETECTED ERROR ON ddd, pa, err, op, stat
```

The variable fields of the message (ddd, pa, err, op, stat) are put into the channel data area (CDA) by IGFCCHCR. The following chart shows what CDA fields are used and where IGFCCHCR obtains the information.

CDA Field	Contents	Obtained from
CDACCHBL	Error indicator	PCCACHBL
CDACCHOP	CCW operation code	LRBCFCCW
CDACCHPA	Channel set ID	LRBCMPPA
CDACCHRn	Message statistics and static message field	Dummy module
CDACCHST	Unit and channel status	LRBCFCSW
CDACCHUA	Unit address	LRBCCUA2

IGFCCHCR formats the message in one of the CCH message buffers: CDACCHM1 or CDACCHM2. The recovery termination manager (RTM) issues the message through a RECORD macro instruction. After the message is issued, IGFCCHCR sets the record buffer (CDACCHRn) to 0.

PCCA Fields Showing CCH Footprints

The following fields in the PCCA (starting at X'134') may be used to trace the action of CCH.

PCCACHF1	CCH footprint byte 1	
1...	PCCACF11	I/O supervisor registers have been saved
.1..	PCCACF12	UCB address supplied by I/O supervisor is 0
..1.	PCCACF13	ERPIB already exists
...1	PCCACF14	IGFCCHSI entered
.... 1...	PCCACF15	ICFCCHII entered
.... .1..	PCCACF16	IGFCCHFE entered
.... ..1.	PCCACF17	IGFC60 entered
....1	PCCACF18	IGFC70 entered
PCCACHF2	CCH footprint byte 2	
1...	PCCACF21	IGFC80 entered
.1..	PCCACF22	IGFCIC entered
..1.	PCCACF23	IGFCCHRD entered
...1	PCCACF24	IGFCCHMP entered
.... 1...	PCCACF25	IGFCCHUC entered
.... .1..	PCCACF26	IGFCCHAS entered
.... ..1.	PCCACF27	IGFCCHIO entered
....1	PCCACF28	IGFCCHEX entered
PCCACHF3	CCH footprint byte 3	
1...	PCCAISRB	SRB for IECVIRST scheduled
.1..	PCCASLCK	Space allocation lock held by CCH
..xx xxxx		Reserved
PCCACHS1	CCH internal switch 1	
1...	PCCACCMP	Command register parity is valid
.1..	PCCACNRE	No recording to be done by CCH
..1.	PCCACFRR	CCH FRR is in the stack
...1	PCCACNLS	Record only; ERPIB not put in EWA
.... 1...	PCCACAND	Attention has been presented
.... .1..	PCCACIBC	ERPIB already created for this error
.... ..1.	PCCACUCB	UCB is invalid
....x		Reserved
PCCACHS2	CCH internal switch 2	
1...	PCCACIOR	I/O restart is required
.1..	PCCACALT	Use the alternate return to I/O supervisor
..1.	PCCACMOD	Module required to analyze channel logout is unavailable
...1	PCCACNLG	Channel failed to log or store an LCL
.... 1...	PCCACURC	CAT entry is valid, but channel type is not recognized
.... .1..	PCCACCRA	Channel reconfiguration hardware active
.... ..xx		Reserved

DDR Diagnostic Aids

This topic contains the following information that can be used to diagnose problems in dynamic device reconfiguration (DDR).

- A description of the DDR task
- An explanation of the DDRCOM
- A mapping of the DDR error recovery parameter list (DERPLIST)
- The return codes set by DDR modules
- An explanation of DDR software recording
- A description of a DDR storage dump

DDR Tasks

There are several DDR tasks: a DDR task for DASD swaps, a DDR task for tape and unit record swaps, and a separate DDR task for each tape swap when a tape is being repositioned on a new tape drive.

The DDR tasks are created via ATTACH macro instructions as subtasks of the master scheduler task. The DDR tasks are in the master scheduler's address space only when the DDR swaps are active. When all swaps are complete, the DDR tasks terminate. The tasks are identified by the TCBTID field being 252 (X'FC').

Tape, disk, and unit record swaps are performed in the master scheduler address space (ASID=1). When a tape is swapped, part of routine IGFDT1 is executed in the user's address space to obtain information about the tape. This address space is pointed to by the UCBASID field of the UCB for the tape.

DDR Communication Table (DDRCOM)

There is a DDRCOM for each DDR request. There are three DDRCOM chains, pointed to from the CVT, for the following:

- DASD swaps
- Tape and unit record swaps
- Tape swaps in the repositioning phase

A DDRCOM for a tape request is moved from the DDRCOM chain (pointed to by CVTTPUR) to the DDRCOM chain (pointed to by CVTTRPOS) when the tape enters the repositioning phase.

The format of the DDRCOM is described in the *Debugging Handbook.* Figure 5-48 shows typical DDRCOM chains.

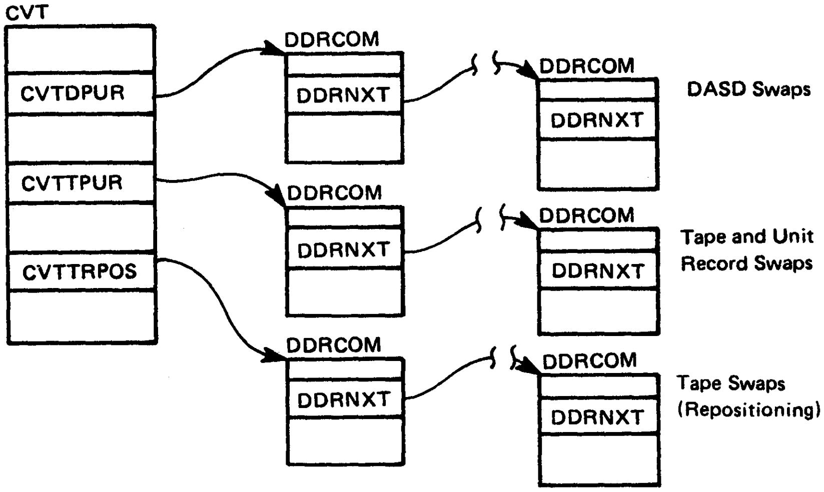


Figure 5-48. Typical DDRCOM Chains

DDR Error Recovery Parameter List (DERPLIST)

The DDR error recovery parameter list is used by a DDR error recovery routine, IGFDE0, to determine whether to cancel or create an error recovery environment. IGFDE0 invokes the ESTAE routine through an ESTAE macro instruction and passes the address of DERPLIST as a parameter.

When an error occurs, IGFDE1, another DDR error routine, uses DERPLIST to determine what queued resources and storage are to be freed, to locate the address of the retry routine, and to determine what information to pass to the retry routine.

Each module creates its own DERPLIST, and the address is known locally.

Figure 5-49 (part 1) is a map of the DERPLIST. The fields shown in the map are listed in alphabetic order in Figure 5-49 (parts 2 and 3).

DERPLIST

DEC	*HEX*				
0	0	DERFUNK	DERSWCHS	DERQUEUE	
4	4	DERREC			
8	8	DERSPEC			
12	C	DERRETRY			
16	10	DERDASPN	DERDALNG		
20	16	DERRSRC	DERLRC	DERPFX	
24	18	DERRCODE			
28	1C	DERRDATA			
32	20	DERRDDR			
36	24	DERRSAVE			
40	28	DERGMSPN	DERGMLNG		
44	2C	DERGMADR			
48	30	DERRCRTY			
52	34	DERDDPA			
56	38	DEREDATA 60 3C			

Figure 5-49 (Part 1 of 3). DDR Error Recovery Parameter List

DEC	HEX	LEN	Field	Description
17	11	3	DERDALNG	Length of the module workarea.
16	10	1	DERDASPN	Subpool of the module workarea.
52	34	4	DERDDPA	If the DDR device dependent exit is in control (X'08' on in DERSWCHS), this field contains the address of the parameter list passed to the DDR device dependent exit. Otherwise, zero.

Figure 5-49 (Part 2 of 3). DDR Error Recovery Parameter List

DEC	HEX	LEN	Field	Description
56	38	8	DEREDATA	Additional DDR device dependent exit data. If the load of the DDR exit is in progress (X'02' on in DERSWCHS), this field contains the name of the module being loaded. If the DDR exit returned an invalid return code (X'04' on in DERSWCHS), the first word of this field contains the invalid return code passed back by the exit and the second word contains the highest valid return code for that call. If the DDR exit is in control (X'08' on in DERSWCHS), this field contains the load module name of the DDR device dependent exit (if the exit is in LPA or LINKLIB). Otherwise, this field is zero.
0	0	1	DERFUNK	Function code.
			DERINIT	X'01'. Issue the initial ESTAE for the calling module.
			DERCANC	X'02'. Issue ESTAE 0 for the calling module.
44	32	4	DERGMADR	Address of the area obtained by a GETMAIN macro instruction specified in DERGMLNG.
41	29	3	DERGMLNG	Length of area obtained by a GETMAIN macro instruction in a DDR module.
40	28	1	DERGMSPN	Subpool number of area obtained by a GETMAIN macro instruction in a DDR module.
21	15	1	DERLRC	Local return code.
22	16	2	DERPFX	Rub prefix.
2	2	1	DERQUEUE	Queue indicators. (If zero, determine the queue from DDRCOM.)
			1...	DASD queue.
			.1..	Tape/unit record queue.
			..1.	Tape repositioning queue.
			...x xxxx	Reserved.
24	18	4	DERRCODE	Module base (CODEREG).
48	30	4	DERRCRTY	Retry address when the DDR device dependent exit supplied an invalid return code.
28	1C	4	DERRDATA	Workarea base (DATAREG).
32	20	4	DERRDDR	Address of the DDRCOM (DERPTR).
4	4	4	DERREC	Address of the 24-byte recording of the ID to be used during IGFDE1 processing.
12	C	4	DERRETRY	Address of retry routine after IGFDE1 processing.
36	24	4	DERRSAVE	Save area register.
20	14	1	DERRSRC	Resources controlled by this module (compared with DDROWN in DDRCOM).
			1...	DERTAPE - Tape allocation resource held.
			.1..	DERUREC - Unit record allocation resource held.
			...1	DERDISK - Disk allocation resource held.
			...x xxxx	Reserved.
8	8	4	DERSPEC	Address of special cleanup exit during IGFDE1 processing.
1	1	1	DERSWCHS	IGFDE1 indicator switches.
			1...	DERCRASH - No SDWA exists, do not issue a SETRP.
			.1..	DERPERK - Do not retry; issue SETRP to continue termination.
			..1.	DERECALL - This DDR module is recallable on error conditions.
			...1	DERESTAE - Initial ESTAE issued was successful.
			.... 1....	DDR device dependent exit in control.
			.... .1..	DDR device dependent exit supplied an invalid return code.
			.... ..1.	Load of DDR device dependent exit in progress.
			1	Vary service routine in control.

Figure 5-49 (Part 3 of 3). DDR Error Recovery Parameter List

DDR Return Codes

The return codes for DDR are:

Object Module	Location of Code	Return Code	Meaning
IGFDD0	register 15	0	Valid condition
		4	Invalid condition
		12	Catastrophic condition
IGFDD1	register 15	0	Valid condition
		4	Invalid condition
		12	Catastrophic error, DDRINVI, DDRTER1 set
IGFDI1	register 15	0	Element removed, queue not empty
		4	Element removed, queue empty
		12	Catastrophic error, queue empty
IGFDL1	register 15	0	Device found
		4	Device not found
		12	Catastrophic error
IGFDM0	register 15	0	Message issued without error
		4	Invalid input detected in DDRCOM
		12	Catastrophic error
IGFDM1	register 15	0	Successful
		4	Unsuccessful
		12	Catastrophic error
IGFDR0	register 15	0	Recording initiated successfully
		4	Invalid function requested
		12	Catastrophic error
IGFDT0	register 15	0	SWAP completed successfully
		12	SWAP terminated
IGFDT1	register 15	0	Valid
		4	Invalid
		12	Catastrophic error
IGFDT2	register 15	0	Positioning completed successfully
		4	I/O error
		8	Wrong volume loop
		12	Catastrophic error
IGFDU0	register 15	0	Valid condition
		4	Invalid condition
		12	Catastrophic condition
IGFDV0	register 15	0	SWAP completed
		4	SWAP canceled or terminated
IGFDV1	register 15	0	Valid
		4	Invalid
		12	Catastrophic error

Software Recording

No special DDR information is recorded in the variable fields of the software ABEND record. However, the FRR ID field of the load module/CSECT/FRR ID data contains the date of the compilation of the CSECT. This may be used in verifying the level of the module against the available microfiche.

DDR Storage Dumps

Whenever an abnormal termination occurs, the DDR ESTAE module IGFDE1 uses the SDUMP macro to dump those portions of main storage necessary to diagnose the problem. Included in the dump are the SQA, PSA, IPA, trace table, CSA, and the local storage. The dump is titled "Dynamic Device Recovery Dump."

MIH Diagnostic Aids

This topic contains the following information that can be used to diagnose problems in the missing interruption handler (MIH).

- A description of the MIH process
- A description of the MIH work area
- An explanation of MIH software recording
- A description of an MIH storage dump

MIH Process

The MIH process is invoked through the ATTACH macro instruction during master scheduler initialization and executes in the master scheduler address space (ASID = 1). Two modules perform the MIH processing; IGFTMCHK, a load module in the link pack area, and IGFTMC00, a CSECT in the nucleus.

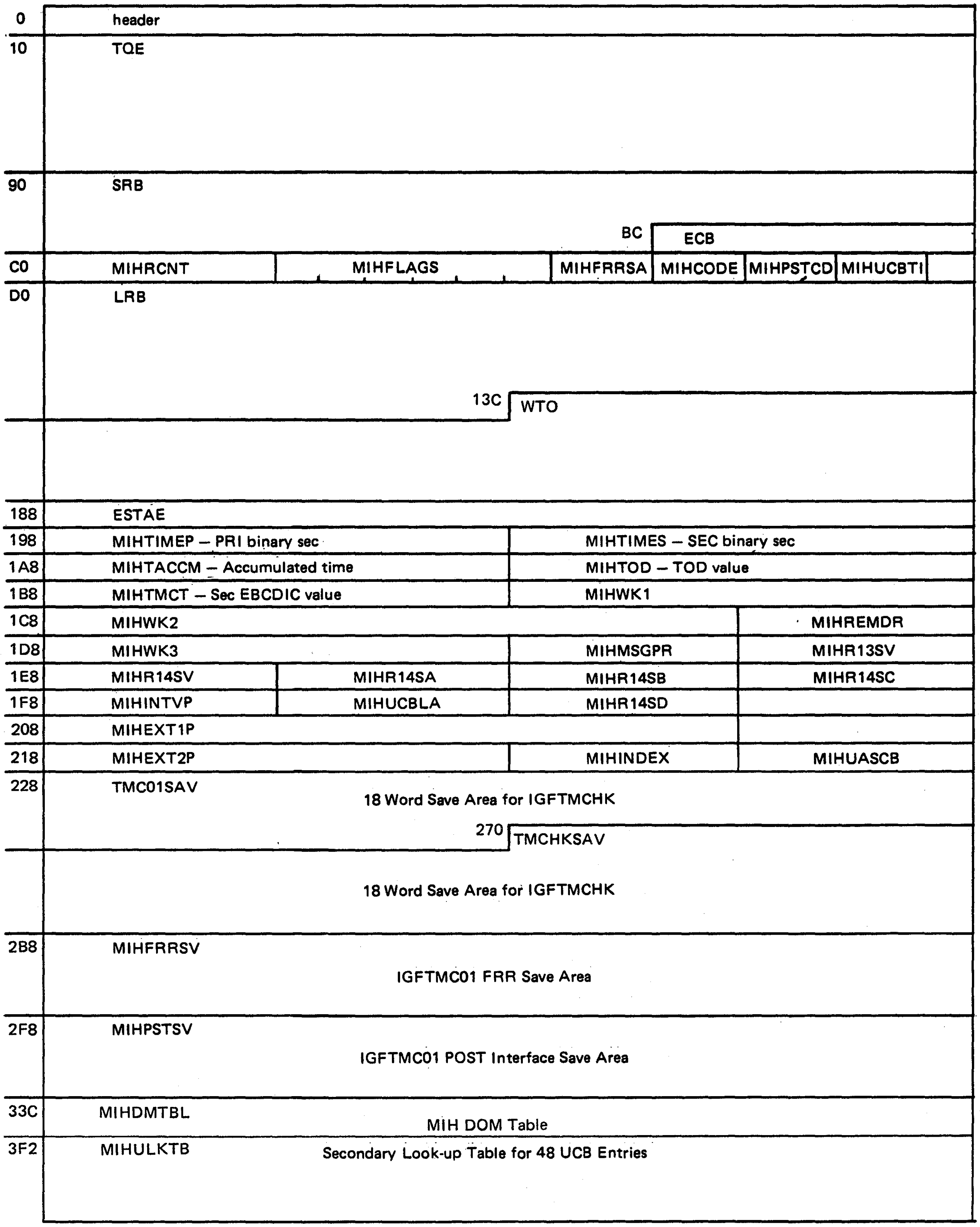
MIH Work Area

IGFTMCHK obtains the MIH work area from fixed SQA, subpool 245. Both IGFTMCHK and IGFTMC00 use this work area which contains the TQE, SRB, LRB, message ECB, WTO buffer, ESTAE parameter list, 16 and 18 word save areas, MIH first and second level exit parameter lists, the message queue pointer, all general usage work areas, and the secondary look-up table. The first 16 bytes of the work area contain an identifier (**MIH WORK AREA**). IGFTMCHK initializes the remainder of the work area to zeros. Once obtained, the work area is not freed. If the MIH process is terminated, the work area is retained for problem determination.

A pointer to the MIH work area is contained in register 10 and at location X'1C' in the IGFINTVL CSECT. Figure 5-50 shows a mapping of the work area. Fields of special interest in the MIH work area are:

Offset (Hex)	Length	Name	Description
C0	4	MIHRCNT	MIH FRR retry counters for IGFTMCHK and IGFTMC01 respectively.
C4	3	MIHMCHKF	IGFTMCHK flag bytes:
C4	1	MIHFLAG1	MIH IGFTMCHK flag byte 1:
	1...	MIHESTAA	ESTAE active.
	.1..	MIHIPLRA	SVC 76, IPL record active.
	..1.	MIHIPLOK	SVC 76, IPL record written.
	...1	MIHUCBSK	UCB scan active.
	.... 1...	MIHINITC	IGFTMCHK initialization complete.
	.... .1..	MIHSDIEA	SET DIE successful.
	.... ..1.	MIHRDEA	SVC 76, RDE active, UPD time/date stamp.
	1		Reserved.
C5	1	MIHFLAG2	MIH IGFTMCHK flag byte 2:
	1...	MIHLGRCA	LOGREC processing active.
	.1..	MIHTIMEA	SVC 11, TIME macro active.
	..1.	MIHRECDA	RECORD LOGREC active.
	...x xxx.		Reserved.
	1	MIHNSDWA	ESTAE routine - no SDWA provided.
C6	1	MIHFLAG3	MIH IGFTMCHK flag byte 3:
	1...	MIHMSGPA	Message processing active.
	.1..	MIHMSGBA	Message build active.
	..1.	MIHMBLDA	Message build routine active.
	...1	MIHWTOA	SVC 35, WTO message active.
	.... 1...	MIHACTNF	Action message requires DOM flag.

	.... .1..	MIHDMSET	DOM table search flag.
	xx		Reserved.
C7	3	MIHMC01F	IGFTMC01 flag bytes:
C7	1	MIHFLAG1	MIH IGFTMC01 flag byte 1:
	1...	MIHFRRA	FRR active.
	.1..	MIHEXTIA	MIH exit 1 active.
	..1.	MIHUCBSA	UCB scan active.
	...1	MIHUCBVA	Valid UCB processing.
	.... 1...	MIHEXT2A	MIH exit 2 active.
	.... .1..	MIHTQEA	MIH TQE ENQ active.
	.... ..1.	MIHEXITE	MIH permanent error in exit.
	1	MIH01RTY	MIH FRR retry active.
C8	1	MIHFLAG5	MIH IGFTMC01 flag byte 2:
	xxxx xxxx		Reserved.
C9	1	MIHPOSTF	MIH POST flag byte:
	1...	MIHPOST	MIH posting active.
	.111 1111		TS instruction is used to serialize posting.
CA	2	MIHFRRSA	MIH IGFTMC01 FRR flag save area
CC	1	MIHCODE	MIH condition translation code.
CD	1	MIHPSTCD	MIH POST code.
CE	1	MIHUCBTI	MIH exit index in error.
CF	1		Reserved.



0	header					
10	TQE					
90	SRB					
BC	ECB					
C0	MIHRCNT	MIHFLAGS	MIHFRRSA	MIHCODE	MIHPSTCD	MIHUCBTI
D0	LRB					
13C	WTO					
188	ESTAE					
198	MIHTIMEP – PRI binary sec	MIHTIMES – SEC binary sec				
1A8	MIHTACCM – Accumulated time	MIHTOD – TOD value				
1B8	MIHTMCT – Sec EBCDIC value	MIHWK1				
1C8	MIHWK2	MIHREMDR				
1D8	MIHWK3	MIHMSGPR	MIHR13SV			
1E8	MIHR14SV	MIHR14SA	MIHR14SB	MIHR14SC		
1F8	MIHINTVP	MIHUCBLA	MIHR14SD			
208	MIHEXT1P					
218	MIHEXT2P	MIHINDEX	MIHUASCB			
228	TMC01SAV	18 Word Save Area for IGFTMCHK				
270	TMCHKSAV	18 Word Save Area for IGFTMCHK				
2B8	MIHFRRSV	IGFTMC01 FRR Save Area				
2F8	MIHPSTSV	IGFTMC01 POST Interface Save Area				
33C	MIHDMTBL	MIH DOM Table				
3F2	MIHULKTB	Secondary Look-up Table for 48 UCB Entries				

Figure 5-50. MIH Work Area

Software Recording

The CSECT name, PTF number and date are moved into the variable field of the software ABEND record. The PTF number and date can be used in verifying the level of the module against the available microfiche. The FRR ID field contains the load module/CSECT/FRR ID.

MIH Storage Dumps

Whenever an abnormal termination occurs, storage areas necessary to diagnose a problem are dumped by the ESTAE recovery routine, IGFTMCHS, in module IGFTMCHK, using the SDUMP macro instruction. Included in the dump are the SQA, NUC, and trace table.

Service Processor Call SVC and MSSFCALL DIAGNOSE Instruction

Note: This topic describes the MVS support for processor complexes with the monitoring and system support facility (MSSF). See the topic "Service Processor Call SVC and SERVICE CALL Instruction" for a description of the MVS support for processor complexes with the Service Processor Architecture.

The Service Processor Call SVC (SVC 122), formerly called the MSSFCALL SVC, is a type 2 extended SVC with a routing code of 6 in register 15. It is used to communicate with the monitoring and system support facility (MSSF) in the processor controller of the processor complex.

MVS system routines issue the Service Processor Call SVC to request various hardware functions and to obtain hardware status information.

When the Service Processor Call SVC is issued, the Service Processor Call SVC processing routine (IEAVMSF) processes the SVC request. IEAVMSF issues the DIAGNOSE instruction (operation code X'83'). The MSSFCALL DIAGNOSE instruction directly interfaces with the MSSF, which is a logical unit within the processor controller.

IEAVMSF issues the SERVICE CALL instruction (rather than the MSSFCALL DIAGNOSE instruction) on processor complexes with the Service Processor Architecture.

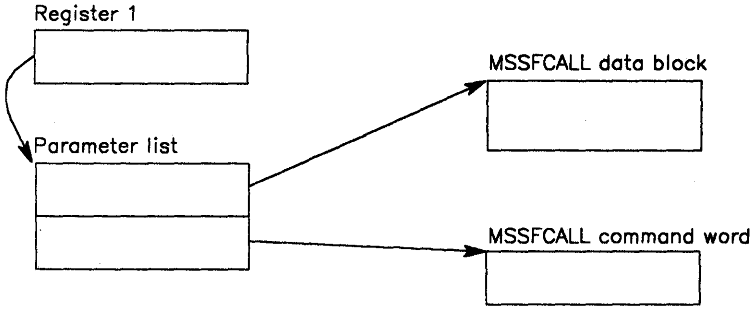
For program logic information, refer to the *System Logic Library*, *System Initialization Logic*, and *Input/Output Configuration Program (IOCP) Logic.*

The following topics describe:

- Service Processor Call SVC (SVC 122) used with the MSSF
- Service Processor Call SVC (SVC 122) processing and control blocks used with the MSSF
- MSSFCALL DIAGNOSE instruction

Service Processor Call SVC (SVC 122) Used With the MSSF

To issue the Service Processor Call SVC for the MSSF, the caller prepares an MSSFCALL data block (also called a service processor data block) and an MSSFCALL command word (also called a service processor command word) in the caller's own private area. The caller sets register 15 to 6 (the routing code) and sets register 1 to the address of a parameter list, which points to the data block and the command word.



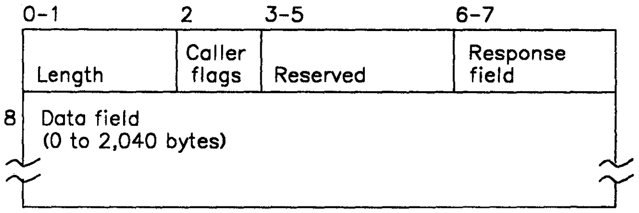
The parameter list is a standard SVC parameter list. The MSSFCALL data block contains command-dependent information and is used to send data to the MSSF and to receive responses and data from the MSSF. The MSSFCALL command word specifies the function requested.

The following topics describe:

- MSSFCALL data block
- SVC abend and return codes with the MSSF

MSSFCALL Data Block

The MSSFCALL data block (also called the service processor data block) is a variable-length data area. The format of the data block is:



0-1	2	3-5	6-7
Length	Caller flags	Reserved	Response field
8 Data field (0 to 2,040 bytes)			

Length - bytes 0 and 1
specifies the length of the data block in eight-byte increments. The minimum length is eight (X'0008') and the maximum length is 2,048 (X'0800') bytes. The length varies depending on the command request. Errors in the specification of the length are indicated by the MSSF in the response field.

Caller flags - byte 2
specifies flags associated with the command request. Set the appropriate flags if used with the command; otherwise, set to X'00'. Errors in the specification of caller flags are indicated by the MSSF in the response field.

Reserved - bytes 3 through 5
set to X'000000'.

Response field - bytes 6 and 7
receives the two-byte response from the MSSF. At the completion of the request, the MSSF sets this two-byte field to indicate the status of the request. The field must be set to X'0000' before issuing the Service Processor Call SVC.

Data field - byte 8 to end of data block
specifies command-dependent information, if any, related to the command. Before issuing the SVC, the caller either puts the appropriate command-dependent information into the data field or sets the field to zeroes. The MSSF returns command-dependent data in the data field if applicable for the command.

SVC Abend and Return Codes With the MSSF

Abend Code - The Service Processor Call SVC processing routine (IEAVMSF) issues system abend X'67A' if the caller is not authorized to issue the SVC or if an error occurs during processing of the SVC. Refer to *System Codes* for a description of abend X'67A'.

Return Codes - IEAVMSF returns the following return codes in register 15 to the caller of the SVC.

Code	Meaning
00(00)	Successful completion - the MSSFCALL data block contains the MSSF response (in bytes 6 and 7) and any data (in the data field) requested on the command.
04(04)	The MSSF is busy - the caller can reissue the SVC.
08(08)	One of the MSSFCALL SVC processing control blocks (MSFCB, MSFAB, or MSFKB) is in use - the caller can reissue the SVC.
12(0C)	The MSSF is not available on the processor complex or the control block chain is invalid.
16(10)	The MSSF did not respond during a 10-second disabled spin.

SVC Processing and Control Blocks With the MSSF

The Service Processor Call SVC processing routine (IEAVMSF) provides the interface between system routines and the monitoring and system support facility (MSSF) in the processor controller. IEAVMSF processes requests made by programs issuing the Service Processor Call SVC (including the users of the data communications link) and routines that branch-enter IEAVMSF.

When processing the Service Processor Call SVC request, IEAVMSF copies the caller's MSSFCALL data block to the data block pointed to by the MSFCB, MSFAB, or MSFKB control block. The MSSF puts its responses and data into the data block (pointed to by the MSFCB, MSFAB, or MSFKB). IEAVMSF then copies the MSSF response and data, if any, into the caller's MSSFCALL data block.

Figure 5-51 shows an overview of IEAVMSF processing.

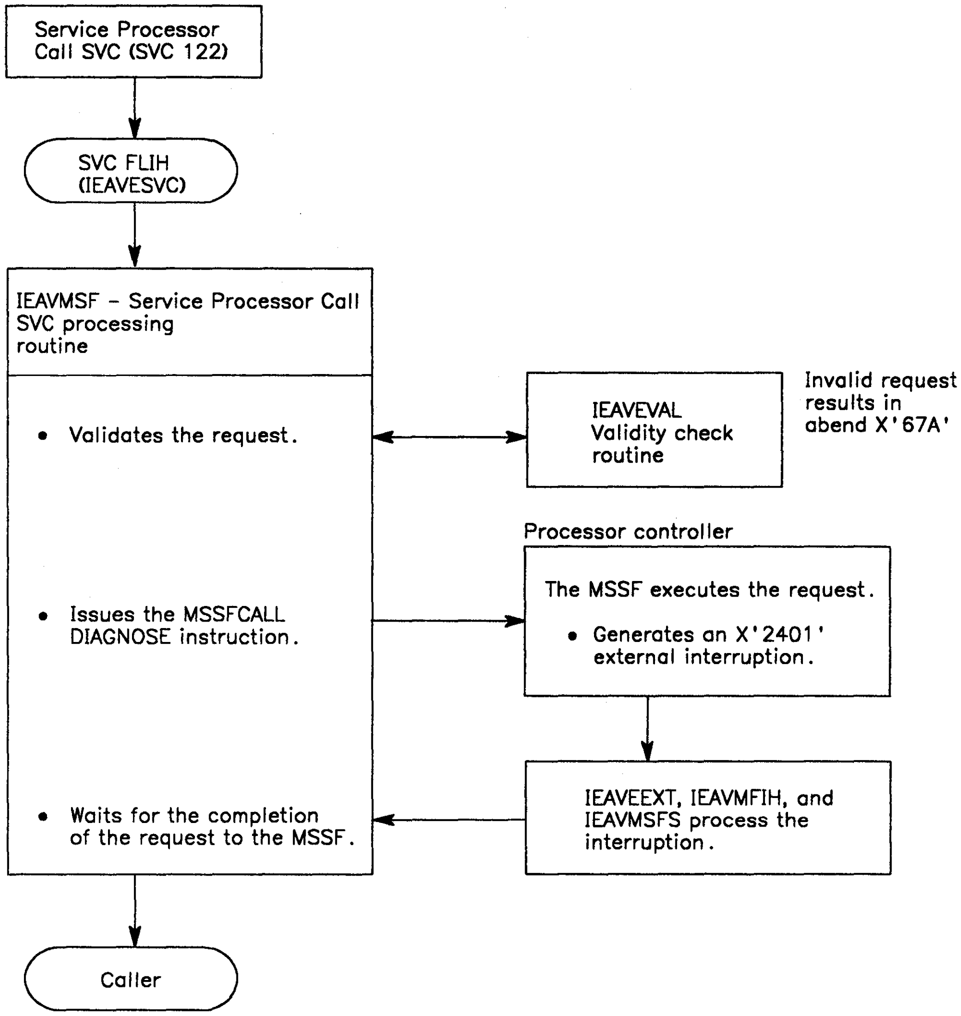

Service Processor Call SVC (SVC 122)
SVC FLIH (IEAVESVC)
IEAVMSF - Service Processor Call SVC processing routine
• Validates the request.
• Issues the MSSFCALL DIAGNOSE instruction.
• Waits for the completion of the request to the MSSF.
IEAVEVAL Validity check routine
Invalid request results in abend X'67A'
Processor controller
The MSSF executes the request.
• Generates an X'2401' external interruption.
IEAVEEXT, IEAVMFIH, and IEAVMSFS process the interruption.
Caller

Figure 5-51. Overview of SVC Processing With the MSSF

SVC Control Blocks Used with the MSSF

IEAVMSF uses three control blocks to serialize Service Processor Call SVC requests, the MSSFCALL control block (MSFCB), the MSSFCALL attention block (MSFAB), and the MSSFCALL communications block (MSFKB). All are created at NIP time by the RIM module IEAVNPE6 and are located in the SQA.

The MSFKB is used to serialize all read-restart-reason commands. The MSFCB is used to serialize all other requests except the TP connect and TP read commands. The MSFAB is used to serialize the TP connect and TP read commands.

Figure 5-52 shows an overview of the SVC control blocks.

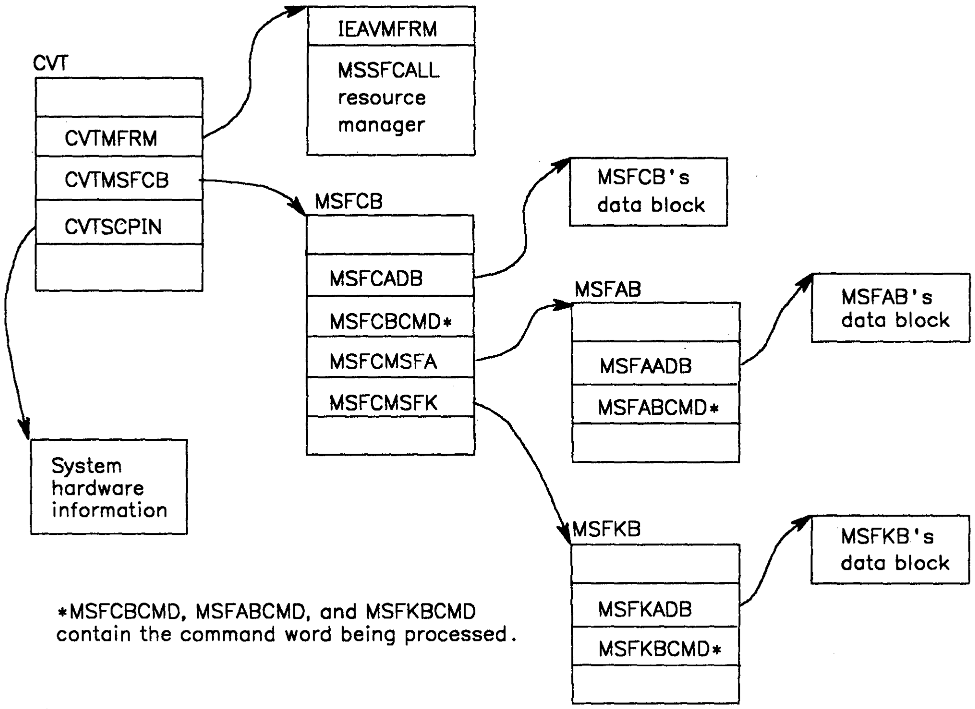


Figure 5-52. SVC Control Block Structure With the MSSF

MSSFCALL DIAGNOSE Instruction

The MSSFCALL DIAGNOSE instruction is a variation of the DIAGNOSE instruction (operation code X '83'). The MSSFCALL DIAGNOSE instruction directly interfaces with the MSSF in the processor controller. It is used to request hardware functions and to request hardware status.

The MSSFCALL DIAGNOSE instruction is issued by the Service Processor Call SVC processing routine (IEAVMSF) to process Service Processor Call SVC (SVC 122) requests. It is also issued by other system components (such as IPL, NIP, and SADMP) to obtain system and hardware information.

The format of the MSSFCALL DIAGNOSE instruction is:

X'83'	R1	R3	X'0080'
0-7	8-11	12-15	16-31

Bits 0-7 specifies the DIAGNOSE instruction (X'83').
R1 specifies the register containing the address of the MSSFCALL data block.
R3 specifies the register containing the MSSFCALL command word.
Bits 16-31 specifies the MSSFCALL instruction (which must be set to X'0080').

IEAVNIPO sets bit 22 of control register 0 (the class 21 external interruption mask bit) to 1 for each processor. This enables the system for service signal interruptions.

The MSSF indicates the completion of the MSSFCALL DIAGNOSE instruction by generating an external interruption. The interruption code is X'2401'.

MSSFCALL DIAGNOSE Instruction Condition Codes

At the completion of the MSSFCALL DIAGNOSE instruction, one of the following condition codes is set in the PSW:

Code	Meaning
0	The MSSF processed the request without error.
2	The MSSF is busy. (Note that IEAVMSF converts this condition code to a return code of 4, busy.)

Service Processor Call SVC and SERVICE CALL Instruction

Note: This topic describes the MVS support for processor complexes with the Service Processor Architecture. See the topic "Service Processor Call SVC and MSSFCALL DIAGNOSE Instruction" for a description of the MVS support for processor complexes with a monitoring and system support facility (MSSF).

The Service Processor Call SVC (SVC 122) is a type 2 extended SVC with a routing code of 6 in register 15. The SVC is used with processor complexes that have the Service Processor Architecture.

MVS system routines issue the Service Processor Call SVC to request various hardware functions and to obtain hardware status information.

When the Service Processor Call SVC is issued, the Service Processor Call SVC processing routine IEAVMSF processes the SVC request. IEAVMSF issues the SERVICE CALL instruction to directly interface with the Service Processor Architecture within the processor complex.

IPL module IEAVNIP0 sets bit CVTSVPRC on (in field CVTFLAG1 at X'178' in the CVT) to indicate to IEAVMSF (and other system modules) that the processor complex has the Service Processor Architecture.

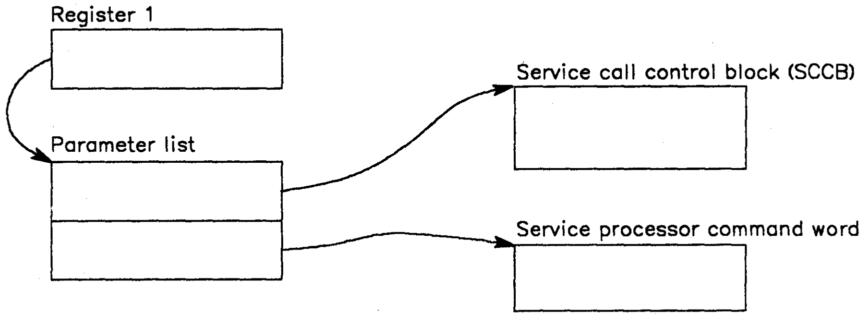
For program logic information, refer to the *System Logic Library*, *System Initialization Logic*, and *Input/Output Configuration Program (IOCP) Logic*.

The following topics briefly describe the:

- Service Processor Call SVC (SVC 122) used with the Service Processor Architecture
- SERVICE CALL instruction

Service Processor Call SVC (SVC 122) Used With the Service Processor Architecture

To issue the Service Processor Call SVC, the caller prepares a service call control block (SCCB), also called a service processor data block, and a service processor command word in the caller's own private area. The caller sets register 15 to 6 (the routing code) and sets register 1 to the address of a parameter list, which points to the SCCB and the command word.


Register 1
Parameter list
Service call control block (SCCB)
Service processor command word

The parameter list is a standard SVC parameter list. The SCCB contains command-dependent information and is used to send data to the service processor and to receive responses and data from the service processor. The command word specifies the command to be executed by the SERVICE CALL instruction.

Service Call Control Block (SCCB)

The SCCB (also called the service processor data block) is a variable-length data area. The format of the block is:

	0-1	2	3-5	6-7
	Length	Caller flags	Reserved	Response field
8	Data field (0 to 4,088 bytes)			

Length - bytes 0 and 1
specifies the length of the data block. The minimum length is eight (X'0008') and the maximum length is 4,096 (X'1000') bytes. The length varies depending on the command request.

Caller flags - byte 2
specifies command-dependent information.

Reserved - bytes 3 through 5
set to X'000000'.

Response field - bytes 6 and 7
receives the two-byte response from the service processor. The service processor sets this two-byte field to indicate the status of the request. The field must be set to X'0000' before issuing the Service Processor Call SVC.

Data field - byte 8 to end of data block
specifies command-dependent information, if any, related to the command.

SVC Abend and Return Codes With the Service Processor Architecture

Abend Code - The Service Processor Call SVC processing routine (IEAVMSF) issues system abend code X'67A' if the caller is not authorized to issue the SVC or if an error occurs during processing of the SVC. Refer to *System Codes* for a description of code X'67A'.

Return Codes - IEAVMSF returns the following codes in register 15 to the caller of the SVC.

Code	Meaning
00(00)	Successful completion - the SCCB contains the service processor response (in bytes 6 and 7) and any data (in the data field) requested on the command.
04(04)	The service processor is busy - the caller can reissue the SVC.
08(08)	One of the Service Processor Call SVC control blocks (MSFCB or MSFKB) is in use - the caller can reissue the SVC.
12(0C)	The service processor is not available or the control block chain is invalid.
16(10)	The service processor did not respond during a 10-second disabled spin.

SVC Processing With the Service Processor Architecture

The Service Processor Call SVC processing routine (IEAVMSF) provides the interface between system routines and the Service Processor Architecture. IEAVMSF processes requests made by programs issuing the Service Processor Call SVC and routines that branch-enter IEAVMSF.

IEAVMSF enqueues (using the ENQ macro) the system resource SYSMSF,MSFCONTROLBLOCK.

When processing the SVC request, IEAVMSF copies the caller's service call control block (SCCB) to the data block pointed to by the MSFCB or MSFKB control block. The service processor puts its responses and data into the data block pointed to by MSFCB or MSFKB. IEAVMSF then copies the response and data (if any) into the caller's SCCB.

Figure 5-53 shows an overview of IEAVMSF processing.

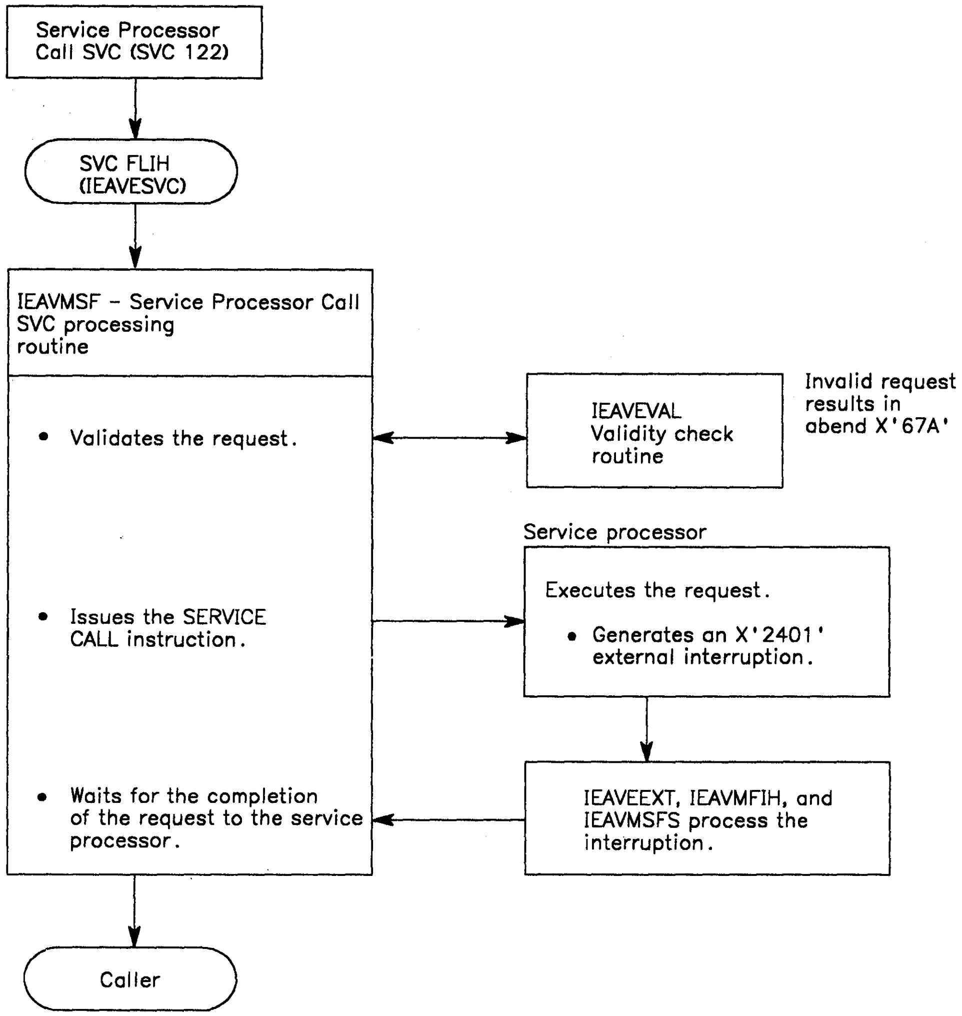
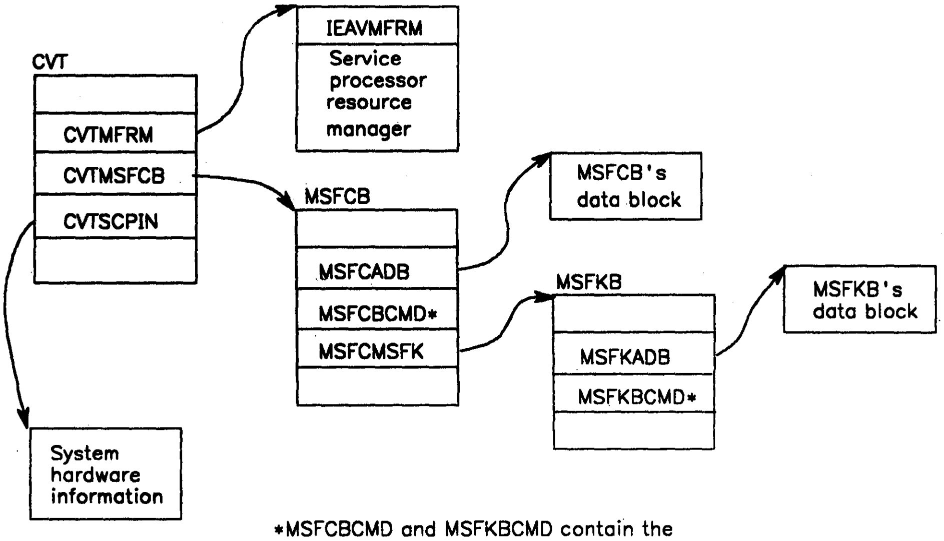

Service Processor Call SVC (SVC 122)
SVC FLIH (IEAVESVC)
IEAVMSF - Service Processor Call SVC processing routine
• Validates the request.
• Issues the SERVICE CALL instruction.
• Waits for the completion of the request to the service processor.
IEAVEVAL Validity check routine
Invalid request results in abend X'67A'
Service processor
Executes the request.
• Generates an X'2401' external interruption.
IEAVEEXT, IEAVMFIH, and IEAVMSFS process the interruption.
Caller

Figure 5-53. Overview of SVC Processing With the Service Processor Architecture

SVC Control Blocks Used With the Service Processor Architecture

IEAVMSF uses the control blocks MSFCB and MSFKB to serialize requests. MSFCB and MSFKB are created at NIP time by the RIM module IEAVNPE6 and are located in the SQA.

Figure 5-54 shows an overview of the Service Processor Call SVC control blocks.



*MSFCBCMD and MSFKBCMD contain the command word being processed.

Figure 5-54. SVC Control Block Structure With the Service Processor Architecture

SERVICE CALL Instruction

The SERVICE CALL instruction (operation code X'B220') interfaces with the Service Processor Architecture. It is used to request hardware functions and to request hardware status.

The SERVICE CALL instruction is issued by the SVC processing routine (IEAVMSF) when IEAVMSF is processing Service Processor Call SVC requests. The SERVICE CALL instruction is also issued by other system components (such as NIP, IPL, and SADMP) to obtain system and hardware information.

IEAVNIP0 sets bit 22 of control register 0 (the class 21 external interruption mask bit) to 1 for each processor. This enables the system for service signal interruptions.

The service processor indicates the completion of the SERVICE CALL instruction by generating an external interruption. The interruption code is X'2401'.

SERVICE CALL Instruction Condition Codes

At the completion of the SERVICE CALL instruction, one of the following condition codes is set in the PSW:

Code	Meaning
0	The service processor has initiated the request and a service-signal interruption will be presented when execution completes.
1	- -
2	The service processor is busy.
3	The service processor is not available.

Cross Memory Services

Cross memory services, a component of the MVS system control program, consists of the program call/authorization (PC/AUTH) services and the PCLINK STACK/UNSTACK/EXTRACT services. This chapter contains diagnostic information for both the PC/AUTH and the PCLINK services.

PC/AUTH Services

The PC/AUTH services create and manage the data structures that support the program call (PC) instruction and allow control of the cross memory authorization structure. These services are used by supervisor state or PSW key mask (PKM) 0-7 callers, usually subsystems, to set up the environment for controlling cross memory access to programs and/or data.

There are three instances, other than when the PC/AUTH address space is initialized, when PC/AUTH code is executed. The first is when an address space is created, the second is when a task or an address space is terminated, and the third is when one of the 11 PC/AUTH services is invoked.

All PC/AUTH code executes in the PC/AUTH address space (PASID=2) except the NIP RIM (IEAVNPF5) and the PC/AUTH address space initialization routine (IEAVXMIN).

When an address space is created, IEAVXMIN initializes its address space second table entry (ASTE) and chains the new address space to the system linkage table (SLT) and the system authorization table (SAT). The new address space is given an authorization index (AX) of 0, which makes it unauthorized to issue the PT or SSAR instruction to another address space. It has access only to those global PC services that are available to all address spaces via the SLT.

When a task or address space terminates, the PC/AUTH resource manager (IEAVXPAM) gets control. If a cross memory resource owning (CMRO) task or address space is terminating, IEAVXPAM recovers PC/AUTH-related resources.

The 11 PC/AUTH services are invoked via macro calls by supervisor state or PKM 0-7 callers. The following table lists the macro names with their descriptive names. The table is arranged in the same order as the entries in the system function table (SFT).

Macro	Descriptive Name
LXRES	Linkage index (LX) reserve
LXFRE	Linkage index (LX) free
ETCRE	Entry table (ET) create
ETDES	Entry table (ET) destroy
ETCON	Entry table (ET) connect
ETDIS	Entry table (ET) disconnect
AXRES	Authorization index (AX) reserve
AXFRE	Authorization index (AX) free
AXEXT	Authorization index (AX) extract
AXSET	Authorization index (AX) set
ATSET	Authorization table (AT) set

For a complete description of these macros, refer to *OS/VS2 System Programming Library: Supervisor.* For a description of the logic of the PC/AUTH services, refer to *OS/VS2 System Logic Library* and *OS/VS2 System Initialization Logic.*

For PC/AUTH services, this topic contains the following:

- Module structure of the PC/Auth modules
- Process flow of PC/AUTH processing
- Control block structure of PC/AUTH control blocks
- Recovery considerations for PC/AUTH services
- Debugging hints for the PC/AUTH services

Module Structure

All PC/AUTH service routines and their FRR are contained in the load module IEAVXPCA, which is loaded into the PC/AUTH address space from SYS1.LINKLIB by the PC/AUTH address space initialization module IEAVXMAS. IEAVXMAS is loaded from SYS1.LINKLIB into the PC/AUTH address space by the PC/AUTH NIP RIM, IEAVNPF5. IEAVXMAS is given control during NIP via a LINK macro from IEEPRWI2.

The PC/AUTH service routines in load module IEAVXPCA are ordered in the same sequence as the entries in the system function table (SFT) for their corresponding services. The following table lists the modules in load module IEAVXPCA. The load module has an alias associated with each entry point that receives control via a PC instruction. The alias enables IEAVXMAS to issue a LOAD macro for each of these entry points to obtain the entry point address, which is stored in the entry table entry (ETE) associated with the service. The FRR (IEAVXPCR) is not loaded and therefore does not require an alias. Also shown in the table are the associated system function table (SFT) index value, the service-in-control code (PCRASERV) value, and the entry table index (EX) value. Note that when a PC/AUTH service routine abends, the service-in-control code is the high-order byte of the halfword reason code.

Module Name	Entry Points	SFT Index	PCRA-SERV	EX Value	Service/Function
IEAVXLRE	IEAVXLRE	1	01	0	LXRES (LX reserve)
IEAVXLFR	IEAVXLFR	2	02	1	LXFRE (LX free)
IEAVXECR	IEAVXECR	3	03	2	ETCRE (ET create)
IEAVXEDE	IEAVXEDE	4	04	3	ETDES (ET destroy)
IEAVXECO	IEAVXECO	5	05	4	ETCON (ET connect)
IEAVXEDI	IEAVXEDI	6	06	5	ETDIS (ET disconnect)
IEAVXRFE	IEAVXARE	7	07	6	AXRES (AX reserve)
	IEAVXAFR	8	08	7	AXFRE (AX free)
	IEAVXAEX	9	09	8	AXEXT (AX extract)
IEAVXSET	IEAVXAXS	10	0A	9	AXSET (AX set)
	IEAVXATS	11	0B	A	ATSET (AT set)
	IEAVXACK	-	-	-	Authorization check
IEAVXPAM	IEAVXPAM	12	0C	B	PC/AUTH resource manager
IEAVXPCR	IEAVXPCR	-	0D	-	PC/AUTH FRR

There are three modules in the nucleus that are related to the PC/AUTH services. They are: .

IEAVXSFM - a nonexecutable module containing the system function table (IEAVXSFT). It provides PC numbers for system services and pre-assigns linkage indexes (LXs) as system LXs. When a system LX is connected to an entry table, all address spaces are then connected to that system entry table.

IEAVXNEP - a nonexecutable module containing the nucleus entry point table (IEAVXNET). It contains pairs of names and virtual addresses used to locate nucleus routines that can be invoked by a PC instruction.

IEAVXEPM - a service module used by the entry table create service (IEAVXECR) to find the location of requested nucleus routines. It is also used by the PC/AUTH address space initialization module (IEAVXMAS) to find the location of the 0D6 abend routine (IEAVXABE, located in IEAVXEPM).

The PC/AUTH address space initialization routine (IEAVXMIN) is in the link pack area:

IEAVXMIN - initializes the address space second table entry (ASTE) of a new address space, and chains the new address space to the system linkage table (SLT) and the system authorization table (SAT). IEAVXMIN is contained in the load module IEAVEMCR.

Process Flow

When one of the PC/AUTH services is invoked by a macro call from a system routine, the macro-generated code saves the caller's registers in a standard save area, sets up the necessary parameters, and then issues the PC instruction. When the service completes its function, it returns to the caller by issuing the PT instruction. The macro-generated code then restores the caller's registers (except those containing output values).

Each PC/AUTH service routine performs a common sequence of operations. Upon entry, each routine saves the PC instruction information by using the PCLINK STACK service. The routine then serializes its operation with other services by obtaining the PC/AUTH address space's local lock. It then sets up two levels of FRR for recovery, and obtains (via a GETMAIN) its dynamic work area from the private area of the PC/AUTH address space.

At this point, each service routine performs its own service function.

Upon successful completion, the service routine frees (via a FREEMAIN) its dynamic work area, deletes its FRR coverage, and releases PC/AUTH address space's local lock. The service then restores the information necessary to issue the PT instruction to the caller by invoking the PCLINK UNSTACK,THRU service, and then issues the PT instruction.

Control Block Structure

Most control blocks and programs related to the PC/AUTH services are located in the LSQA, or pageable private area of the PC/AUTH address space.

LSQA

Because they must reside in fixed real storage, the control blocks that are accessed by the cross memory instructions reside in the LSQA (subpool 255) of the PC/AUTH address space. These control blocks include:

LT - Linkage tables (LTs) consisting of linkage table entries (LTEs), mapped by IHALTE.

SLT - System linkage table.

AT - Authorization tables (ATs) with entries mapped by IHAATD.

SAT - System authorization table.

ET - Entry tables (ETs) consisting of entry table entries (ETEs), mapped by IHAETE.

LPA - PC/AUTH's latent parameter areas (LPAs) associated with PC/AUTH's ETEs. (Used only to find the PCLINK stack pool header.)

Pageable Private Area

The control blocks that are needed only by the PC/AUTH service routines reside in the pageable private area (subpool 229) of the PC/AUTH address space. These control blocks include:

XMD - Cross memory directory, mapped by IHAXMD.

LXAT - Linkage index allocation table, mapped by IHALXAT.

AXAT - Authorization index allocation table, mapped by IHAAXAT.

ETIB/ETIX - Entry table information block (ETIB) and ETIB extension (ETIX), mapped by IHAETIB.

The formats of these control blocks are described later in this chapter.

Subpool 229 is also used for the dynamic data area storage for each of the PC/AUTH service routines and their FRR, and for all temporary storage needed by the services (such as the force disconnect queue blocks (FDQBs) used by the LXFRE service for the FORCE option).

SQA

The doubleword latent parameter area (LPA) associated with each entry table entry (ETE) is located in the SQA (subpool 245) for all entry tables (ETs) except PC/AUTH's LPAs, which are located in the LSQA (subpool 255) of the PC/AUTH address space.

Figure 5-55 shows an overview of the control block structure related to the PC/AUTH services. More detailed control block diagrams are shown in *OS/VS2 System Logic Library* (in the topic "PC/AUTH Services"), and in *OS/VS2 System Programming Library: Debugging Handbook.*

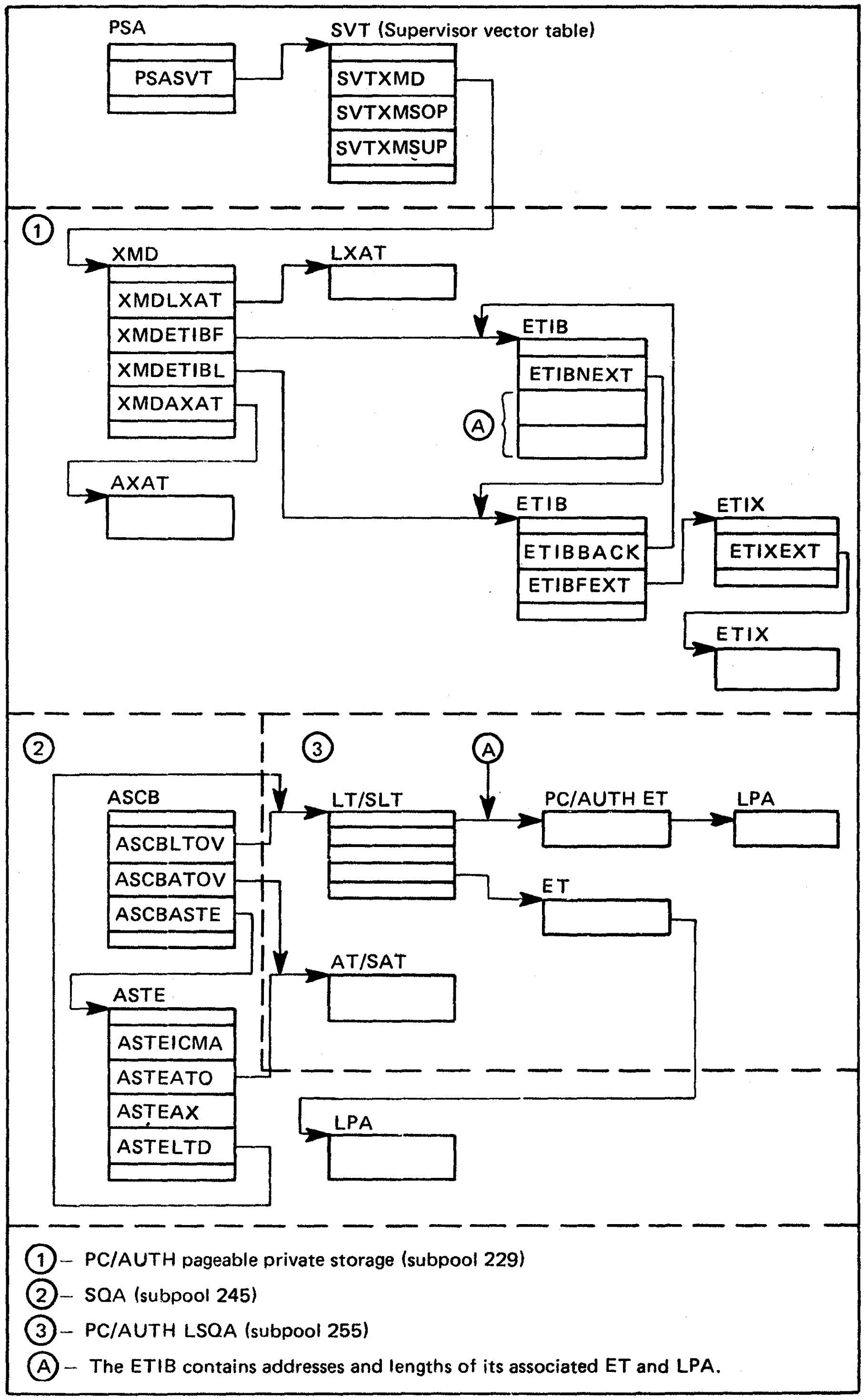


Figure 5-55. PC/AUTH Control Block Structure

Control Block Formats

The format of the PC/AUTH control blocks (XMD, LXAT, AXAT, and ETIB/ETIX) are described in this topic. Also, descriptions of key fields and indicators in selected system control blocks (ASCB, ASTE, and SVT) are provided.

XMD Control Block

The cross memory directory (XMD) anchors the other PC/AUTH control blocks. Also, IEAVXMIN uses values from the XMD to initialize the AT and LT pointers for a new address space.

Offset	Field	Contents
X'00'	XMDXMD	'XMD' acronym.
X'04'	XMDLXAT	Address of the LXAT.
X'08'	XMDETIBF	Address of the first ETIB on the queue.
X'0C'	XMDETIBL	Address of the last ETIB on the queue.
X'10'	XMDAXAT	Address of the AXAT.
X'14'	XMDFLGS	Reserved.
X'15'	XMDRSV9	Reserved.
X'18'	XMDRSV10	Reserved.
X'1C'	XMDATLNB	Length, in bytes, of the SAT.
X'1E'	XMDSATLN	Bits 0-11 contain the number of words, minus one, in the SAT. Bits 12-15 are zero. Used to initialize an ASTE.
X'20'	XMDSATOR	Real address of the SAT in a format to initialize an ASTE.
X'24'	XMDSATOV	Virtual address of the SAT.
X'28'	XMDSLTD	Real address of the SLT (with the valid bit on) in an ASTE format.
X'2C'	XMDSLT	Virtual address of the SLT.

AXAT Control Block

The authorization index allocation table (AXAT) contains authorization index (AX) ownership information.

Offset	Field	Contents
X'00'	AXATNAME	'AXAT' acronym.
X'04'	AXATCT	Count of entries in the AXAT.
X'08'	AXATAVAL	Number of entries currently available (unreserved).
X'0C'	AXATRSVD	Reserved.
X'10'	AXATENT	First two-byte AXAT entry.
	AXATENTY	(Alternate name for the AXAT entry.)

A zero entry indicates that an AX is unreserved. If an entry is nonzero, it contains the ASID that has reserved (owns) the corresponding AX. The first entry in the table is the AX=0 entry, the second is the AX=1 entry, and so on. The first two AXs are pre-assigned and owned by the PC/AUTH ASID. Therefore, the first two entries always contain the PC/AUTH ASID, which is 2. The following shows an AXAT as it might appear in a dump.

```
'AXAT'     CT    AVAL   AX=0   AX=1   AX=2   AX=3   AX=4   ...
C1E7C1E3   0080  003C   0002   0002   0009   0013   0000   ...
```

LXAT Control Block

The linkage index allocation table (LXAT) contains linkage index (LX) ownership information.

Offset	Field	Contents
X‘00’	LXATHDR	LXAT header.
X‘00’	LXATLXAT	‘LXAT’ acronym.
X‘04’	LXATHILX	Highest LX contained in the LXAT.
X‘06’	LXATMSLX	Maximum system LX in the LXAT.
X‘08’	LXATINDX	Linkage index status array -

Each entry in the array contains:

X‘00’	LXATASID	The ASID that owns this index. (Valid only if LXATOWND is on.)
X‘02’	LXATBIND	The bind count - count of address spaces using this LX. For a system LX that was ever connected, the field is X‘FFFF’.
X‘04’	LXATETCT	The entry table connect count - count of ETs connected to this LX. For a system LX that is connected, the field is X‘FFFF’.
X‘06’	LXATFLGS	Flag byte:
1...	LXATRIP	An LXRES macro is in process for this LX.
.1..	LXATOWND	This LX is reserved - owned by the address space whose ASID is in LXATASID.
..1.	LXATSYS	This is a system LX.
...1	LXATDORM	This system LX is dormant. A dormant LX is a system LX whose owning address space has terminated. A restartable subsystem that restarts in a new address space can reconnect to this LX.
.... 1111	LXATRSV1	Reserved.
X‘07’	LXATRSV2	Reserved.

The LXAT is divided into two sections. All the LXs up to the LX number in the LXATMSLX field are permanent system LXs. This number can be altered by zapping the SVTMSLX field prior to IPL or during IPL but before PC/AUTH initialization; it cannot be altered after IPL. The remainder of the LXs are non-system LXs. For an LX of either type to be available for assignment by the LXRES macro, the LXATOWND bit must be off and the LXATBIND and LBXATETCT fields must both be zero. The following shows an LXAT as it might appear in a dump:

```
        ‘LXAT’          HILX    MSLX
        D3E7    C1E3    0020    0010
        ASID    BIND    ETCT    FLGS
LX=0:   0002    FFFF    FFFF    6000    Program call/authorization
                                        (PC/AUTH)
        ASID    BIND    ETCT    FLGS
LX=1:   0003    FFFF    FFFF    6000    Global resource serialization
                                        (GRS)
        ASID    BIND    ETCT    FLGS
LX=2:   0004    FFFF    FFFF    6000    Allocation address space
                                        (ALLOCAS)
                :
                :
        ASID    BIND    ETCT    FLGS
LX=10:  0000    0000    0000    2000    Available system LX
        ASID    BIND    ETCT    FLGS
LX=11:  0000    0000    0000    0000    Available non-system LX
                :
                :
```

ETIB and ETIX Control Blocks

The entry table information blocks (ETIBs) contain entry table ownership and connection information. They are arranged in a double-threaded queue. For non-system connections, connection information is kept in ETIB extension blocks (ETIXs), which are queued off the ETIB. ETIXs are arranged in a single-threaded queue.

The format of the ETIB is:

Offset	Field	Contents
X'00'	ETIBETIB	'ETIB' acronym.
X'04'	ETIBASCB	Address of the ASCB that owns the entry table (ET).
X'08'	ETIBNEXT	Forward link for the ETIB queue.
X'0C'	ETIBBACK	Backward link for the ETIB queue.
X'10'	ETIBETR	Real address of the associated ET. Bits 26-31 contain an indicator of the ET length (the number of ETEs/4 - 1).
X'14'	ETIBETV	Virtual address of the associated ET.
X'18'	ETIBLPAD	Address of the latent parameter area (LPA) for the ET.
X'1C'	ETIBLPLN	Length of the latent parameter area (LPA).
X'20'	ETIBRSV1	Reserved.
X'24'	ETIBRSV2	Reserved.
X'28'	ETIBRSV3	Reserved.
X'29'	ETIBFLGS	Flag byte:
	1...	ETIBSYS This entry table is a system ET. (It is connected to a system LX and is available to all address spaces.)
	.1..	ETIBSS This ET has space switch entries. (That is, an address space switch occurs when the program described by the ETE is invoked via the PC instruction.
	..1.	ETIBCIL Connection information has been lost.
	...1 1111	Reserved.
X'2A'	ETIBCNCT	Count of connections to this entry table (ET). (For a system ET, this value is X'FFFF'.
X'2C'	ETIBFEXT	Address of first ETIX. This field is zero if there are no connections. It is not used for system ETIBs (ETIBSYS = 1).

The format of the ETIX is:

Offset	Field	Contents
X'00'	ETIXETIX	'ETIX' acronym.
X'04'	ETIXRESV	Reserved.
X'08'	ETIXSLOT	Count of connection slots in the ETIX (X'A').
X'0A'	ETIXFREE	Free slot count.
X'0C'	ETIXEXT	Pointer to next ETIX.
	This field is followed by an array of 10 connection slots that show which address spaces are connected to the ET and which LX the ET is connected to.	
X'00'	ETIXASID	ASID of the address space connected to this ET.
X'02'	ETIXLX	Linkage index (LX) at which this ET is connected in the linkage table of the above ASID. (This value is the 12-bit LX value, right justified - not the LX value as it is returned by the LXRES service.)

The following shows an ETIX as it might appear in a dump:

```
'ETIX'     reserved   SLOT/FREE  next ETIX
C5E3C9E7   00000000   000A0007   00000000

SLOT1:     SLOT2:     SLOT3:     SLOT4:     SLOT 5:    ... SLOT 10:
ASID/LX    ASID/LX    ASID/LX    ASID/LX    ASID/LX    ... ASID/LX
00140005   000E0005   00000000   002C0005   00000000   ... 00000000
                      (free)                (free)         (free)
```

Note: Used slots are not necessarily contiguous. Slot 3 is free, which indicates that an address space was once connected and has been disconnected.

Key Fields and Indicators in System Control Blocks

The following fields and indicators in the ASCB, ASTE, and SVT (which are not in the PC/AUTH address space) contain information related to the PC/AUTH services.

ASCBETC - This field indicates the number of entry tables (ETs) currently owned by the address space. This count should always be equal to or greater than the number actually owned, which is indicated in the ETIB queue. The PC/AUTH resource manager (IEAVXPAM) keys on this value to determine if any ETs owned by an address space must be cleaned up.

ASCBETCN - The field indicates the number of connections to entry tables (ETs), owned by this address space that contain space switch entries. This count should always be equal to or greater than the number of connections that actually exist. A nonzero value in this field indicates that the authorization index (AX) of the address space cannot be changed.

ASCBLXR - This field indicates the number of linkage indexes (LXs) reserved by this address space. This count should always be equal to or greater than the number of LXs reserved as indicated in the LXAT. The PC/AUTH resource manager (IEAVXPAM) keys on this value to determine if any LXs must be freed for an address space.

ASCBAXR - This field indicates the number of authorization indexes (AXs) reserved by this address space. This count should always be equal to or greater than the number of AXs as indicated in the AXAT. The PC/AUTH resource manager (IEAVXPAM) keys on this value to determine if any AXs must be freed for an address space.

ASCBXMET - This bit indicates that either space switch entry tables are owned by the address space, or incomplete termination processing of PC/AUTH resources for the address space has prevented reuse of the ASID.

ASCBXMEC - This bit indicates that one or more entry tables containing space switch entries were created by this address space.

ASTEICMA - This bit (high-order bit in field ASTEATO) indicates that the address space associated with this ASTE is not available for cross memory access.

ASTELTV - This bit (high-order bit in field ASTELTD) indicates that the linkage table entry (LTE) is valid.

SVTXMSOP - This bit indicates that the PC/AUTH services are operational.

SVTXMSUP - This bit indicates that the PC/AUTH services have been initialized.

Recovery Considerations

PC/AUTH recovery is designed to recover from nonrecursive errors that do not cause permanent damage to critical PC/AUTH data structures. All of the PC/AUTH service routines and the resource manager are covered by a common FRR routine, IEAVXPCR. The FRR contains a validation routine that attempts to detect damage to PC/AUTH's pageable control blocks. If the FRR determines that damage has occurred, it disables further execution of PC/AUTH services by turning off the SVTXMSOP (operational) bit in the SVT. This prevents PC/AUTH from continuing execution with known defective data. The FRR does not contain logic to repair or reinitialize PC/AUTH if its control blocks are damaged, with the exception of the ETIB/ETIX structure. The ETIB/ETIX queues can be repaired by the queue verification routines, isolating the problem to a subset of users while maintaining system integrity. PC/AUTH mainline service routines always check that PC/AUTH is operational (SVTXMSOP = 1). They assume, if PC/AUTH is operational, that PC/AUTH's control blocks are valid if their acronyms are valid.

The FRR performs those recovery functions that are common to all the service routines. This includes recording data in SYS1.LOGREC, taking an SVC dump of PC/AUTH data at the time of the error, and validating the PC/AUTH control blocks. If a service requests a retry, a retry of the function is performed. If retry is not performed, a cleanup exit (in the service routine module) might be given control to clean up resources specific to the service. The FRR then cleans up the service routine's dynamic storage and PCLINK stack entry, and percolates to the caller with an X'053' abend.

For a detailed description of the interfaces between the service routines and the FRR, refer to the program listing for module IEAVXPCR. Figure 5-56 shows the relationship of the key PC/AUTH recovery data structures. The Program Call recovery area (PCRA) is mapped by the IHAPCRA mapping macro. The fixed portion of the service routine recovery area (SRRA) isa mapped by the IHASRRA mapping macro. The formats of the SRRA, PCRA, and PCRA flag field is provided in the following topics.

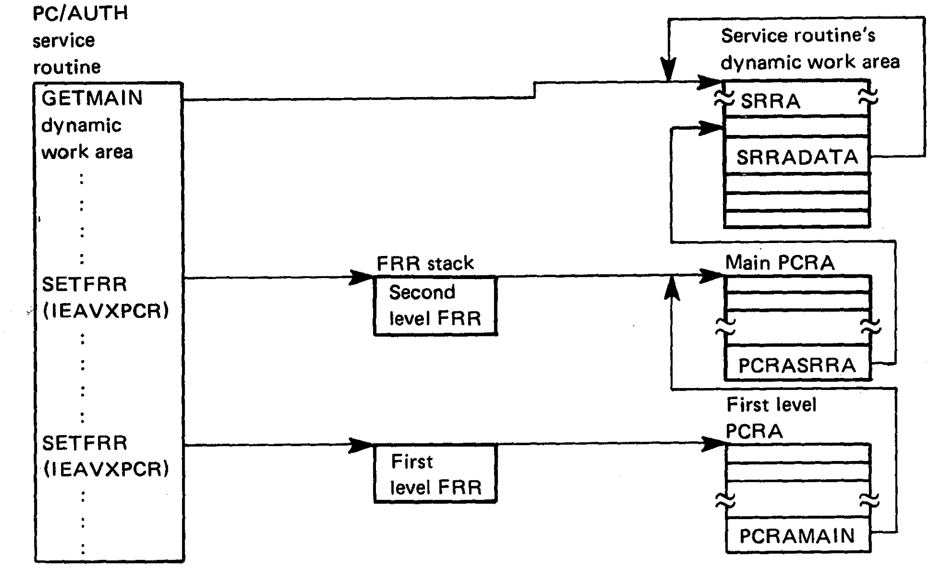


Figure 5-56. PC/AUTH Recovery Areas

SRRA Control Block

The format of the service routine recovery area (SRRA) is:

Offset	Field	Contents
X'00'	SRRANAME	'SRRA' acronym.
X'04'	SRRADATA	Address of the service routine's dynamic work area.
X'08'	SRRADLEN	Length of the work area.
X'0A'	SRRASLEN	Length of the SRRA.
X'0C'	SRRABASE	Service routine base register.
X'10'	SRRABAS2	Second base register.
X'14'	SRRARREG	Address of the retry registers (0-15).
X'18'	SRRARTY@	Address of retry routine.
X'1C'	SRRASUML	Optional SUMLSTA list.
X'20'	SRRAESAR	Secondary ASID.
X'24'	SRRAHOME	Address of home ASCB.
X'28'	SRRAMLIA	Address of module level information.
X'2C'	--	Key variables: These variables are unique for each service routine. Refer to the program listings of the service routines for a description of these variables.

Main PCRA Control Block

The format of the main Program Call recovery area (PCRA) is:

Offset	Field	Contents
X'00'	PCRASERV	Service-in-control code.
X'01'	PCRAREAS	Abend reason code.
X'02'	--	PCRA flags: Refer to the following topic "PCRA Flags" for a description of these bits. For the main PCRA, PCRA2ND = 1.
X'04'	PCRASTTK	PCLINK token value.
X'0C'	PCRAFOOT	FRR footprint data. (See Note 1.)
X'10'	PCRARRDA	Address of the FRR dynamic work area. (See Note 2.)
X'14'	PCRASRRA	Address of the SRRA.

Notes:

1. *The PCRAFOOT field contains a value that indicates which section of the FRR (IEAVXPCR) is processing. These footprints are useful in debugging when the FRR encounters an error. The functional meaning of these footprints is:*

Value in PCRAFOOT	FRR Label	Function
X'00'	IEAVXPCR	Preliminary processing, SYS1.LOGREC preparation, and recursion checking.
X'01'	POINT1	Issue the SDUMP macro.
X'02'	POINT2	Validate PC/AUTH's control blocks.
X'03'	POINT3	Retry, if requested.
X'04'	POINT4	Call cleanup exit, if requested.
X'05'	POINT5	Free the service routine's dynamic work area.
X'06'	POINT6	Cleanup the FRR's dynamic work area.
X'07'	POINT7	Issue PCLINK UNSTACK and percolate the error.

2. *The value in PCRARRDA does not point to the RRDA structure defined in IEAVXPCR, it points to the beginning of the FRR's dynamic work area.*

First Level PCRA Control Block

Offset	Field	Contents
X'00'	PCRASERV	Zero.
X'01'	PCRAREAS	Zero.
X'02'	PCRA	PCRA flags: Refer to the following topic "PCRA Flags" for a description of these bits. For the first level PCRA, PCRA1ST = 1.
X'04'	PCRASTTK	Zero.
X'0C'	PCRAFOOT	Zero.
X'10'	PCRARRDA	Zero.
X'14'	PCRAMAIN	Address of the main PCRA.

PCRA Flags

The contents of the PCRA flag bytes at offsets X'02' and X'03' are:

Offset	Field	Contents
X'02'	--	Flag byte:
1...	PCRARSB1	Reserved.
.1..	PCRACML	The PC/AUTH address space local lock is held.
..1.	PCRACMS	The cross memory services lock is held.
...1	PCRAKCML	The caller of the service now running already held the PC/AUTH address space's local lock, and therefore it should not be released if recovery percolates to RTM. (This occurs when the ETDIS service is called by either the LXFRE macro with the FORCE option, or by the ETDES amcro with the PURGE option.)
.... 1...	PCRACLUP	Cleanup exit invocation is requested.
.... .1..	PCRARCUR	Retry recursion indicator. If this bit is on when the FRR is entered after a retry, then the retry is recursive aud a retry request is not honored. The bit is set on by the FRR when it retries to a service. The service must clear the bit (set it to 0) after setting a new retry address if the service wants further retry capability.
.... ..1.	PCRAFRRE	IEAVXPCR has been entered as an FRR.
....1	PCRARMGR	IEAVXPCR has been entered as a CML (cross memory lock) resource manager.
X'03'	--	Flag byte:
1...	PCRA1ST	This PCRA is associated with the first level FRR. If this bit is on, the sixth word of the PCRA (PCRAMAIN) contains the address of the main PCRA.
.1..	PCRA2ND	This is the main PCRA, associated with the second level FRR. If this bit is on, the sixth word of the PCRA (PCRASRRA) contains the address of the SRRA.
..1.	PCRANTH	This PCRA is associated with a nested (or Nth level) FRR, which is set by the second (or another Nth) level FRR to protect itself. If this bit is on, the sixth word of the PCRA (PCRAMAIN) contains the address of the main PCRA.
...1	PCRAPERC	Percolate to the caller. This bit is set on by the FRR to indicate to any higher level FRR that recovery is complete and it should percolate to RTM.
.... 1...	PCRAREC2	Recursion in the FRR. This bit is set on to indicate that the FRR has detected a recursive error in its processing.
.... .1..	PCRAFRRG	An FRR GETMAIN is in progress.
.... ..1.	PCRADUMP	An SDUMP macro has been issued during this recovery process.
....1	PCRARSB2	Reserved.

Debugging Hints

1. The RRDA structure defined in IEAVXPCR contains many flags that are useful in debugging problems in the FRR, including flags that footprint the progress through the control block validation process. If the validation routine takes a recursive program check during its processing, the PC/AUTH services are made inoperable. Because an SDUMP macro is issued only once during the recovery process, the values in the RRDA cannot be obtained from a dump when errors occur in the FRR. Each level of the FRR that executes, however, records the RRDEBUG portion of the RRDA in the SYS1.LOGREC variable recording area. For a mapping of the RRDA, refer to the program listing for module IEAVXPCR.

2. For SVC dumps, the dump title indicates which service had the problem. It also provides the reason code and diagnostic value (described in *System Codes*) for X'053' abends, or the system completion code and register 14 contents for other failures. The SUMDUMP portion of the dump provides a synchronous dump of the PC/AUTH data at the time of the failure and

before any retry or cleanup action has been taken by the FRR. The first SUMLSTA record contains the SUMLSTA list of storage ranges to be synchronously dumped. The second SUMLSTA record is the IEAVXPCA load module containing the service routines, resource manager, and FRR as they appear at the time of the failure. These two records are followed by SUMLSTA records that contain the PC/AUTH control blocks as they existed at the time of the failure.

3. The main PCRA contains the pointer to the SRRA in the executing service's working storage; it also contains many flags of value in determining what is going on, especially if the error occurs during recovery, retry, or cleanup processing. The main PCRA at the time the suspended summary dump was taken can be found by looking in the SUMDUMP portion of the dump. It is the work area portion of the first FRR on the formatted FRR stack from the interrupt handler save area (IHSA). The same PCRA can also be found in the variable recording area of the SDWA record in the SUMDUMP section of the dump but it is more difficult to read because it might not start on a word boundary. Be sure to use the PCRA found in the SUMDUMP output and not one found in asynchronous portions of the dump because the latter does not necessarily relate to the error.

4. The SYS1.LOGREC records written during recovery processing provide a large amount of data relevant to the problem. The variable recording area (VRA) in the SDWA is fully used. The data in the VRA is in the key-length-data format where the key fields are defined by the IHAVRA macro. The data includes the main PCRA at the time of entry to the FRR, possible diagnostic messages from the FRR's validation routine, possible diagnostics from the queue verification routines, and as much of the service routine's SRRA as possible. Each service routine declares variables that are of value in debugging within the SRRA structure, with the most useful near the beginning of the SRRA, thus enhancing the diagnostic value of the SYS1.LOGREC record. The SYS1.LOGREC entry contains the registers at the time of the failure except that for X'053' abends with a X'xx97' (unexpected error) reason code, the original value of register 15 is in the VRA (VRAKEY=VRAOR15) and the original system completion code is also in the VRA (VRAKEY=VRAOA).

5. In stand-alone dumps, if a PC/AUTH service routine was running when the system failed, two PCRAs should be found in the PSA. Look for a six-word FRR parameter area with a pointer in the sixth word pointing to another FRR parameter area. The second (main) PCRA has a PCLINK stack token value in the second word. The PCLINK token value appears as two small halfword values, the first is the number of levels of Program Calls to the currently executing service (usually 0001), and the second is the PC/AUTH ASID. The sixth word is a pointer to the SRRA for the executing service routine in the PC/AUTH address space. The first word of the SRRA contains the SRRA acronym. The fourth word contains the first or only base register value of the service routine that was executing. This value is close to the beginning of the code for the service that was executing.

The following shows how FRR parameter areas might appear in a dump.

```
      FRR ENTRIES
... PARMAREA 05004242  00010002  00000000  01000000  007D21E8  007D24EC
... PARMAREA 00000080  00000000  00000000  00000000  00000000  00000D08
... PARMAREA 80000000  50027A1E  007D2230  00000000  00000000  0002883C
... PARMAREA 60000000  50E42032  50E42032  00A4F2B8  00E44368  00A4F4F0
```

6. If the problem is an X'052' abend that should not be occurring, the reason code indicates what part of the data structure to look at if the caller's input is not in error. To look at the PC/AUTH data structures, use the DUMP command or a SLIP trap with ACTION=SVCD to dump the PC/AUTH address space. For example, if a caller tries to disconnect an ET that the caller had connected and receives an X'052' abend with an X'0613' reason code (indicating the specified ET is not connected), then the debugger should look at the ETIB queue. It is possible in this example that the ETIX queue had been truncated due to an error. If this had occurred, the ETIBCIL (connection information lost) bit would be set on in the ETIB associated with the ET. If no ETIB can be found that is associated with the ET, then the ETIB might have been removed from the queue due to an error. If either of these queue repair operations has occurred, there should be an X'053' abend dump and a SYS1.LOGREC record written with queue verification diagnostics in its VRA indicating why the element was removed. Note that an ETIB associated with a connection or disconnection operation can be readily found in the PC/AUTH address space. This is because the token value received from the ETCRE service and passed to the ETCON, ETDIS, or ETDES service is the address of the ETIB. For this ETDIS service X'052' abend described in this example, the token value related to the error is supplied in register 2.

SLIP Traps

If a program check is occurring in a PC/AUTH service routine or in the FRR that cannot be solved with the information provided in a dump or the SYS1.LOGREC records, set a SLIP trap similar to the following to stop the system at the point of error.

```
SLIP SET,ID=ERR2,ASID=2,ERRTYP=ALL,A=WAIT,END
```

This trap matches on all errors that occur in the PC/AUTH address space. Only the PC/AUTH service routines, resource manager, and FRR execute in the PC/AUTH address space. To determine the PC/AUTH ASID, issue a 'D A,ALL' operator command.

If the error is an erroneous X'052' abend, a SLIP trap with COMP=052 and a comparison of the reason code in register 15 can be used to detect a specific X'052' abend. Further selectivity can be achieved in some cases by checking for a specific diagnostic register value. For example, the following trap checks for an AXRES request with an invalid request count of 0, and provides an SVC dump of both the current (calling) address space and the PC/AUTH address space (ASID=2).

```
SLIP SET,ID=C052,COMP=052,ACTION=SVCD,ASIDLST=(0,2),
     DATA=(15R,EQ,00000703,4R,EQ,00000000),ML=1,END
```

PCLINK Services

The PCLINK service routines provide services for users of the Program Call (PC) and Program Transfer (PT) instructions by saving key linkage information in queues of PCLINK stack elements (STKEs). There is a separate queue for each RB or SRB. A program that receives control via a PC instruction should issue the PCLINK STACK macro prior to updating registers 2, 3, 4, 13, and 14 (and 0, 1, and 15 if they are parameter registers). A PCLINK stack token value is returned in register 14. The called program can then perform its processing, which might result in other PC instructions being issued.

When the called program is about to return to its caller, it should load registers 0, 1, and 15 with any data to be passed back to the caller. It should then issue the PCLINK UNSTACK,THRU macro and provide the token value received on the PCLINK STACK macro. This restores registers 3, 13, and 14, and, optionally, the original PSW protect key. The called program can then issue the PT instruction to return to the caller.

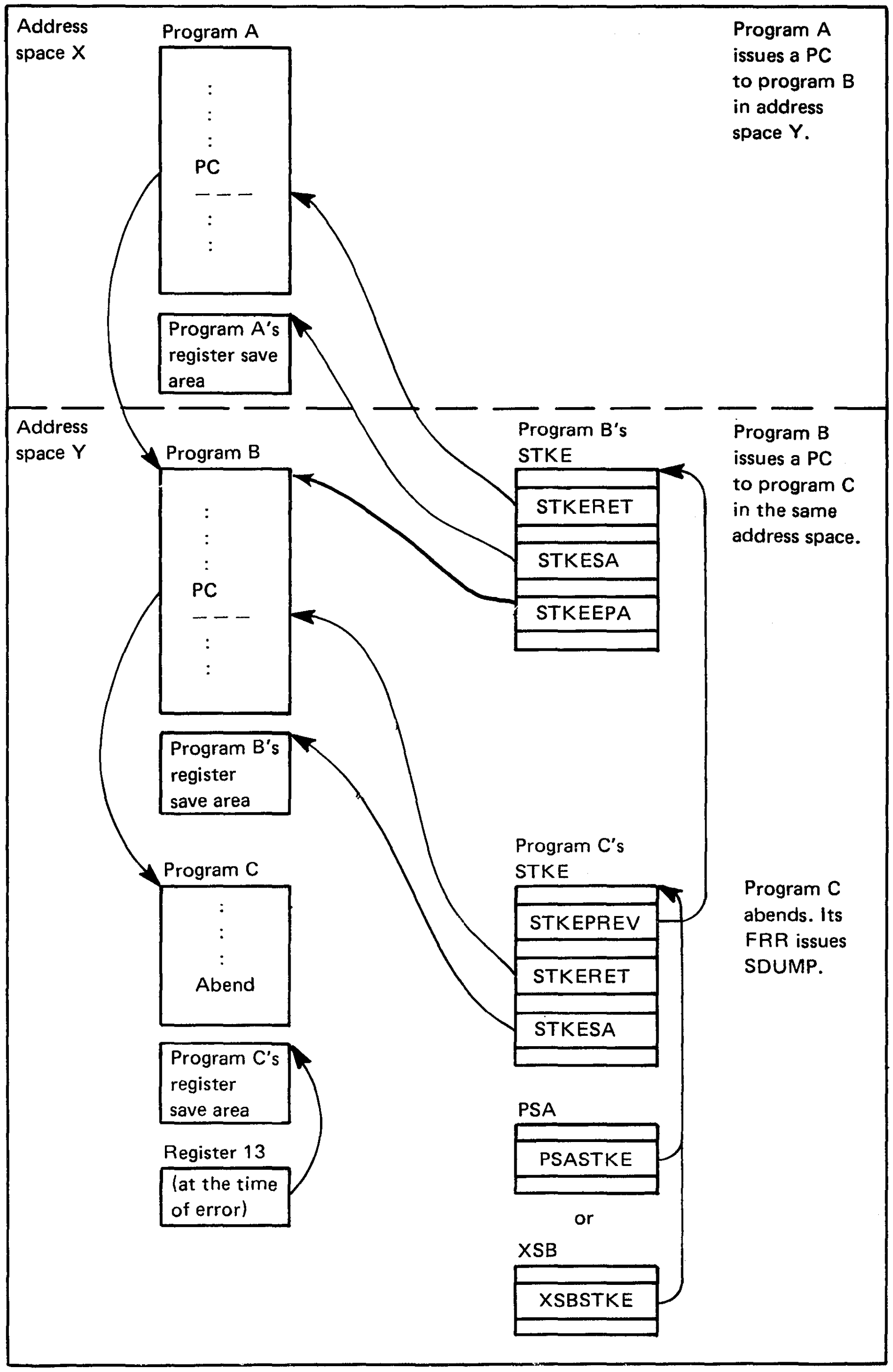
For a description of the PCLINK macro and services, refer to *OS/VS2 System Programming Library: Supervisor*. For a description of the logic of the PCLINK services, refer to *OS/VS2 System Logic Library*.

STKE Control Block

Figure 5-57 shows the control block structure of the PCLINK stack elements (STKEs) when a program has passed control to another program (via a PC instruction), and when a program that has received control (via a PC instruction) abends.

Important fields in the STKE are:

Offset	Length	Name	Contents
X'C'	4	STKEPREV	Contains either the address of the prior STKE, or the address of the next free STKE.
X'14'	4	STKESA	Address of the caller's register save area.
X'18'	4	STKERET	Address of the instruction following the caller's PC instruction.
X'1C'	4	STKEPR15	Contents of parameter register 15.
X'20'	4	STKEPRM0	Contents of parameter register 0.
X'24'	4	STKEPRM1	Contents of parameter register 1.
X'34'	4	STKEEPA	Address (near the beginning) of the program receiving control (via the PC instruction).



Program A issues a PC to program B in address space Y.

Program B issues a PC to program C in the same address space.

Program C abends. Its FRR issues SDUMP.

Figure 5-57. PCLINK Control Block Structure

Module Structure

The PCLINK services are contained in the nucleus module IEAVXSTK, which has the following entry points:

Entry Point	Function
IEAVXSTS	PCLINK STACK,SAVE=YES
IEAVXSTN	PCLINK STACK,SAVE=NO
IEAVXUNS	PCLINK UNSTACK,SAVE=YES
IEAVXUNN	PCLINK UNSTACK,SAVE=NO
IEAVXEXT	PCLINK EXTRACT

The recovery routines in IEAVXSTK are:

Entry Point	Function
FRRFRSTK	FRR for PCLINK STACK service
FRRUNSTK	FRR for PCLINK UNSTACK service
FRRUNSK2	FRR for the FRRUNSTK FRR

Debugging Hints

1. All of the FRRs turn off the PCLINK super bit and issue the SETRP macro to record the error to SYS1.LOGREC. FRRUNSTK also issues the SDUMP macro and, if requested by the caller, attempts a retry.

2. By looking at the STKEs in a formatted dump, you can track the transfers of control between modules that issue the PC instruction. You can do this in the same way you would look at save areas to track transfers of control between modules that use standard linkage conventions.

3. A key indicator and field in the PSA useful for debugging are:

   PSASTKE - Pointer to the current PCLINK stack element (STKE).

   PSASTKSP - PCLINK STACK/UNSTACK super bit. This bit is set on to allow I/O and external interruptions to be disabled when the PCLINK service routine sets an FRR or issues the GETMAIN or GETSRB macro during expansion of the local or global STKE pools.

4. If an X'0D5' abend occurs (which indicates that a PT instruction failed) and the value in register 3, 4 or 14 is in error, check the following possible causes before assuming that the PCLINK UNSTACK macro failed. First check that none of the contents in the registers are destroyed between the issuance of the PCLINK UNSTACK macro and the PT instruction. Second, check that none of the contents in the registers are destroyed prior to the issuance of the PCLINK STACK macro at the beginning of the program. Note that the PCLINK service does not check the contents of the registers it is saving.

Global Resource Serialization

The global resource serialization component provides ENQ/DEQ/RESERVE services, operator commands, and GQSCAN macro support.

The ENQ/DEQ/RESERVE services control the use of serially reusable resources. The ENQ macro requests the use of a resource. The RESERVE macro requests the use of a resource and generates the hardware RESERVE instruction if the resource is on a shared DASD. The DEQ macro releases the resource.

The DISPLAY GRS and VARY GRS operator commands are provided. The DISPLAY GRS command obtains information about the systems in a global resource serialization complex and the CTC links for the system on which the command was issued. The VARY GRS command resumes global resource serialization activities following a ring disruption, resumes or suspends global resource serialization processing in a system, or purges a system from the global resource serialization complex.

The GQSCAN macro obtains the status of resources and their uses from control blocks in the global resource serialization address space.

For information on how to use the services provided by global resource serialization, refer to *OS/VS2 System Programming Library: Supervisor.* For information about the global resource serialization commands, refer to *Operator's Library: System Commands.* For logic information about this component, refer to *OS/VS2 MVS Global Resource Serialization Logic.*

This section contains the following topics for global resource serialization:

- Functional Overview - includes a brief description of the various subcomponent functions.

- Control Blocks - contains control block overviews related to selected subcomponents.

- Module Flow Diagrams - these diagrams show the flow of control between global resource serialization modules for selected functions.

- Diagnostic Aids - contains debugging hints to help you isolate problems in global resource serialization.

Functional Overview

The global resource serialization component consists of several subcomponents. All global resource serialization module names begin with ISG and the fourth character identifies the subcomponent that the module supports. The following table summarizes the module naming conventions.

Module names: ISGsmmmm

ISG = Global resource serialization

s =	**Subcomponent**
B	Ring processing
C	Command processing
D	Dump support
G/L	Resource request processing G - Mainline ENQ/DEQ/RESERVE processing L - Fast path ENQ/DEQ/processing
J	CTC processing
M	WTO/WTOR message processing
N	Initialization
Q	Queue scanning services
S	Storage management

mmmm = Module identifier

(*Note:* An exception is ISGJPARM, an initialization module.)

Ring Processing

The ring processing subcomponent has the following functions:

- Pass to all systems in the main ring the information required to serialize global resource requests across all the systems.
- Add or delete systems from the main ring as specified in the initialization parameters or requested by the operator.
- Provide information about the system and CTCs in the complex.

Other subcomponents invoked by ring processing are:

- CTC processing
- Storage management
- WTO/WTOR message processing
- Resource request processing

Command Processing

The command processing subcomponent supports the VARY GRS and the DISPLAY GRS operator commands. The global resource serialization interface module executes in the master scheduler's address space and receives control from the command service processor. The command interface module posts the command router in the global resource serialization address space. The command router module attaches the DISPLAY, PURGE, QUIESCE, or RESTART command request processor.

Other subcomponents invoked by command processing are:

- Queue scanning services
- Ring processing
- WTO/WTOR message processing
- Storage management

Dump Support

The dump support subcomponent provides the support to dump the global resource serialization control blocks. The dump support modules obtain and format information for SNAP dump, SVC dump, print dump (PRDMP), and interactive problem control system (IPCS). Queue scanning services is invoked by dump support.

Resource Request Processing (Mainline and Fast Path)

The resource request processing subcomponent performs ENQ/DEQ/RESERVE request processing. It is also invoked during task and address space terminations to clean up outstanding requests. Fast path processing handles local ENQ and DEQ requests which do not require special processing. Mainline processing handles all other requests.

Other subcomponents invoked by resource request processing are:

- Ring processing
- Storage management

CTC Processing

The CTC processing subcomponent builds control blocks for the CTCs that connect the system in a global resource serialization complex (based on information specified in the GRSCNFxx member of SYS1.PARMLIB). It initiates I/O on the CTCs and handles their interrupts. When an interrupt is received, it invokes the ring processing subcomponent.

WTO/WTOR Message Processing

The message processing subcomponent issues global resource serialization messages (ISGnnns) to the operator and to the system log. It issues both informational (WTO/MLWTO) and reply (WTOR) messages to the operator and informational (WTL) messages to the system log.

Initialization

The initialization subcomponent has two primary functions:

- Creating and initializing the global resource serialization address space.

- Establishing the global resource serialization complex defined in the GRSCNFxx and IEASYSxx members of SYS1.PARMLIB.

All other subcomponents are invoked by initialization except dump support.

Initialization is not discussed further in this publication. See *OS/VS2 System Initialization Logic* for information on this subcomponent.

Queue Scanning Services

The queue scanning services subcomponent processes the requests made via the GQSCAN macro.

Other subcomponents invoked by queue scanning are:

- Storage management
- Ring processing

Storage Management

The storage management subcomponent maintains the resource queue area (RQA) of the global resource serialization address space. An interface module provides access to the storage manager for routines that do no execute in the global resource serialization address space. Also, the storage manager hashes resource names and SYSID/ASID information to expedite searches into the local queue hash table (LQHT), global queue hash table (GQHT), and system/ASID hash table (SAHT).

Control Blocks

Global resource serialization uses the following control blocks. For the format of these data areas, refer to *OS/VS2 System Programming Library: Debugging Handbook* and *OS/VS2 Data Areas* (microfiche).

Data Area	Description
CEPL	Command ESTAE parameter list - anchors the LIFO queue of CRWAs and contains an error recording area for requested functions.
CRB	Command request block - contains information required to process a DISPLAY GRS or VARY GRS command.
CRWA	Command recovery work area - contains the error information used by the command recovery routine to handle errors.
DEPL	SDUMP ESTAE parameter list - contains information used by the global resource serialization dump support subcomponent to process an SDUMP request.
DPL	DEQ purge list - contains the information needed to complete processing for a DEQ SYSID, DEQ ASID, or DEQ TCB purge request.
DSPL	Dump sort parameter list - contains information for the global resource serialization dump sort routine.
GCB	Global resource serialization CTC-driver request block - is the parameter list required by the CTC-driver for all functions (except extracting area lengths).
GCC	Global resource serialization CTC-driver control card table - contains the information from the global resource serialization SYS1.PARMLIB member for this system.
GCL	Global resource serialization CTC-driver link control block - contains information related to each CTC in the system.

GCP Global resource serialization CTC-driver buffer prefix - contains message length and validity checking data.

GCQ Global resource serialization CTC-driver queueing element - contains information used by CTC processing when sending or receiving a message or an unusual-event notification.

GCT Global resource serialization CTC-driver branch table - contains addresses of the CTC processing DIE routines and exit routines.

GCV Global resource serialization CTC-driver vector table - contains addresses of CTC-driver entry points for CTC-driver functions and information common to all CTCs used by CTC processing.

GCX Global resource serialization CTC-driver extract table - is the parameter list required by the CTC-driver for the extraction of area lengths.

GVT Global resource serialization vector table - contains common information (global queues, pointers and entry point addresses) for all global resource serialization functions. It also has sections containing information for the various subcomponents.

GVTX Global resource serialization vector table extension - contains information specific to the global resource serialization address space.

MRB Message request block - contains information required to process message requests.

PEL Parameter element - is the input parameter list to ENQ/DEQ/RESERVE processing.

PEXB Pool extent block - maps a 4K page in the RQA.

PQCB Placeholder queue control block - contains the information necessary to resume a global resource serialization queue scanning request.

QCB Queue control block - describes a resource to global resource serialization.

QEL Queue element - describes the requester of a resource to global resource serialization.

QFPL ENQ/DEQ/FRR parameter list - is the FRR parameter list used by ENQ/DEQ/RESERVE processing.

QFPL1 Queue scanning services FRR parameter list - is the FRR parameter list used by queue scanning services.

QHT Queue has table - contains queue hash table entries. Each queue hash table entry is a double-headed anchor of QCBs. There are two QHTs; one for global requests (GQHT), and one for local requests (LQHT).

QWA Queue work area - is a work area used by ENQ/DEQ/RESERVE processing modules.

QWB Queue work block - describes a resource request. A global resource request is described by a QWB in the private area of the global resource serialization address space. A local resource request is described by the permanent QWB in the SQA.

QXB Queue extension block - contains the data that describes an ENQ/DEQ/RESERVE request.

REPL Ring processing ESTAE parameter list - is the ESTAE parameter list used by ring processing.

RIB/RIBE Resource information block - contains the information that describes a resource and any requesters for the resource. The variable portion of the RIB (containing RIB extents) is located immediately after the RIB. Each RIB extent (RIBE) describes a requester of the resource. RIBs and RIBEs are returned to the issuer of the GQSCAN macro.

RNLE Resource name list entry - contains information about resources that are to be included or excluded from global resource serialization processing and RESERVE resources that are to be converted to global ENQs.

RPT Resource pool table - contains entries for each cell type in the RQA. There are two RPTs - one for global resources (GRPT), and one for local resources (LRPT). Each RPT points to the first and last PEXB for that pool.

RQA Resource queue area - contains PEXBs that define QCBs, QELs, QXBs, QWBs, PQCBs, MRBs, CRBs, and work areas.

RSA Ring processing system authority message - is used to pass command data and ENQ/DEQ/RESERVE requests between global resource serialization systems in the main ring.

RSAIRCD Ring processing information record - is used to pass control information between systems that are not both in the main ring.

RSC Ring status change parameter list - is the parameter list used to call the interface module ISGBCI.

RSL Ring processing system link block - contains information about a CTC and is used by global resource serialization ring processing functions.

RST Ring processing status table - contains the status of global resource serialization systems and CTCs.

RSV Ring processing system vector table - contains information used by the global resource serialization ring processing modules.

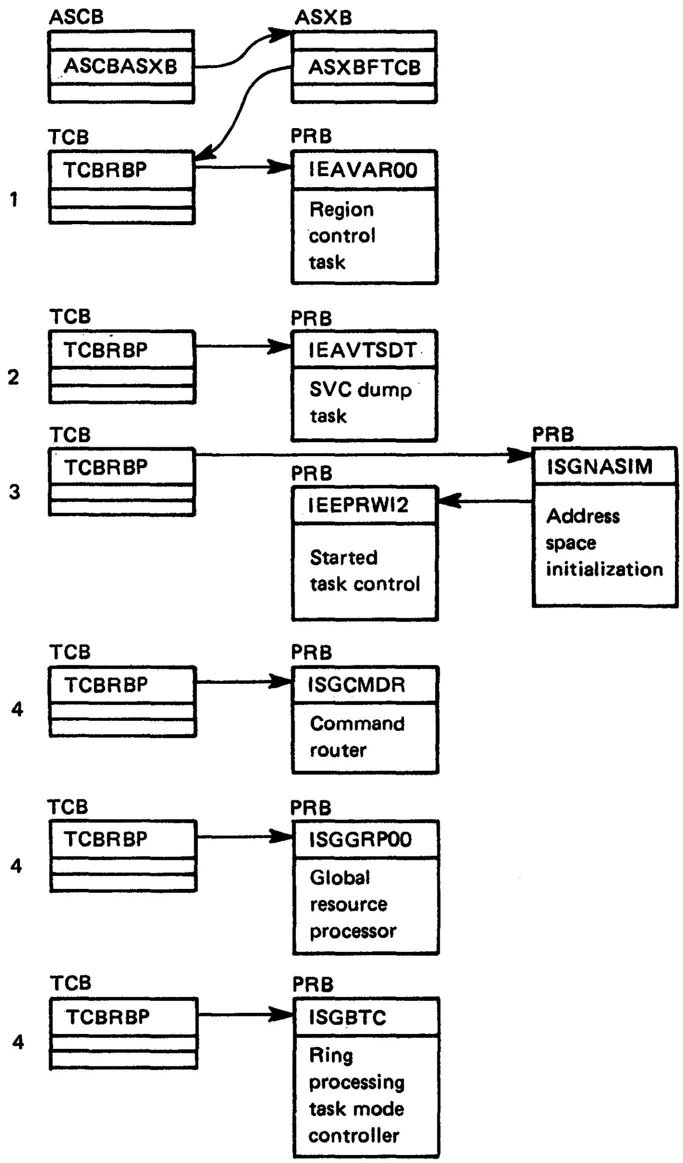
SAHT System/ASID hash table - contains entries that point to a chain of QELs that define global resource requesters from another system.

SMPL Storage management parameter list entry - contains information for a request to global resource serialization storage management.

Control Block Overviews

The figures in this topic show the control block structures of the global resource serialization control blocks for the following:

- Permanent TCBs
- CTC processing
- Ring processing
- Command processing
- ENQ/DEQ processing
  - Local resources
  - Global resources
- Queue scanning services
  - Local resources
  - Global resources
- Storage management
- WTO/WTOR Message processing



Notes:

- The numbers show the hierarchy.
- When GRS=START or JOIN, all TCB/PRBs are permanent.
- When GRS=NONE: all TCB/PRBs are permanent except the TCB/PRB for ISGGRP00, which is temporary; and the TCB/PRB for ISGBTC, which is not present.

Figure 5-58. TCBs in the Global Resource Serialization Address Space

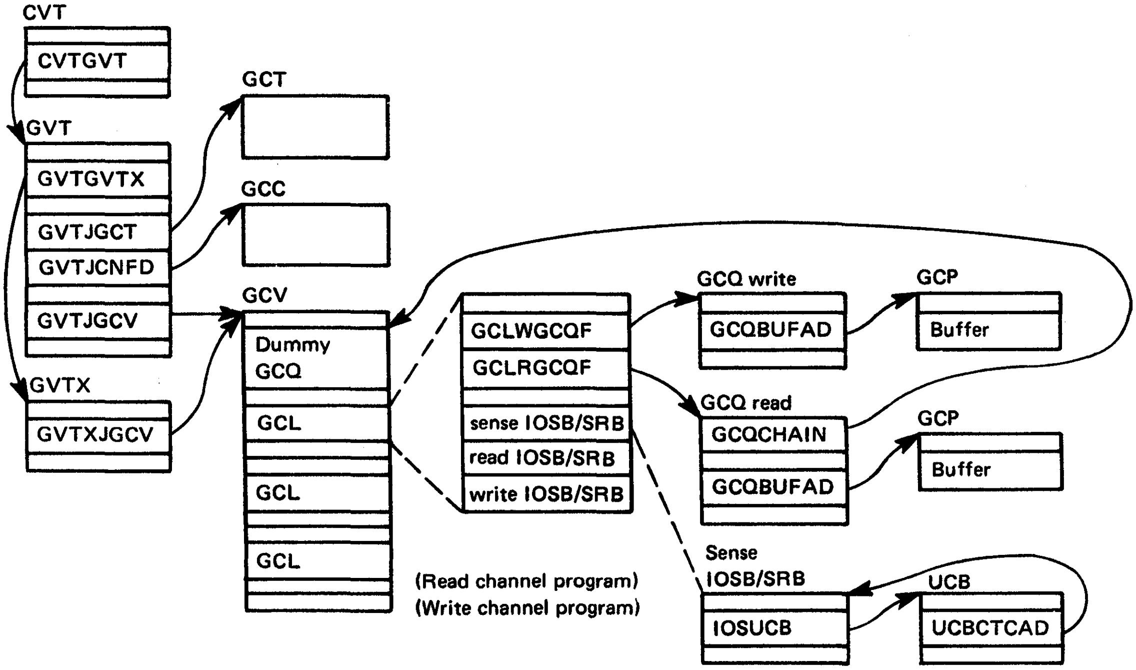


Figure 5-59. CTC Processing - Control Block Overview

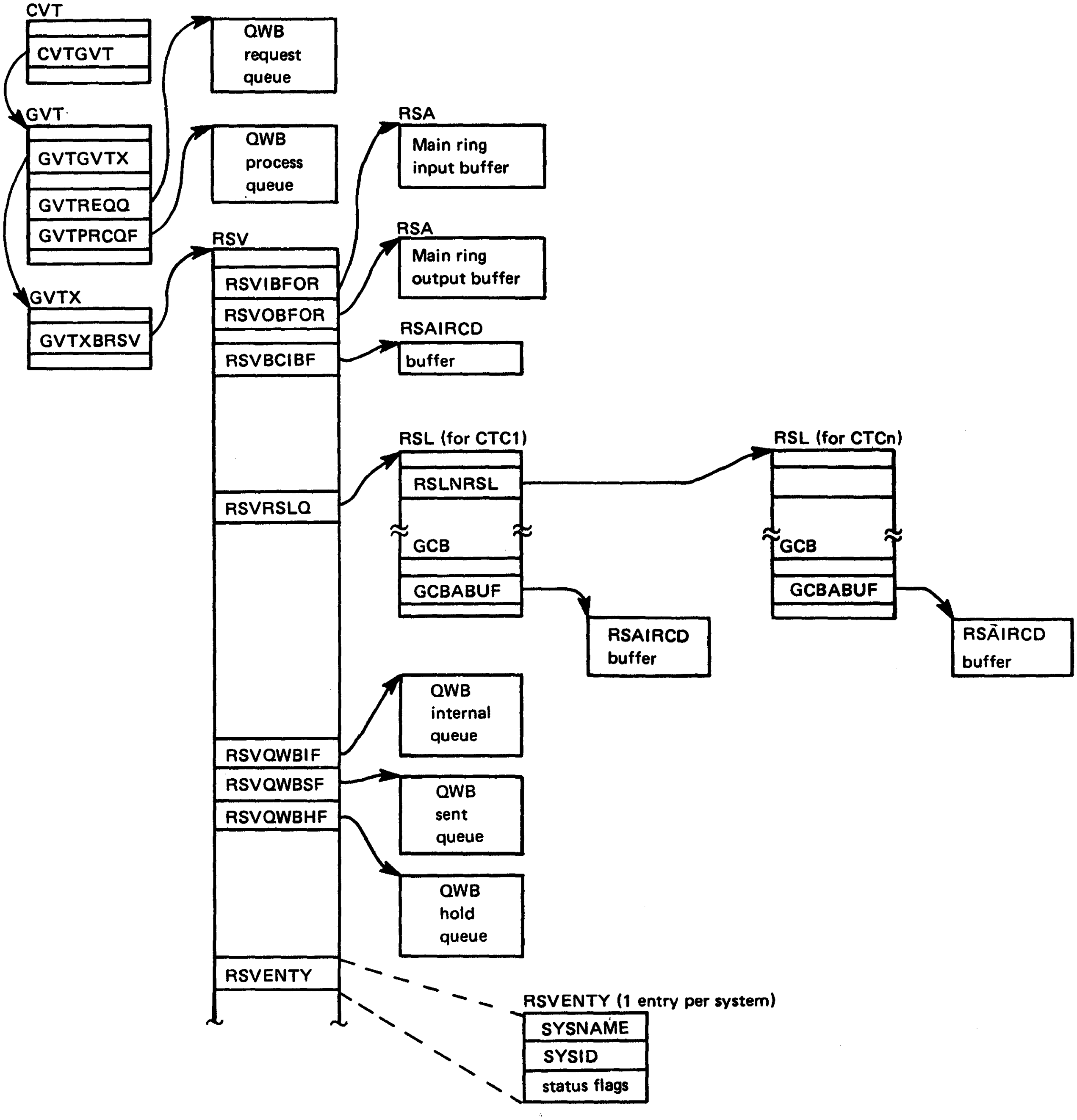


Figure 5-60. Ring Processing - Control Block Overview

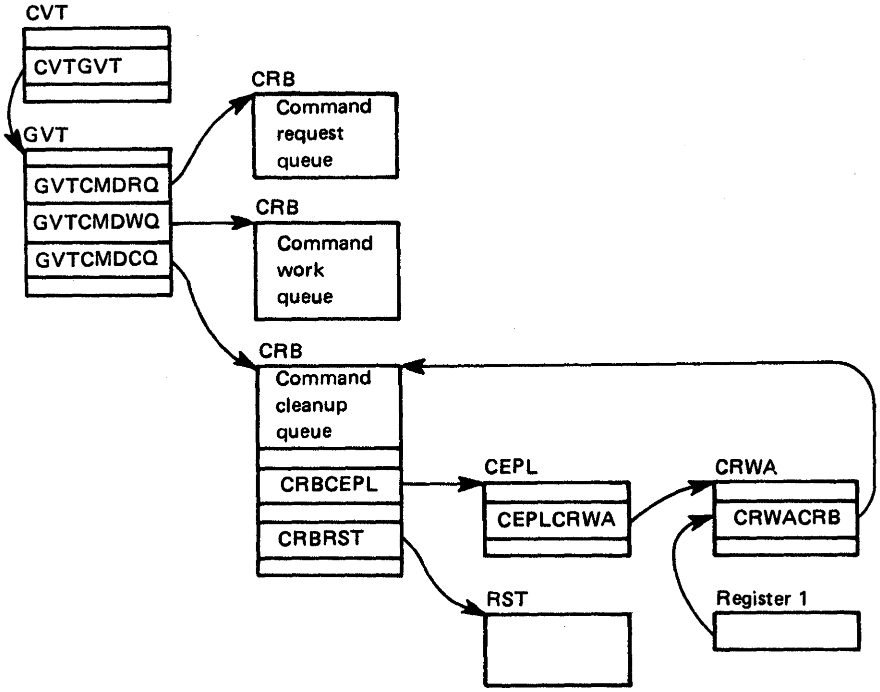


Figure 5-61. Command Processing - Control Block Overview

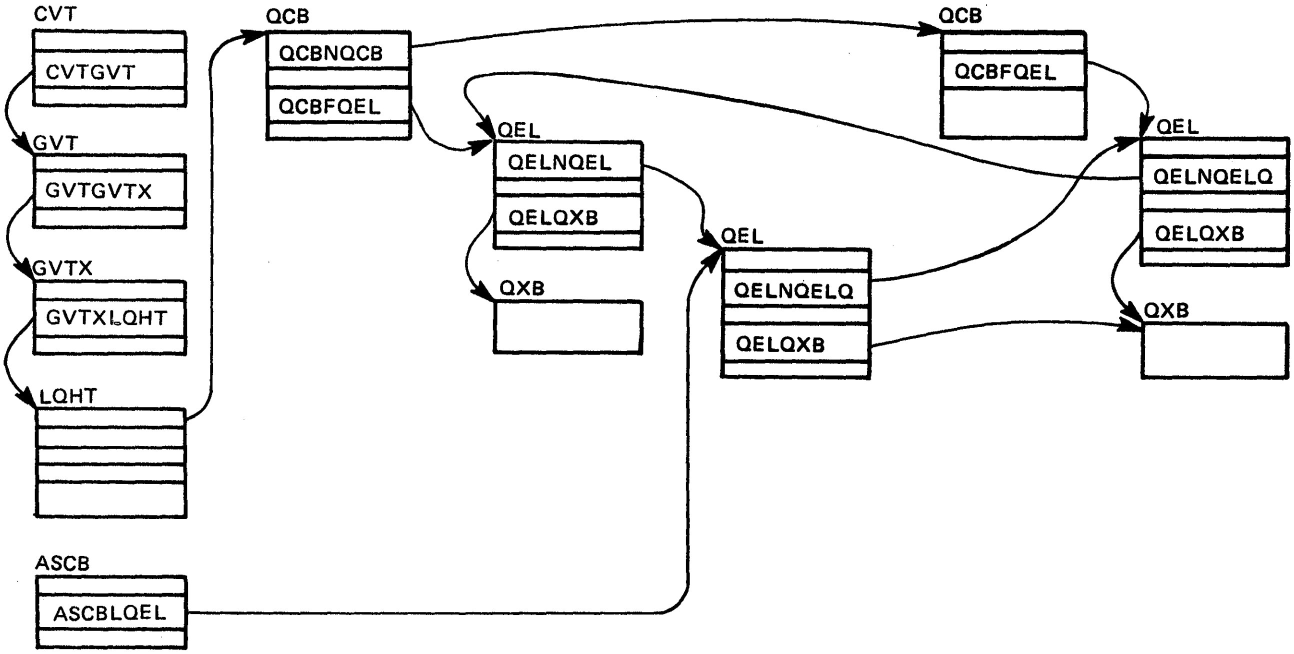


Figure 5-62. ENQ/DEQ Processing - Local Resources - Control Block Overview

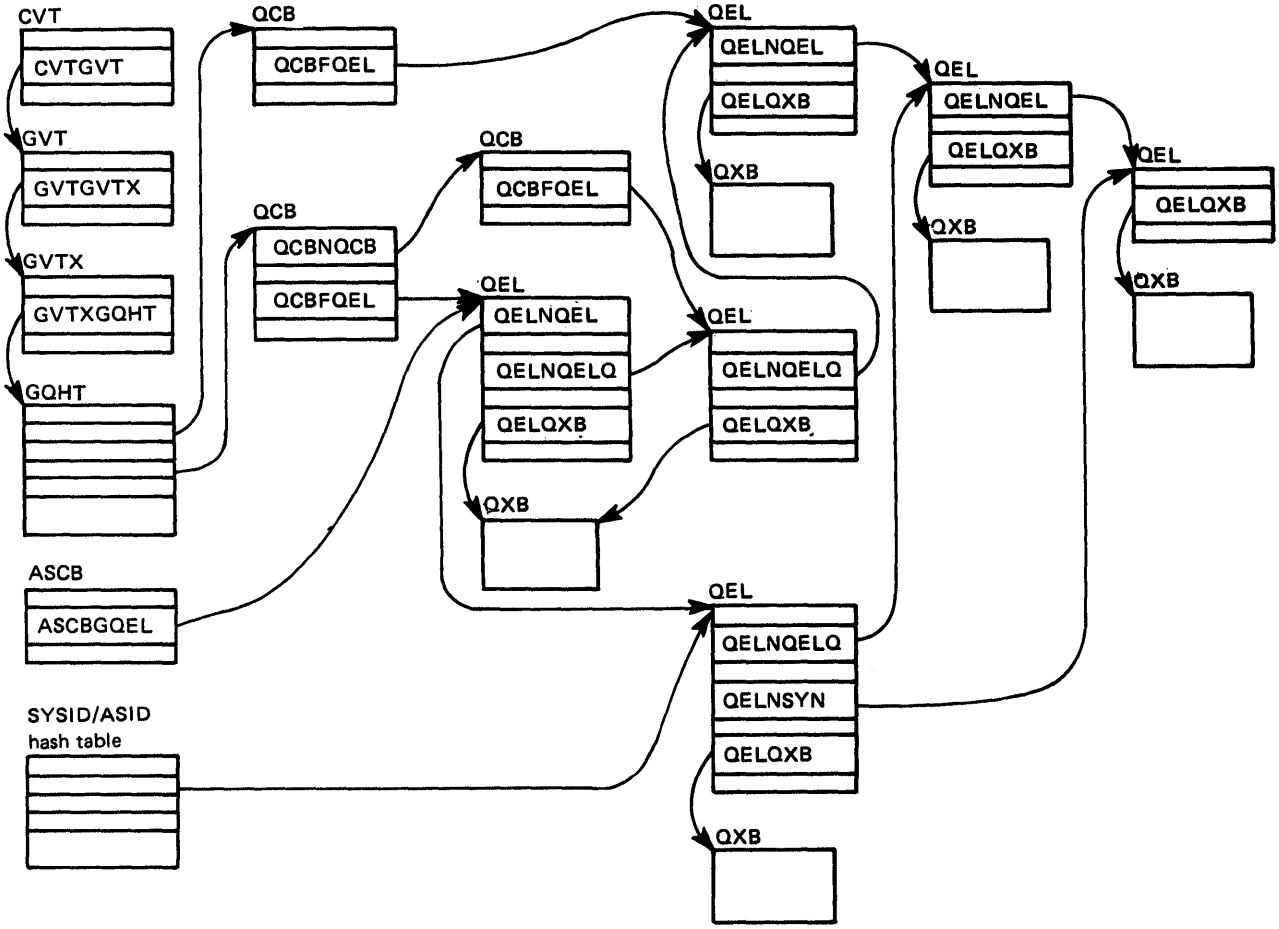


Figure 5-63. ENQ/DEQ Processing - Global Resources - Control Block Overview

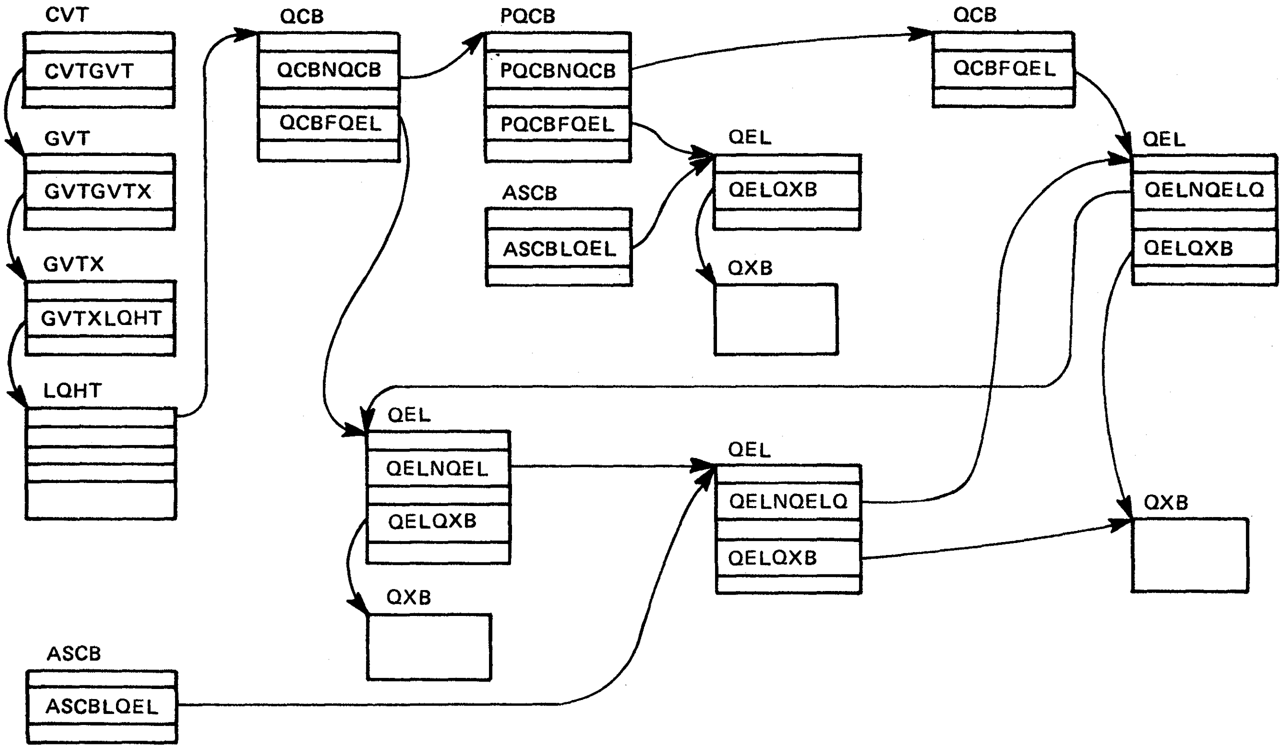


Figure 5-64. Queue Scanning Services - Local Resources - Control Block Overview

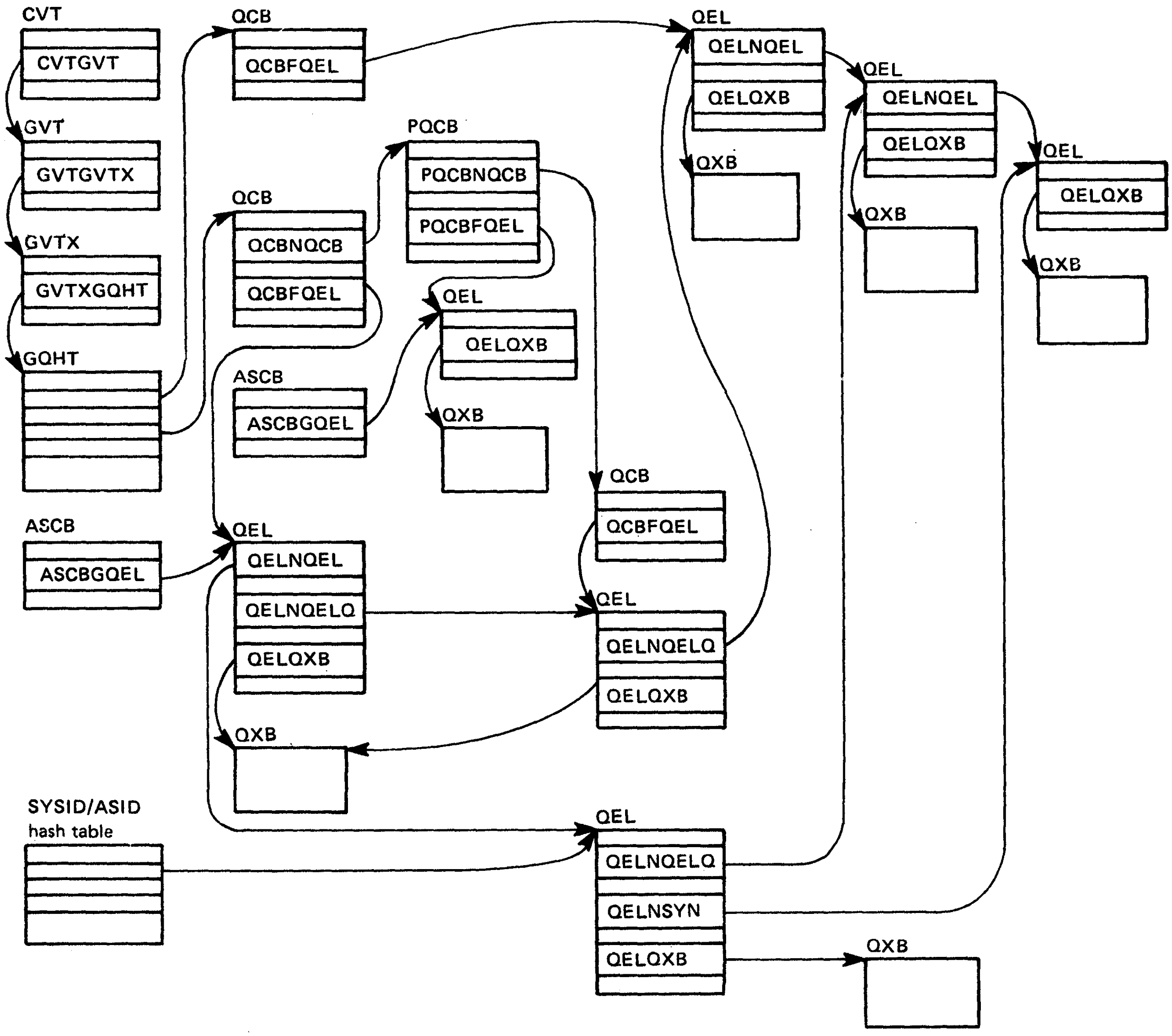


Figure 5-65. Queue Scanning Services - Global Resources - Control Block Overview

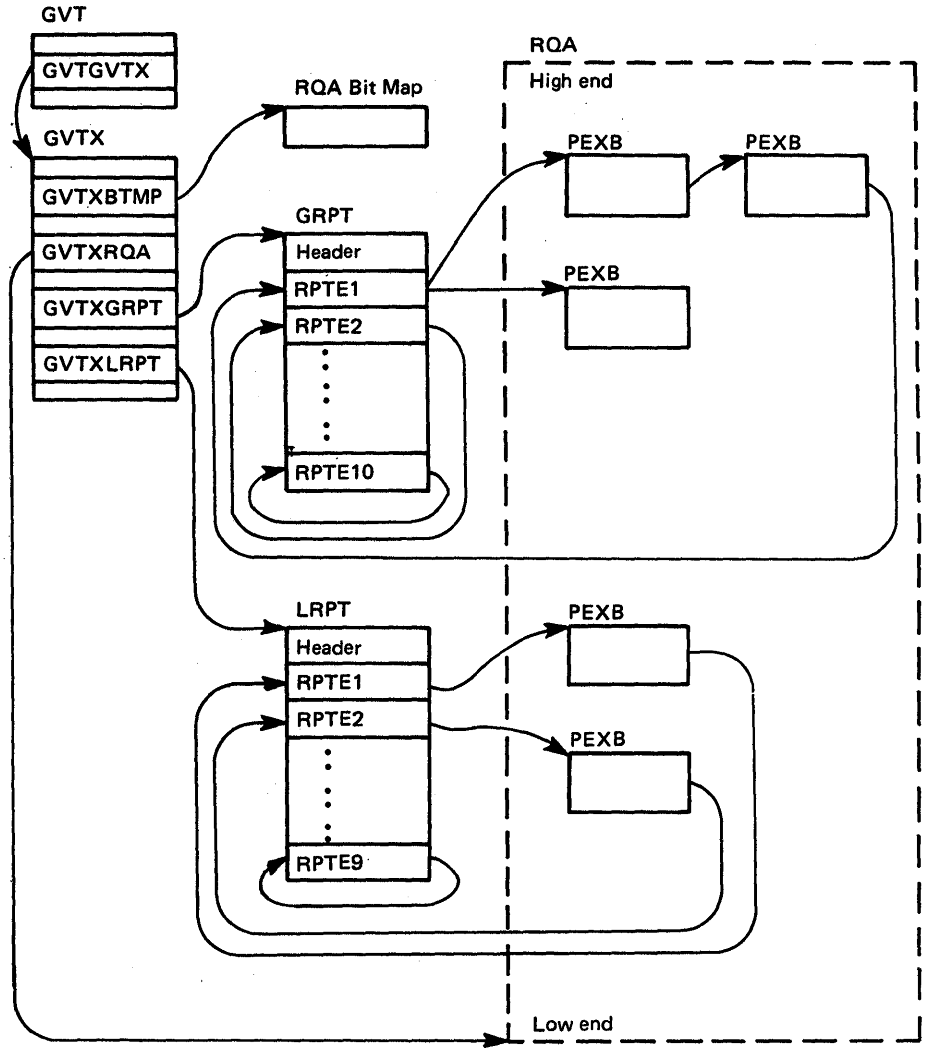
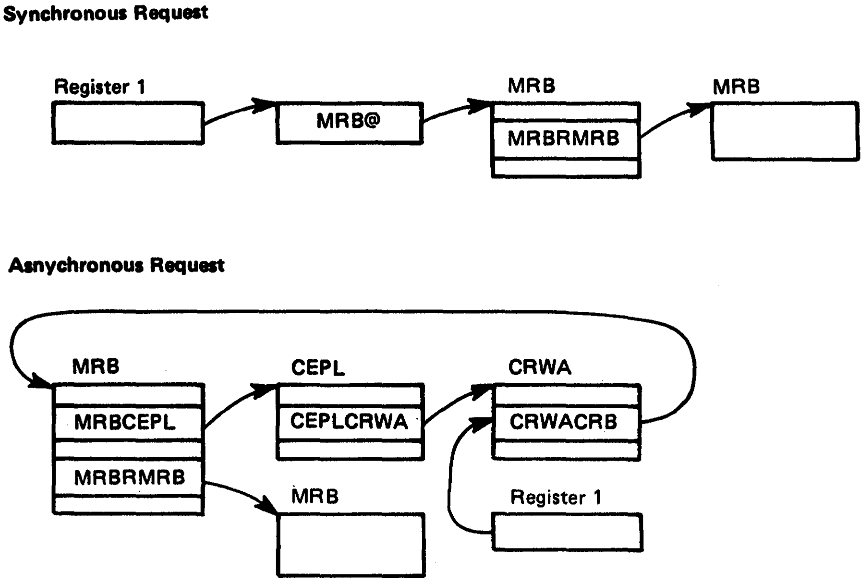


Figure 5-66. Storage Management - Control Block Overview



Note: Control block structure when the message processing routine (ISGMSG00) receives control.

Figure 5-67. WTO/WTOR Message Processing - Control Block Overview

Module Flow Diagrams

The figures in this topic show the flow of control between global resource serialization modules for selected functions. Figure 5-68 shows an overview of module flow between subcomponents, and includes a directory of the module flow diagrams to help you use them.

In the diagrams, a solid double line indicates communication across the I/O interface between systems. A broken double line indicates communication between address spaces within one system.

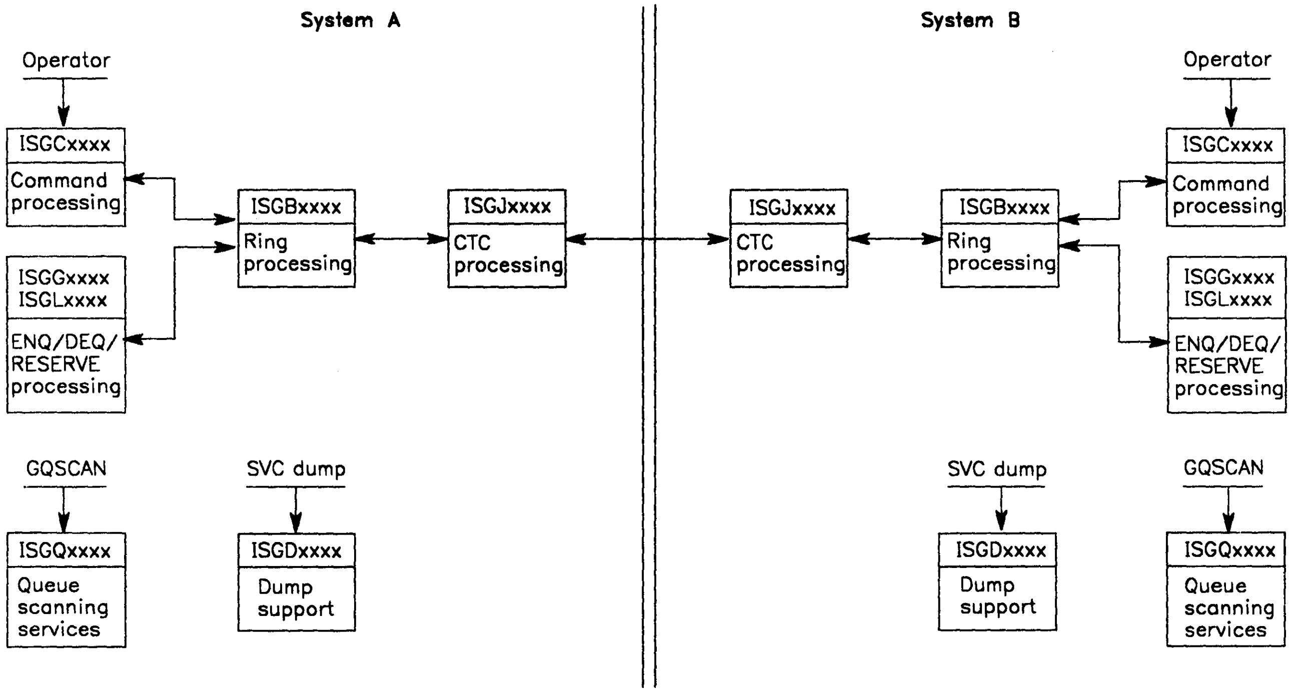


Figure Title (Module Flow for:)

CTC Processing

5-69 -- Handle Arrival of Immediate-CCW
5-70 -- Handle Arrival of RSA or RSAIRCD
5-71 -- Send a RSA or RSAIRCD

Ring Processing

5-72 -- Send/Receive a RSA
5-73 -- Send a RSAIRCD or Immediate-CCW (Requested by ISGBCI)
5-74 -- Send a RSAIRCD (Requested by ISGBTC)
5-75 -- Handle Arrival of RSAIRCD (Not Requested by This System)
5-76 -- SNAPSHOT Function
5-77 -- SENDCMD (RSCRADDS) Function
5-78 -- SENDCMD (RSCRSNAD) Function

Command Processing

5-79 -- Command Initialization and Cleanup
5-80 -- DISPLAY GRS
5-81 -- VARY GRS(x), PURGE
5-82 -- VARY GRS(x), QUIESCE to Another System
5-83 -- VARY GRS(x), QUIESCE by a System to Quiesce Itself
5-84 -- VARY GRS(x), RESTART to Restart Another System
5-85 -- VARY GRS(ALL), RESTART to Restart All Systems
5-86 -- VARY GRS(x), RESTART by a System Not in the Main Ring
5-87 -- Join Processing at Initialization Time

ENQ/DEQ Mainline (Resource request processing)

5-88 -- Local Resource Request
5-89 -- Global Resource Request
5-90 -- Termination Resource Manager

5-91 -- *Queue Scanning Services*

5-92 -- *Dump Support -- SVC Dump*

Figure 5-68. Module Flow Overview and Directory

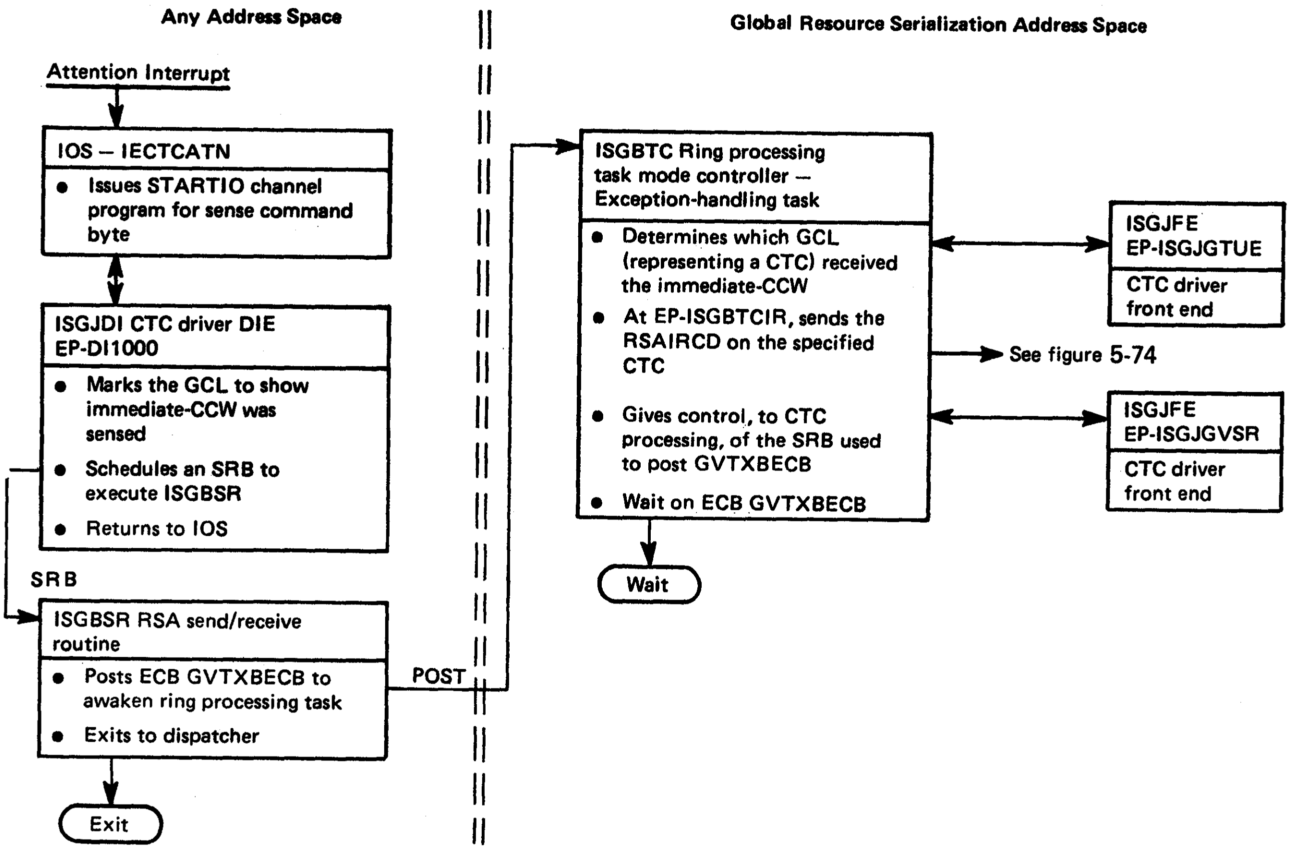

Any Address Space
Global Resource Serialization Address Space
Attention Interrupt
IOS – IECTCATN
● Issues STARTIO channel program for sense command byte
ISGJDI CTC driver DIE EP-DI1000
● Marks the GCL to show immediate-CCW was sensed
● Schedules an SRB to execute ISGBSR
● Returns to IOS
SRB
ISGBSR RSA send/receive routine
● Posts ECB GVTXBECB to awaken ring processing task
● Exits to dispatcher
Exit
POST
ISGBTC Ring processing task mode controller – Exception-handling task
● Determines which GCL (representing a CTC) received the immediate-CCW
● At EP-ISGBTCIR, sends the RSAIRCD on the specified CTC
● Gives control, to CTC processing, of the SRB used to post GVTXBECB
● Wait on ECB GVTXBECB
Wait
ISGJFE EP-ISGJGTUE
CTC driver front end
See figure 5-74
ISGJFE EP-ISGJGVSR
CTC driver front end

Figure 5-69. Module Flow for CTC Processing - Handle Arrival of Immediate-CCW

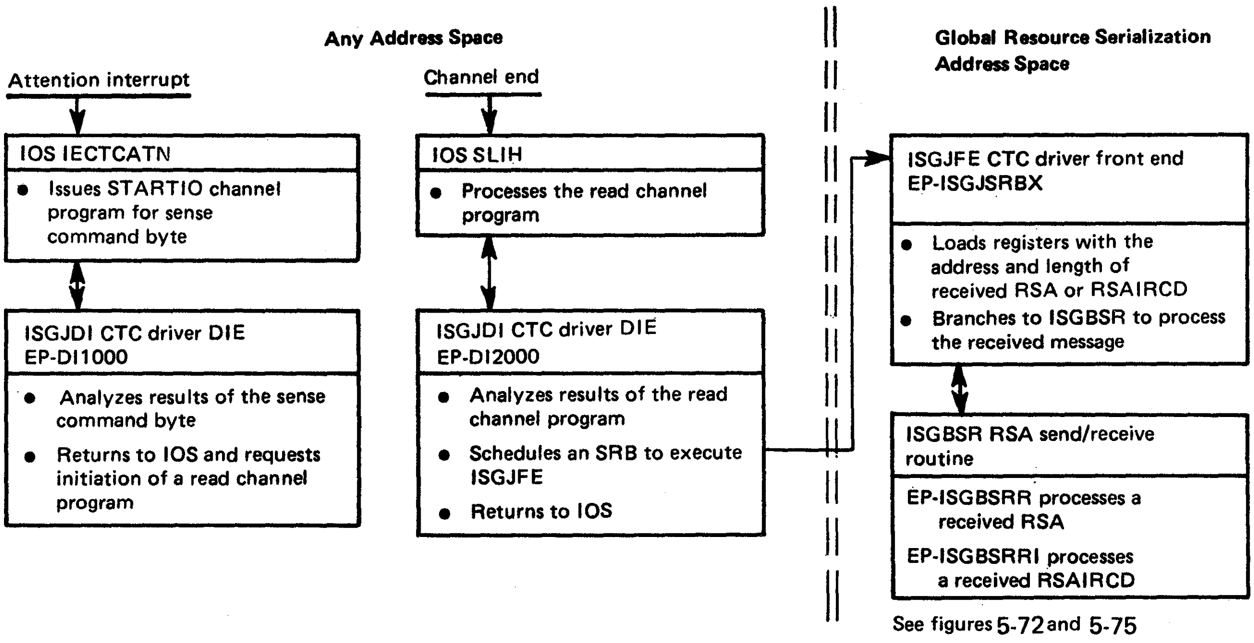

Any Address Space
Global Resource Serialization Address Space
Attention interrupt
IOS IECTCATN
● Issues STARTIO channel program for sense command byte
ISGJDI CTC driver DIE EP-DI1000
● Analyzes results of the sense command byte
● Returns to IOS and requests initiation of a read channel program
Channel end
IOS SLIH
● Processes the read channel program
ISGJDI CTC driver DIE EP-DI2000
● Analyzes results of the read channel program
● Schedules an SRB to execute ISGJFE
● Returns to IOS
ISGJFE CTC driver front end EP-ISGJSRBX
● Loads registers with the address and length of received RSA or RSAIRCD
● Branches to ISGBSR to process the received message
ISGBSR RSA send/receive routine
EP-ISGBSRR processes a received RSA
EP-ISGBSRRI processes a received RSAIRCD
See figures 5-72 and 5-75

Figure 5-70. Module Flow for CTC Processing - Handle Arrival of RSA or RSAIRCD

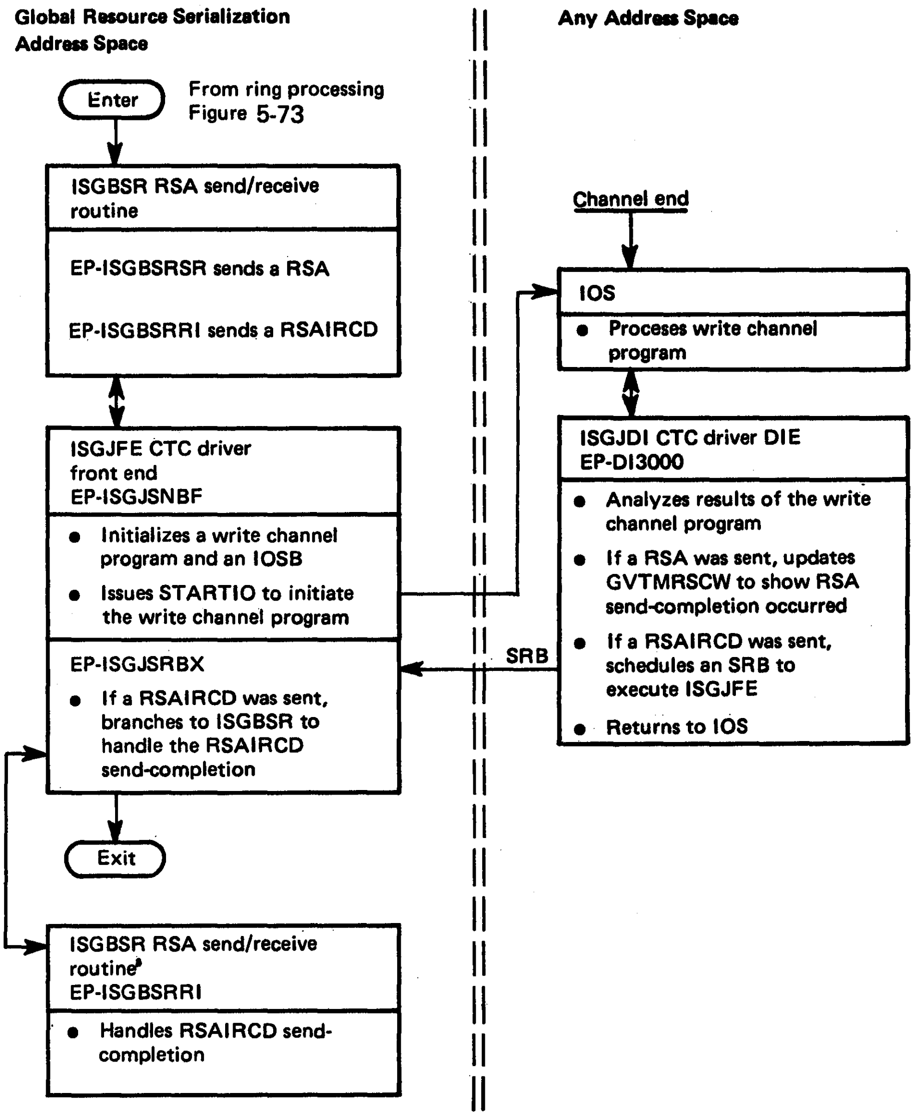
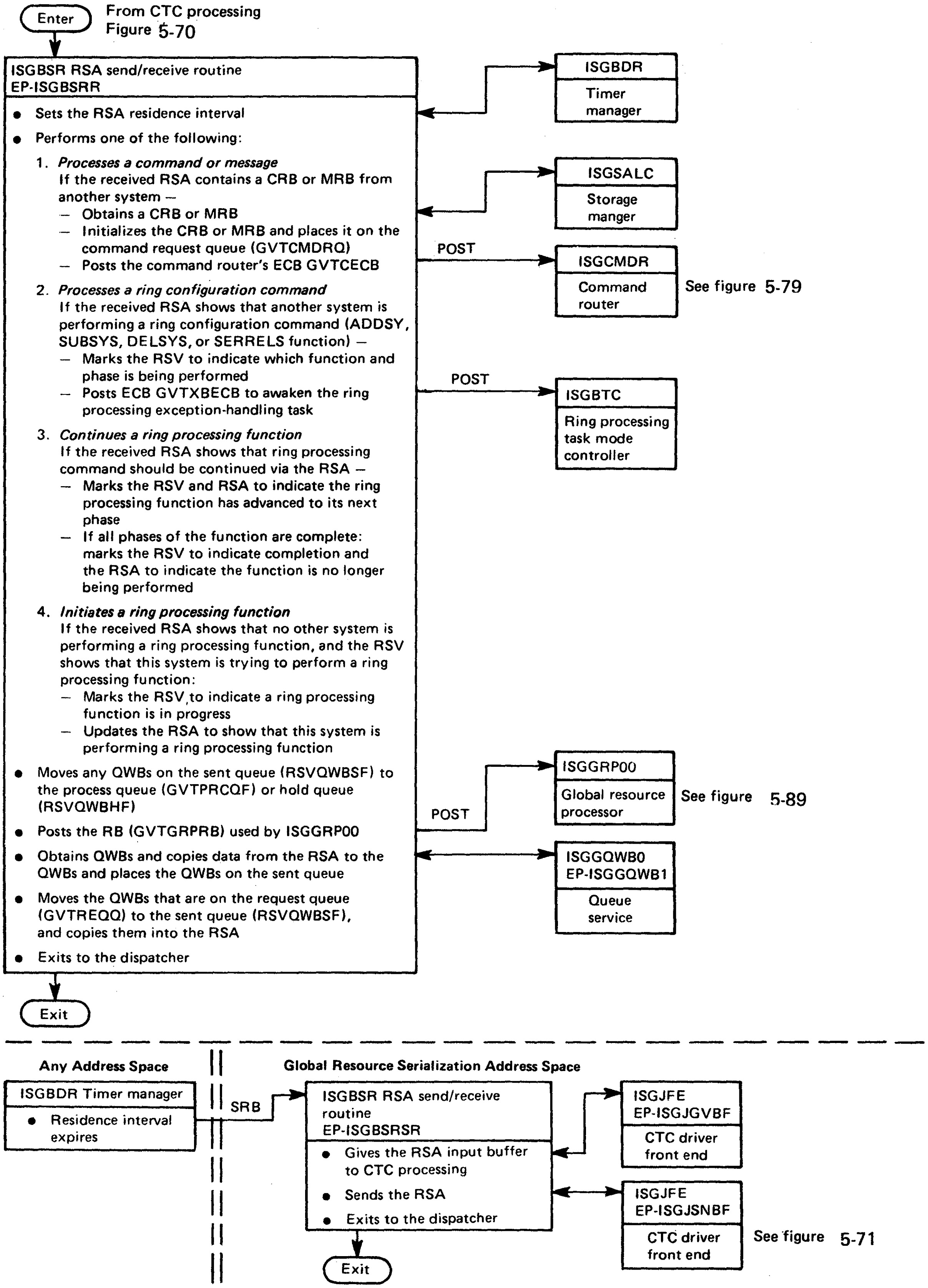

Global Resource Serialization Address Space
Any Address Space
Enter
From ring processing Figure 5-73
ISGBSR RSA send/receive routine
EP-ISGBSRSR sends a RSA
EP-ISGBSRRI sends a RSAIRCD
ISGJFE CTC driver front end EP-ISGJSNBF
● Initializes a write channel program and an IOSB
● Issues STARTIO to initiate the write channel program
EP-ISGJSRBX
● If a RSAIRCD was sent, branches to ISGBSR to handle the RSAIRCD send-completion
Exit
ISGBSR RSA send/receive routine EP-ISGBSRRI
● Handles RSAIRCD send-completion
Channel end
IOS
● Proceses write channel program
ISGJDI CTC driver DIE EP-DI3000
● Analyzes results of the write channel program
● If a RSA was sent, updates GVTMRSCW to show RSA send-completion occurred
● If a RSAIRCD was sent, schedules an SRB to execute ISGJFE
● Returns to IOS
SRB

Figure 5-71. Module Flow for CTC Processing - Send a RSA or RSAIRCD



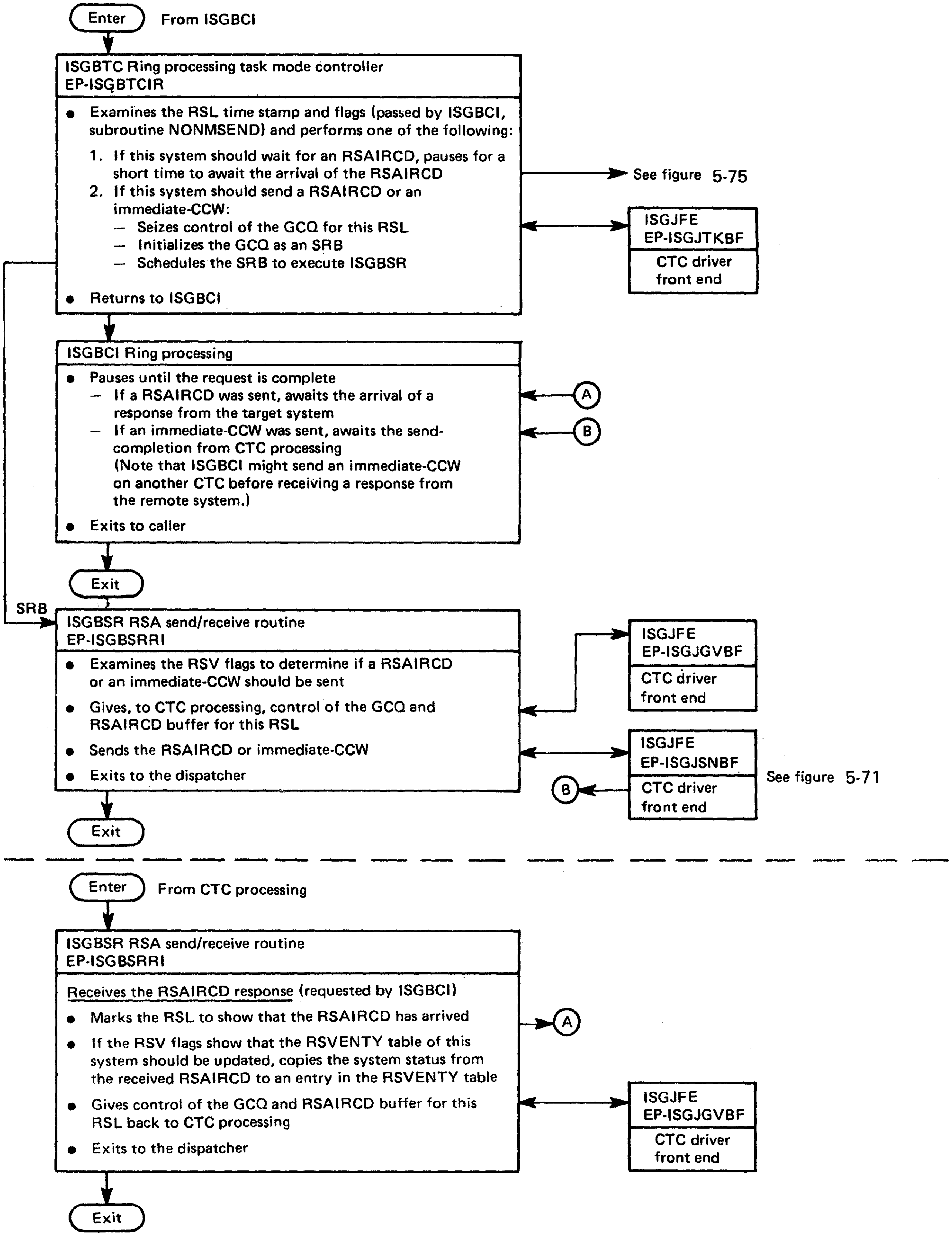
ISGBSR RSA send/receive routine
EP-ISGBSRR

- Sets the RSA residence interval
- Performs one of the following:
  1. *Processes a command or message*
     If the received RSA contains a CRB or MRB from another system –
     – Obtains a CRB or MRB
     – Initializes the CRB or MRB and places it on the command request queue (GVTCMDRQ)
     – Posts the command router's ECB GVTCECB
  2. *Processes a ring configuration command*
     If the received RSA shows that another system is performing a ring configuration command (ADDSY, SUBSYS, DELSYS, or SERRELS function) –
     – Marks the RSV to indicate which function and phase is being performed
     – Posts ECB GVTXBECB to awaken the ring processing exception-handling task
  3. *Continues a ring processing function*
     If the received RSA shows that ring processing command should be continued via the RSA –
     – Marks the RSV and RSA to indicate the ring processing function has advanced to its next phase
     – If all phases of the function are complete: marks the RSV to indicate completion and the RSA to indicate the function is no longer being performed
  4. *Initiates a ring processing function*
     If the received RSA shows that no other system is performing a ring processing function, and the RSV shows that this system is trying to perform a ring processing function:
     – Marks the RSV,to indicate a ring processing function is in progress
     – Updates the RSA to show that this system is performing a ring processing function
- Moves any QWBs on the sent queue (RSVQWBSF) to the process queue (GVTPRCQF) or hold queue (RSVQWBHF)
- Posts the RB (GVTGRPRB) used by ISGGRP00
- Obtains QWBs and copies data from the RSA to the QWBs and places the QWBs on the sent queue
- Moves the QWBs that are on the request queue (GVTREQQ) to the sent queue (RSVQWBSF), and copies them into the RSA
- Exits to the dispatcher

ISGBDR Timer manager
- Residence interval expires

ISGBSR RSA send/receive routine
EP-ISGBSRSR
- Gives the RSA input buffer to CTC processing
- Sends the RSA
- Exits to the dispatcher

Figure 5-72. Module Flow for Ring Processing - Send/Receive a RSA



ISGBTC Ring processing task mode controller
EP-ISGBTCIR

- Examines the RSL time stamp and flags (passed by ISGBCI, subroutine NONMSEND) and performs one of the following:
  1. If this system should wait for an RSAIRCD, pauses for a short time to await the arrival of the RSAIRCD
  2. If this system should send a RSAIRCD or an immediate-CCW:
     - Seizes control of the GCQ for this RSL
     - Initializes the GCQ as an SRB
     - Schedules the SRB to execute ISGBSR
- Returns to ISGBCI

ISGBCI Ring processing

- Pauses until the request is complete
  - If a RSAIRCD was sent, awaits the arrival of a response from the target system
  - If an immediate-CCW was sent, awaits the send-completion from CTC processing (Note that ISGBCI might send an immediate-CCW on another CTC before receiving a response from the remote system.)
- Exits to caller

ISGBSR RSA send/receive routine
EP-ISGBSRRI

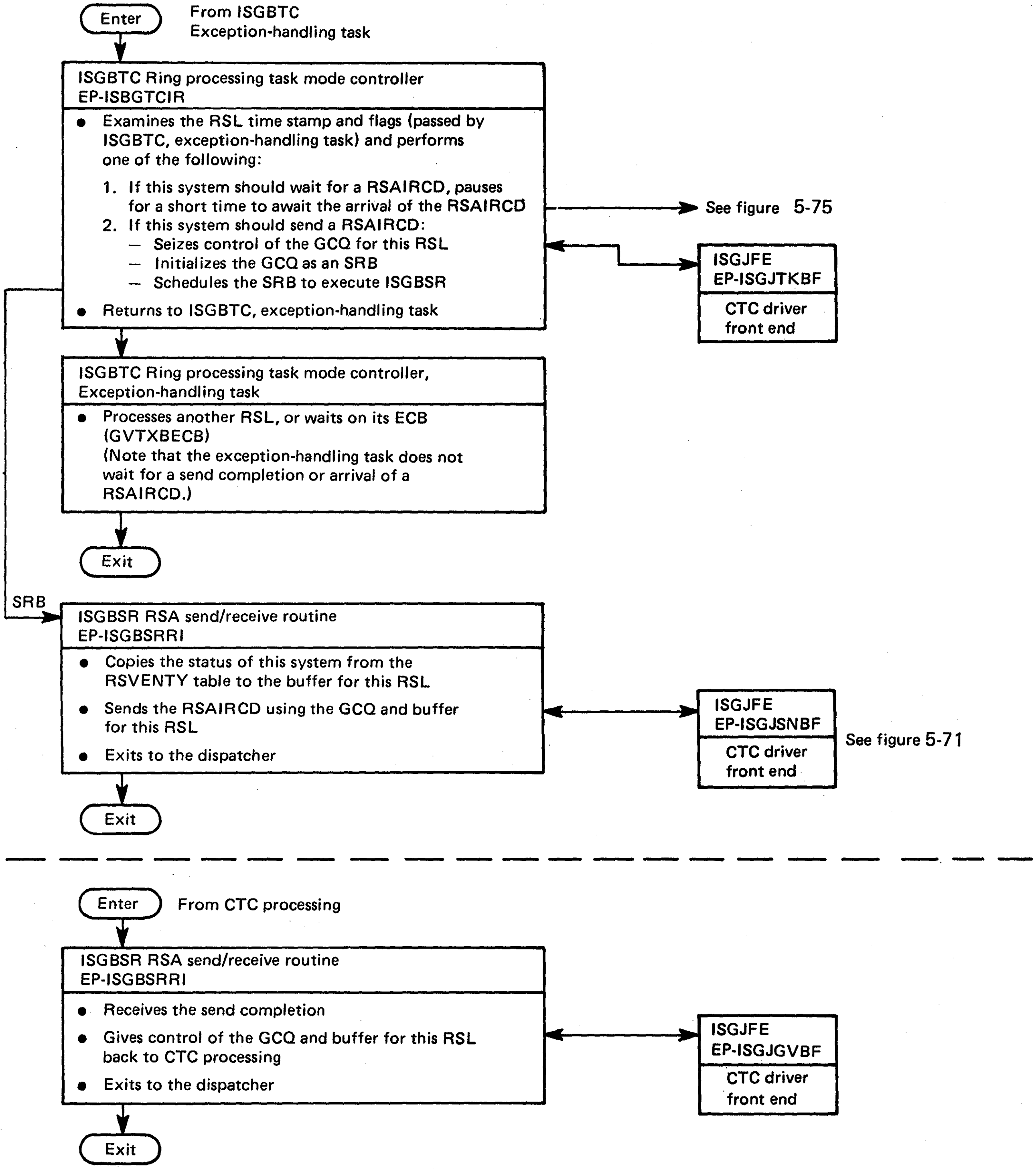
- Examines the RSV flags to determine if a RSAIRCD or an immediate-CCW should be sent
- Gives, to CTC processing, control of the GCQ and RSAIRCD buffer for this RSL
- Sends the RSAIRCD or immediate-CCW
- Exits to the dispatcher

ISGBSR RSA send/receive routine
EP-ISGBSRRI

Receives the RSAIRCD response (requested by ISGBCI)

- Marks the RSL to show that the RSAIRCD has arrived
- If the RSV flags show that the RSVENTY table of this system should be updated, copies the system status from the received RSAIRCD to an entry in the RSVENTY table
- Gives control of the GCQ and RSAIRCD buffer for this RSL back to CTC processing
- Exits to the dispatcher

Figure 5-73. Module Flow for Ring Processing - Send a RSAIRCD or Immediate-CCW (Requested by ISGBCI)



Enter — From ISGBTC Exception-handling task

ISGBTC Ring processing task mode controller
EP-ISBGTCIR

- Examines the RSL time stamp and flags (passed by ISGBTC, exception-handling task) and performs one of the following:
  1. If this system should wait for a RSAIRCD, pauses for a short time to await the arrival of the RSAIRCD
  2. If this system should send a RSAIRCD:
     - Seizes control of the GCQ for this RSL
     - Initializes the GCQ as an SRB
     - Schedules the SRB to execute ISGBSR
- Returns to ISGBTC, exception-handling task

ISGBTC Ring processing task mode controller, Exception-handling task

- Processes another RSL, or waits on its ECB (GVTXBECB) (Note that the exception-handling task does not wait for a send completion or arrival of a RSAIRCD.)

ISGBSR RSA send/receive routine
EP-ISGBSRRI

- Copies the status of this system from the RSVENTY table to the buffer for this RSL
- Sends the RSAIRCD using the GCQ and buffer for this RSL
- Exits to the dispatcher

Enter — From CTC processing

ISGBSR RSA send/receive routine
EP-ISGBSRRI

- Receives the send completion
- Gives control of the GCQ and buffer for this RSL back to CTC processing
- Exits to the dispatcher

Figure 5-74. Module Flow for Ring Processing - Send a RSAIRCD (Requested by ISGBTC)

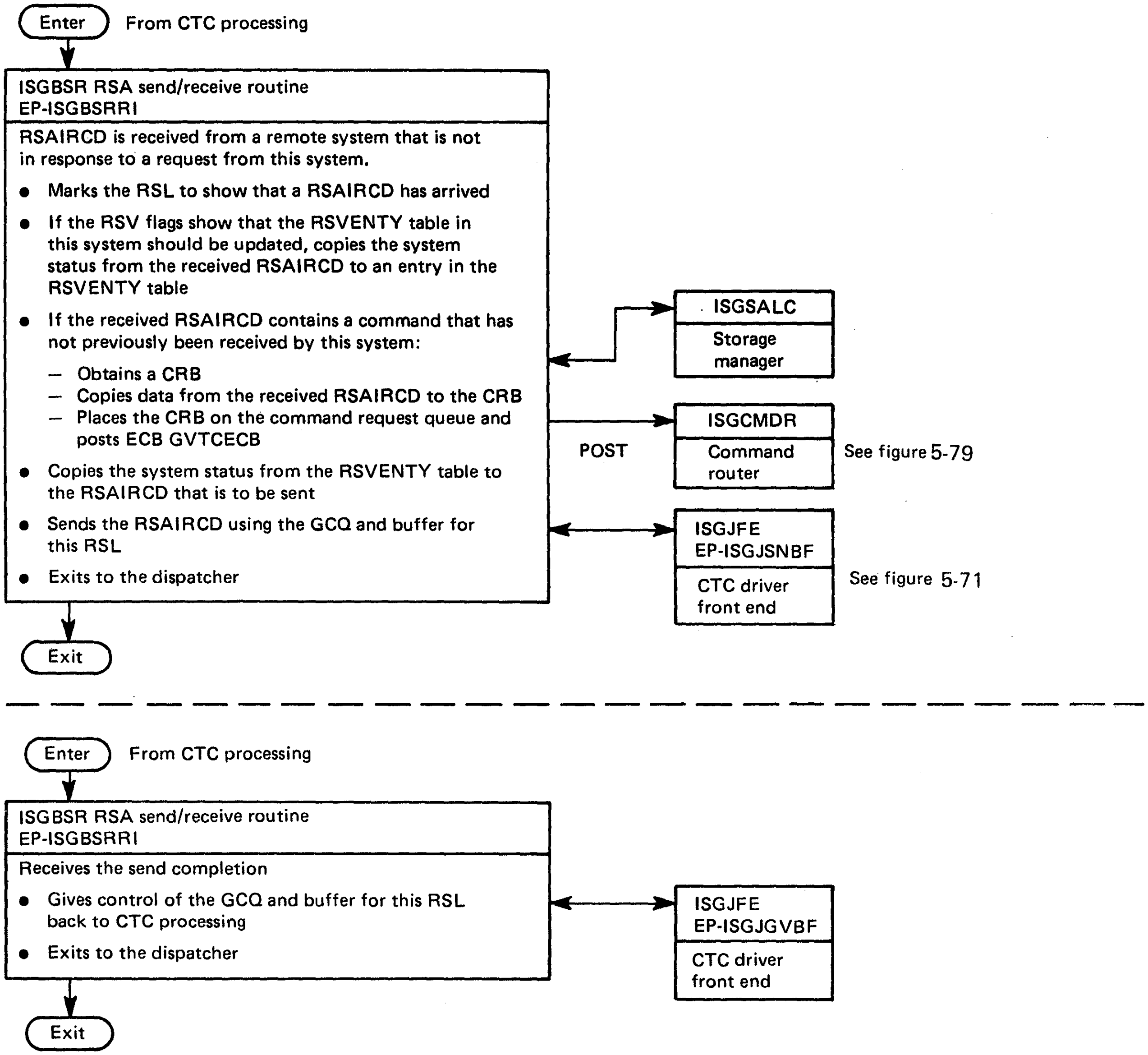
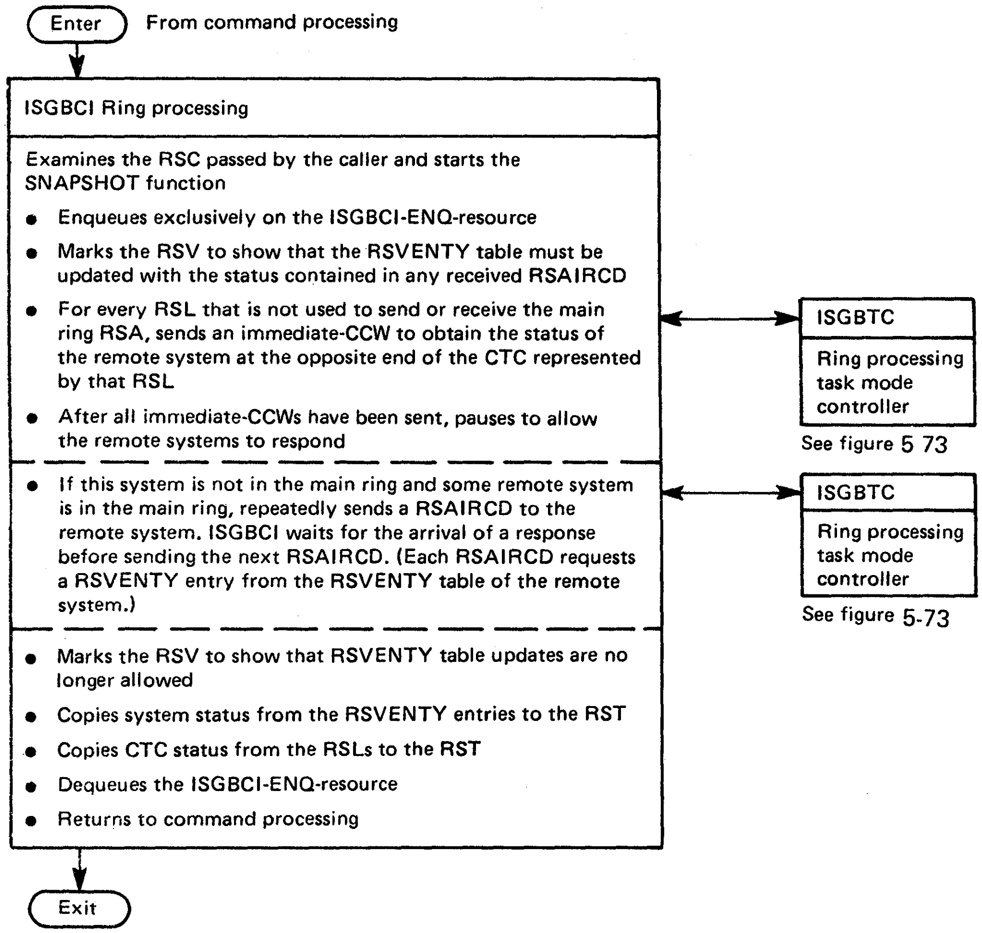

Enter
From CTC processing
ISGBSR RSA send/receive routine
EP-ISGBSRRI
RSAIRCD is received from a remote system that is not in response to a request from this system.
• Marks the RSL to show that a RSAIRCD has arrived
• If the RSV flags show that the RSVENTY table in this system should be updated, copies the system status from the received RSAIRCD to an entry in the RSVENTY table
• If the received RSAIRCD contains a command that has not previously been received by this system:
– Obtains a CRB
– Copies data from the received RSAIRCD to the CRB
– Places the CRB on the command request queue and posts ECB GVTCECB
• Copies the system status from the RSVENTY table to the RSAIRCD that is to be sent
• Sends the RSAIRCD using the GCQ and buffer for this RSL
• Exits to the dispatcher
Exit
ISGSALC
Storage manager
POST
ISGCMDR
Command router
See figure 5-79
ISGJFE
EP-ISGJSNBF
CTC driver front end
See figure 5-71
Enter
From CTC processing
ISGBSR RSA send/receive routine
EP-ISGBSRRI
Receives the send completion
• Gives control of the GCQ and buffer for this RSL back to CTC processing
• Exits to the dispatcher
Exit
ISGJFE
EP-ISGJGVBF
CTC driver front end

Figure 5-75. Module Flow for Ring Processing - Handle Arrival of RSAIRCD (Not Requested by This System)



Enter From command processing

ISGBCI Ring processing

Examines the RSC passed by the caller and starts the SNAPSHOT function

- Enqueues exclusively on the ISGBCI-ENQ-resource
- Marks the RSV to show that the RSVENTY table must be updated with the status contained in any received RSAIRCD
- For every RSL that is not used to send or receive the main ring RSA, sends an immediate-CCW to obtain the status of the remote system at the opposite end of the CTC represented by that RSL
- After all immediate-CCWs have been sent, pauses to allow the remote systems to respond

ISGBTC

Ring processing task mode controller

See figure 5 73

- If this system is not in the main ring and some remote system is in the main ring, repeatedly sends a RSAIRCD to the remote system. ISGBCI waits for the arrival of a response before sending the next RSAIRCD. (Each RSAIRCD requests a RSVENTY entry from the RSVENTY table of the remote system.)

ISGBTC

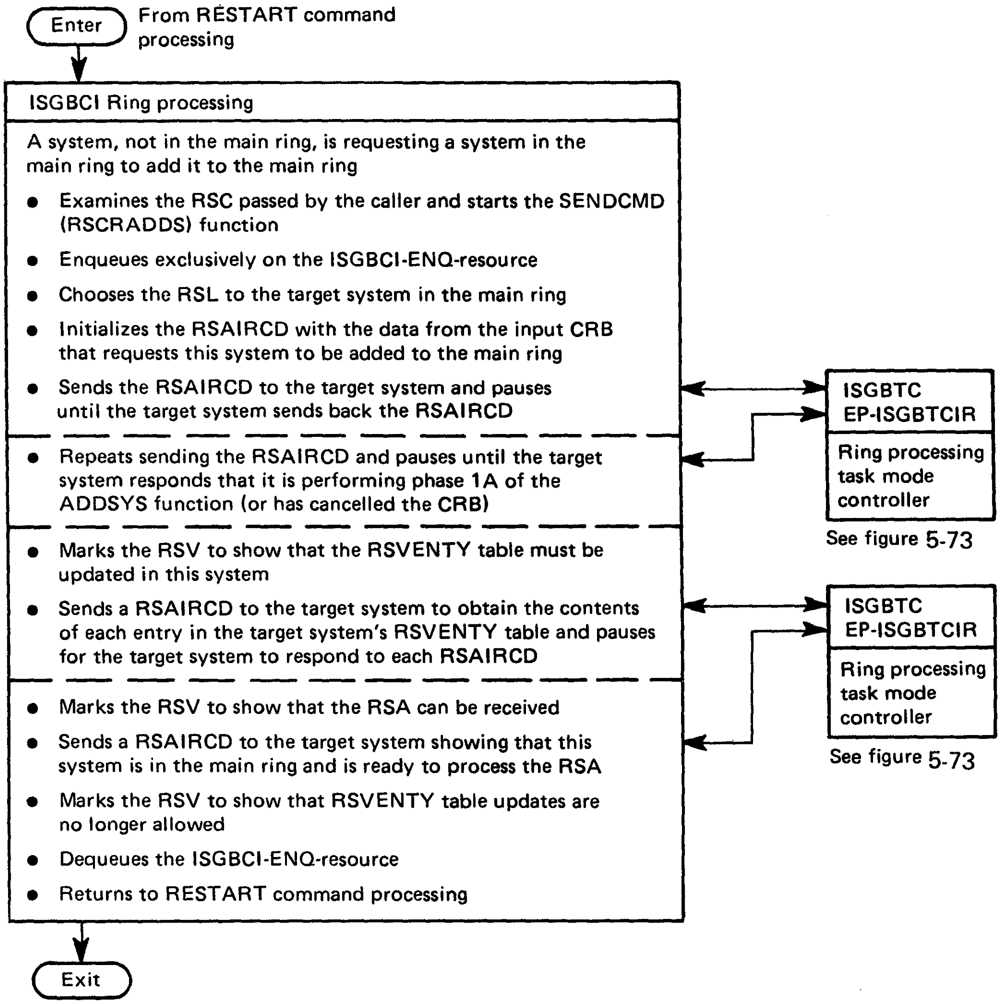
Ring processing task mode controller

See figure 5-73

- Marks the RSV to show that RSVENTY table updates are no longer allowed
- Copies system status from the RSVENTY entries to the RST
- Copies CTC status from the RSLs to the RST
- Dequeues the ISGBCI-ENQ-resource
- Returns to command processing

Exit

Figure 5-76. Module Flow for Ring Processing - SNAPSHOT Function



ISGBCI Ring processing

A system, not in the main ring, is requesting a system in the main ring to add it to the main ring

- Examines the RSC passed by the caller and starts the SENDCMD (RSCRADDS) function
- Enqueues exclusively on the ISGBCI-ENQ-resource
- Chooses the RSL to the target system in the main ring
- Initializes the RSAIRCD with the data from the input CRB that requests this system to be added to the main ring
- Sends the RSAIRCD to the target system and pauses until the target system sends back the RSAIRCD

- Repeats sending the RSAIRCD and pauses until the target system responds that it is performing phase 1A of the ADDSYS function (or has cancelled the CRB)

- Marks the RSV to show that the RSVENTY table must be updated in this system
- Sends a RSAIRCD to the target system to obtain the contents of each entry in the target system's RSVENTY table and pauses for the target system to respond to each RSAIRCD

- Marks the RSV to show that the RSA can be received
- Sends a RSAIRCD to the target system showing that this system is in the main ring and is ready to process the RSA
- Marks the RSV to show that RSVENTY table updates are no longer allowed
- Dequeues the ISGBCI-ENQ-resource
- Returns to RESTART command processing

ISGBTC
EP-ISGBTCIR
Ring processing task mode controller
See figure 5-73
ISGBTC
EP-ISGBTCIR
Ring processing task mode controller
See figure 5-73
Exit

Figure 5-77. Module Flow for Ring Processing - SENDCMD (RSCRADDS) Function

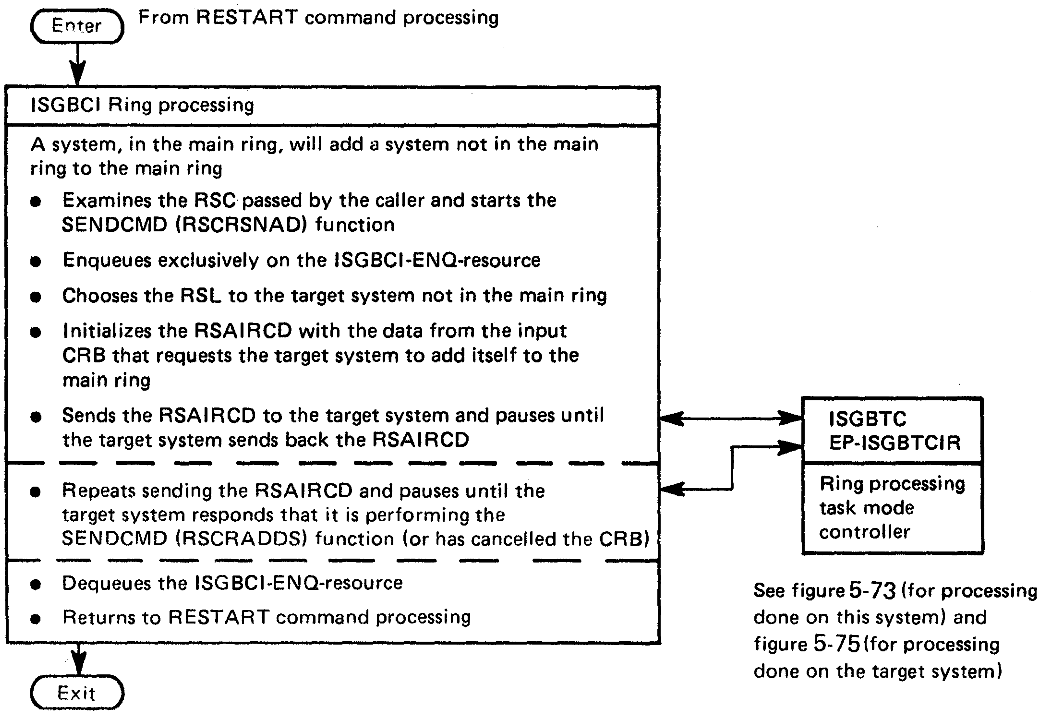
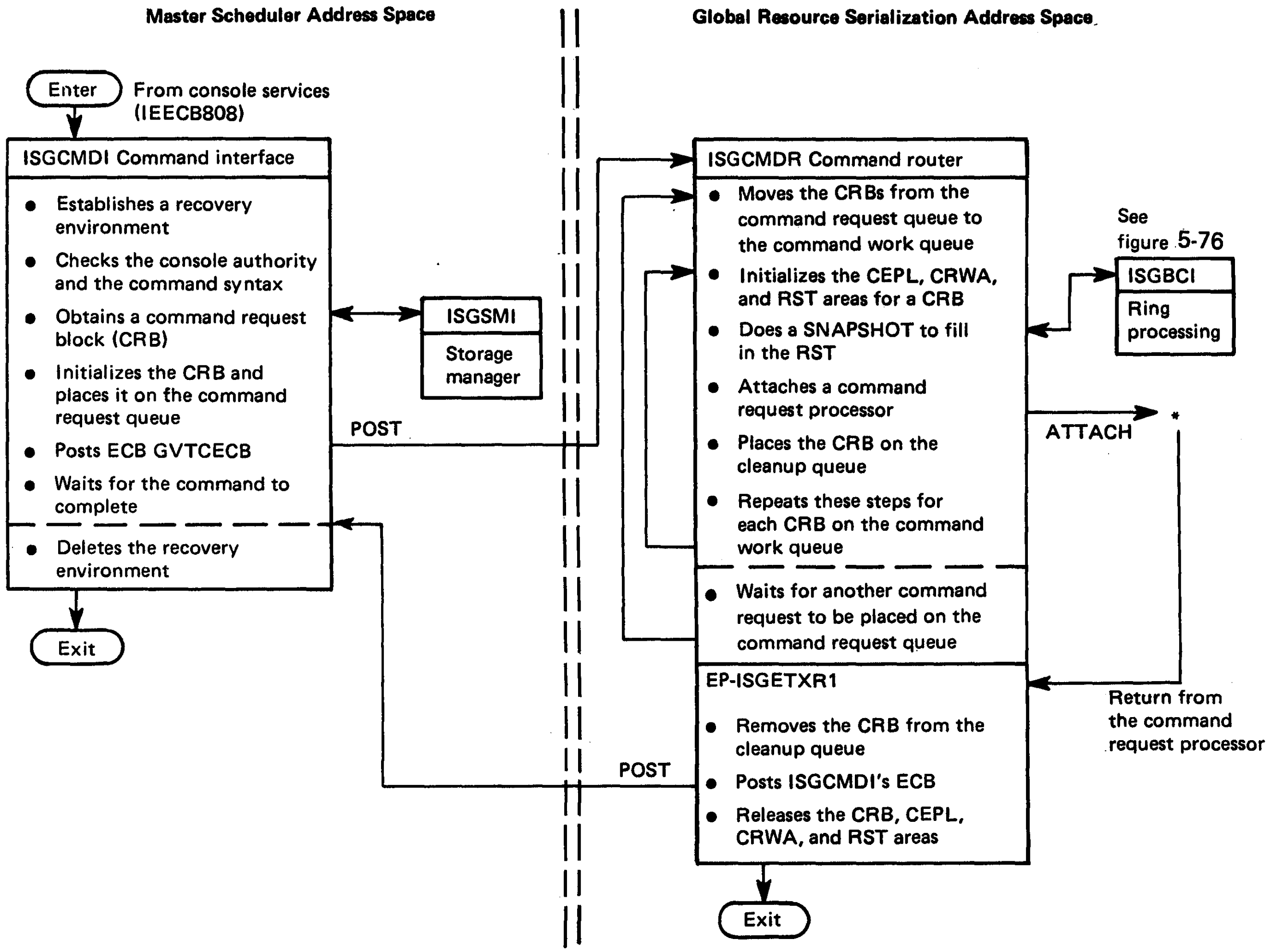

Enter
From RESTART command processing
ISGBCI Ring processing
A system, in the main ring, will add a system not in the main ring to the main ring
● Examines the RSC passed by the caller and starts the SENDCMD (RSCRSNAD) function
● Enqueues exclusively on the ISGBCI-ENQ-resource
● Chooses the RSL to the target system not in the main ring
● Initializes the RSAIRCD with the data from the input CRB that requests the target system to add itself to the main ring
● Sends the RSAIRCD to the target system and pauses until the target system sends back the RSAIRCD
● Repeats sending the RSAIRCD and pauses until the target system responds that it is performing the SENDCMD (RSCRADDS) function (or has cancelled the CRB)
● Dequeues the ISGBCI-ENQ-resource
● Returns to RESTART command processing
Exit
ISGBTC
EP-ISGBTCIR
Ring processing task mode controller
See figure 5-73 (for processing done on this system) and figure 5-75 (for processing done on the target system)

Figure 5-78. Module Flow for Ring Processing - SENDCMD (RSCRSNAD) Function


Master Scheduler Address Space
Global Resource Serialization Address Space
Enter
From console services (IEECB808)
ISGCMDI Command interface
• Establishes a recovery environment
• Checks the console authority and the command syntax
• Obtains a command request block (CRB)
• Initializes the CRB and places it on the command request queue
• Posts ECB GVTCECB
• Waits for the command to complete
• Deletes the recovery environment
Exit
ISGSMI
Storage manager
POST
ISGCMDR Command router
• Moves the CRBs from the command request queue to the command work queue
• Initializes the CEPL, CRWA, and RST areas for a CRB
• Does a SNAPSHOT to fill in the RST
• Attaches a command request processor
• Places the CRB on the cleanup queue
• Repeats these steps for each CRB on the command work queue
• Waits for another command request to be placed on the command request queue
See figure 5-76
ISGBCI
Ring processing
ATTACH
*
EP-ISGETXR1
• Removes the CRB from the cleanup queue
• Posts ISGCMDI's ECB
• Releases the CRB, CEPL, CRWA, and RST areas
Return from the command request processor
POST
Exit

***Command request processors**

ISGCDSP – DISPLAY GRS (figure 5-80)
ISGCPRG – VARY GRS(x), PURGE (figure 5-81)
ISGCQSC – VARY GRS(x), QUIESCE (figures5-82and 5-83)
ISGCRST – VARY GRS(x), RESTART (figures 5-84, 5-85, and 5-86)
ISGMSG00 – Asynchronous message request

Figure 5-79. Module Flow for Command Initialization and Cleanup

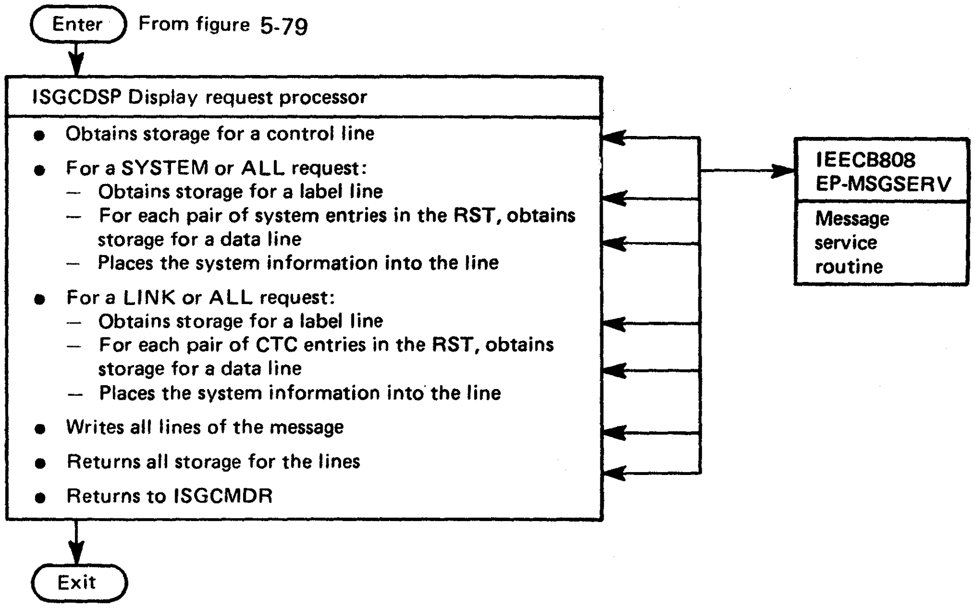

Enter
From figure 5-79
ISGCDSP Display request processor
• Obtains storage for a control line
• For a SYSTEM or ALL request:
– Obtains storage for a label line
– For each pair of system entries in the RST, obtains storage for a data line
– Places the system information into the line
• For a LINK or ALL request:
– Obtains storage for a label line
– For each pair of CTC entries in the RST, obtains storage for a data line
– Places the system information into the line
• Writes all lines of the message
• Returns all storage for the lines
• Returns to ISGCMDR
Exit
IEECB808
EP-MSGSERV
Message service routine

Figure 5-80. Module Flow for DISPLAY GRS

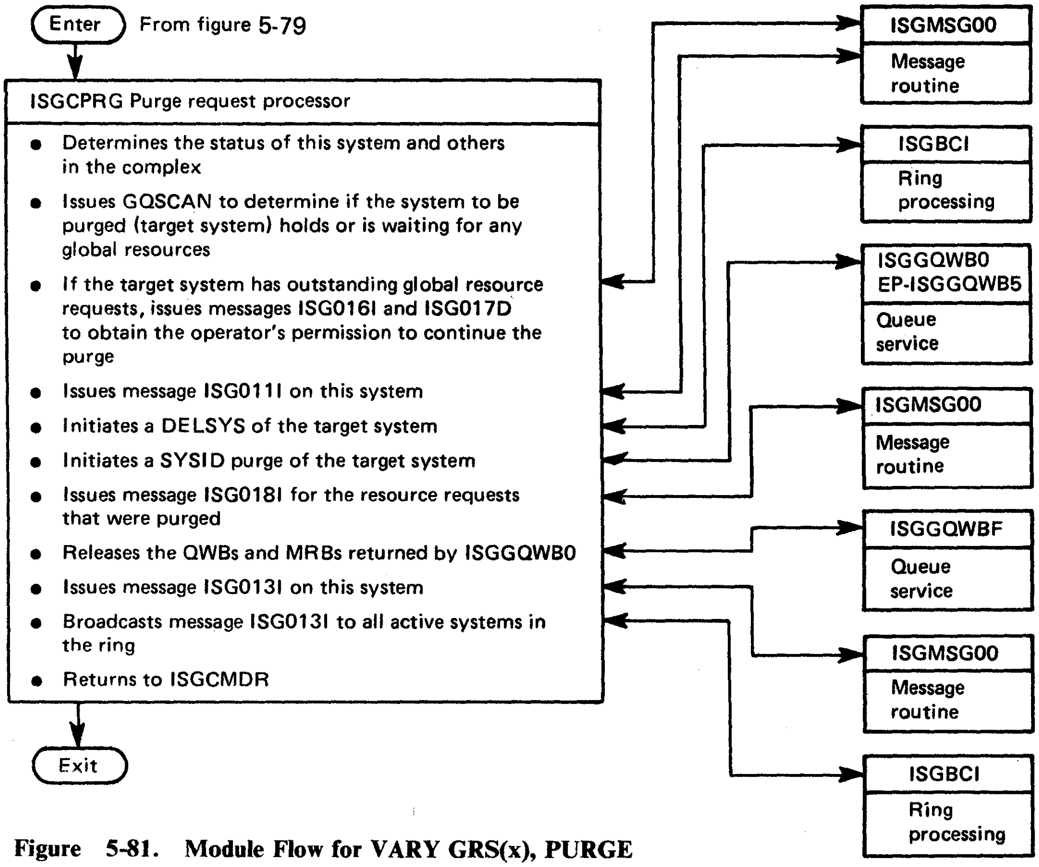

Enter
From figure 5-79
ISGCPRG Purge request processor
• Determines the status of this system and others in the complex
• Issues GQSCAN to determine if the system to be purged (target system) holds or is waiting for any global resources
• If the target system has outstanding global resource requests, issues messages ISG016I and ISG017D to obtain the operator's permission to continue the purge
• Issues message ISG011I on this system
• Initiates a DELSYS of the target system
• Initiates a SYSID purge of the target system
• Issues message ISG018I for the resource requests that were purged
• Releases the QWBs and MRBs returned by ISGGQWB0
• Issues message ISG013I on this system
• Broadcasts message ISG013I to all active systems in the ring
• Returns to ISGCMDR
Exit
ISGMSG00
Message routine
ISGBCI
Ring processing
ISGGQWB0
EP-ISGGQWB5
Queue service
ISGMSG00
Message routine
ISGGQWBF
Queue service
ISGMSG00
Message routine
ISGBCI
Ring processing

Figure 5-81. Module Flow for VARY GRS(x), PURGE

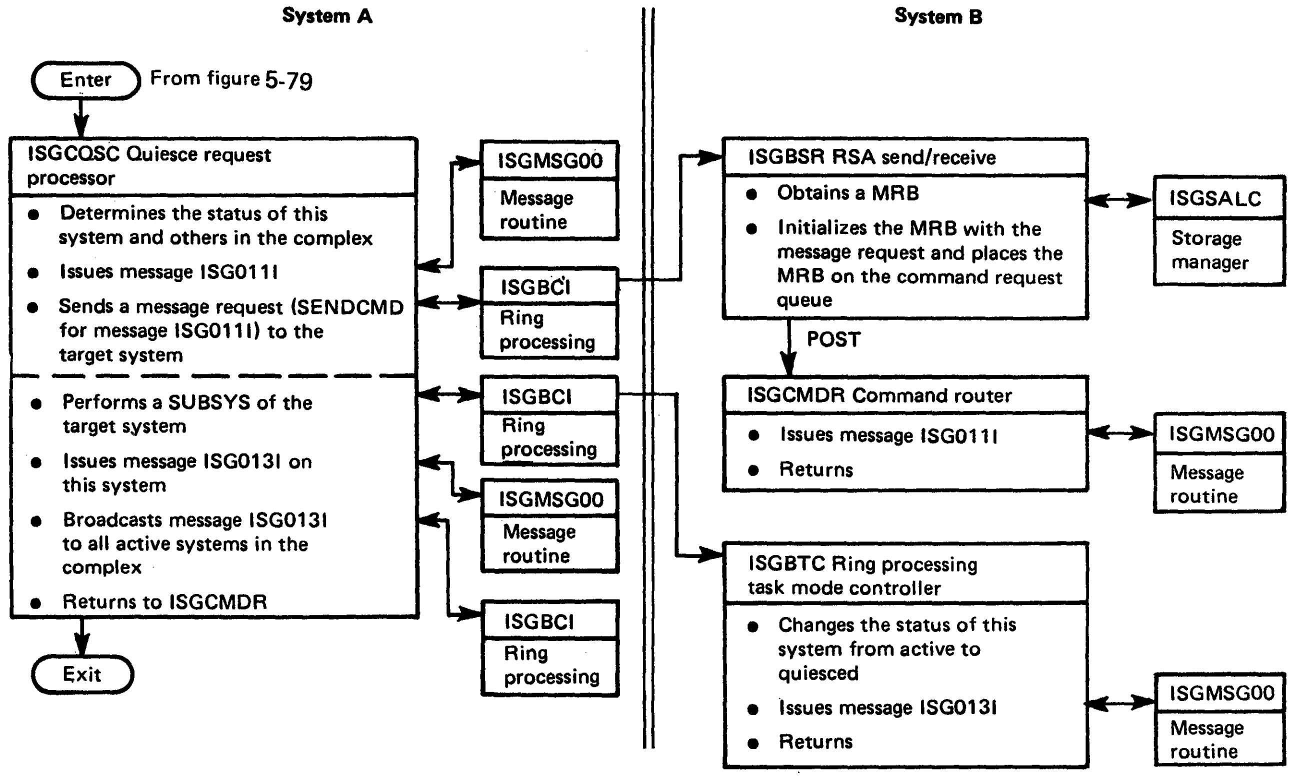

System A
System B
Enter
From figure 5-79
ISGCQSC Quiesce request processor
Determines the status of this system and others in the complex
Issues message ISG011I
Sends a message request (SENDCMD for message ISG011I) to the target system
Performs a SUBSYS of the target system
Issues message ISG013I on this system
Broadcasts message ISG013I to all active systems in the complex
Returns to ISGCMDR
Exit
ISGMSG00
Message routine
ISGBCI
Ring processing
ISGBCI
Ring processing
ISGMSG00
Message routine
ISGBCI
Ring processing
ISGBSR RSA send/receive
Obtains a MRB
Initializes the MRB with the message request and places the MRB on the command request queue
ISGSALC
Storage manager
POST
ISGCMDR Command router
Issues message ISG011I
Returns
ISGMSG00
Message routine
ISGBTC Ring processing task mode controller
Changes the status of this system from active to quiesced
Issues message ISG013I
Returns
ISGMSG00
Message routine

Figure 5-82. Module Flow for VARY GRS(x), QUIESCE to Another System

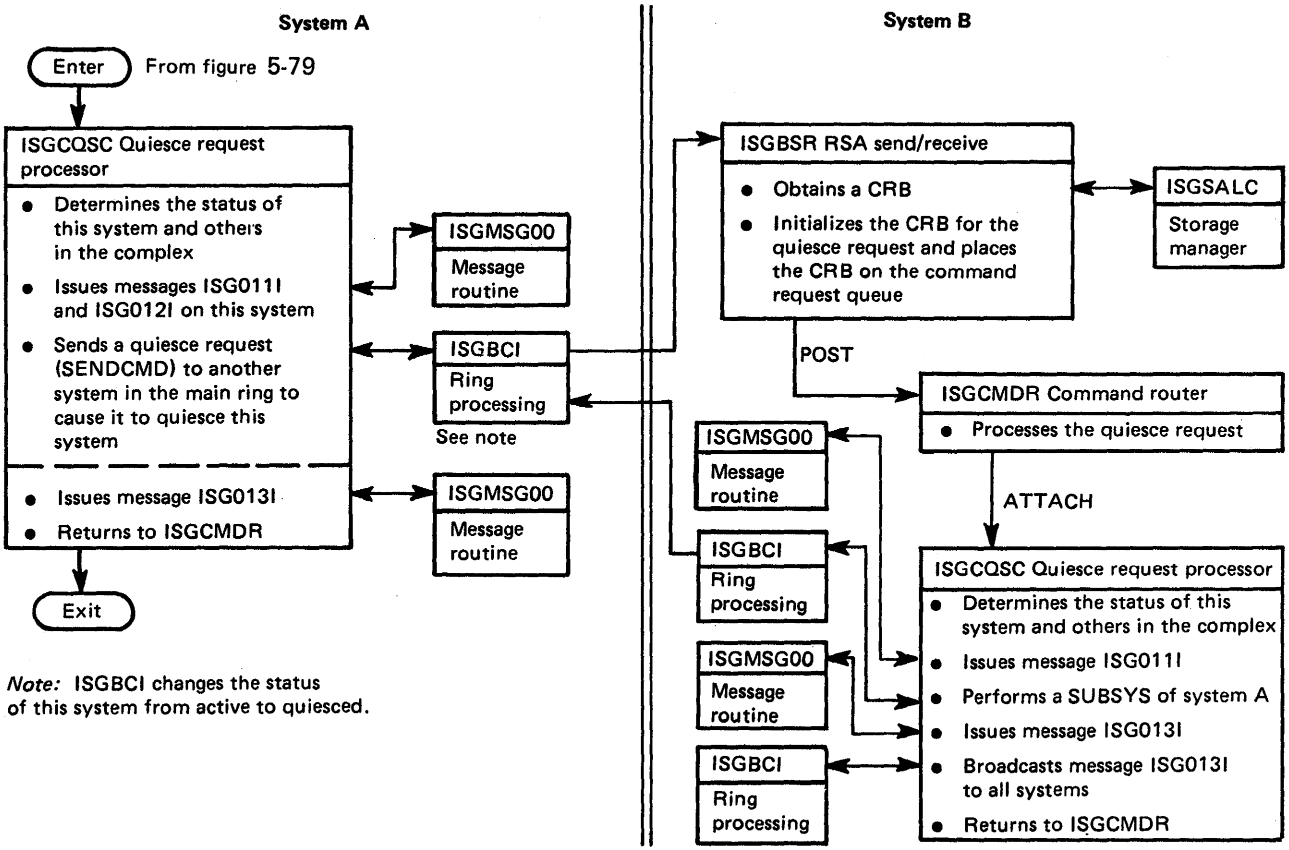

System A
Enter
From figure 5-79
ISGCQSC Quiesce request processor
Determines the status of this system and others in the complex
Issues messages ISG011I and ISG012I on this system
Sends a quiesce request (SENDCMD) to another system in the main ring to cause it to quiesce this system
Issues message ISG013I
Returns to ISGCMDR
Exit
ISGMSG00
Message routine
ISGBCI
Ring processing
See note
ISGMSG00
Message routine
Note: ISGBCI changes the status of this system from active to quiesced.
System B
ISGBSR RSA send/receive
Obtains a CRB
Initializes the CRB for the quiesce request and places the CRB on the command request queue
ISGSALC
Storage manager
POST
ISGCMDR Command router
Processes the quiesce request
ATTACH
ISGMSG00
Message routine
ISGBCI
Ring processing
ISGMSG00
Message routine
ISGBCI
Ring processing
ISGCQSC Quiesce request processor
Determines the status of this system and others in the complex
Issues message ISG011I
Performs a SUBSYS of system A
Issues message ISG013I
Broadcasts message ISG013I to all systems
Returns to ISGCMDR

Figure 5-83. Module Flow for VARY GRS(x), QUIESCE by a System to Quiesce Itself

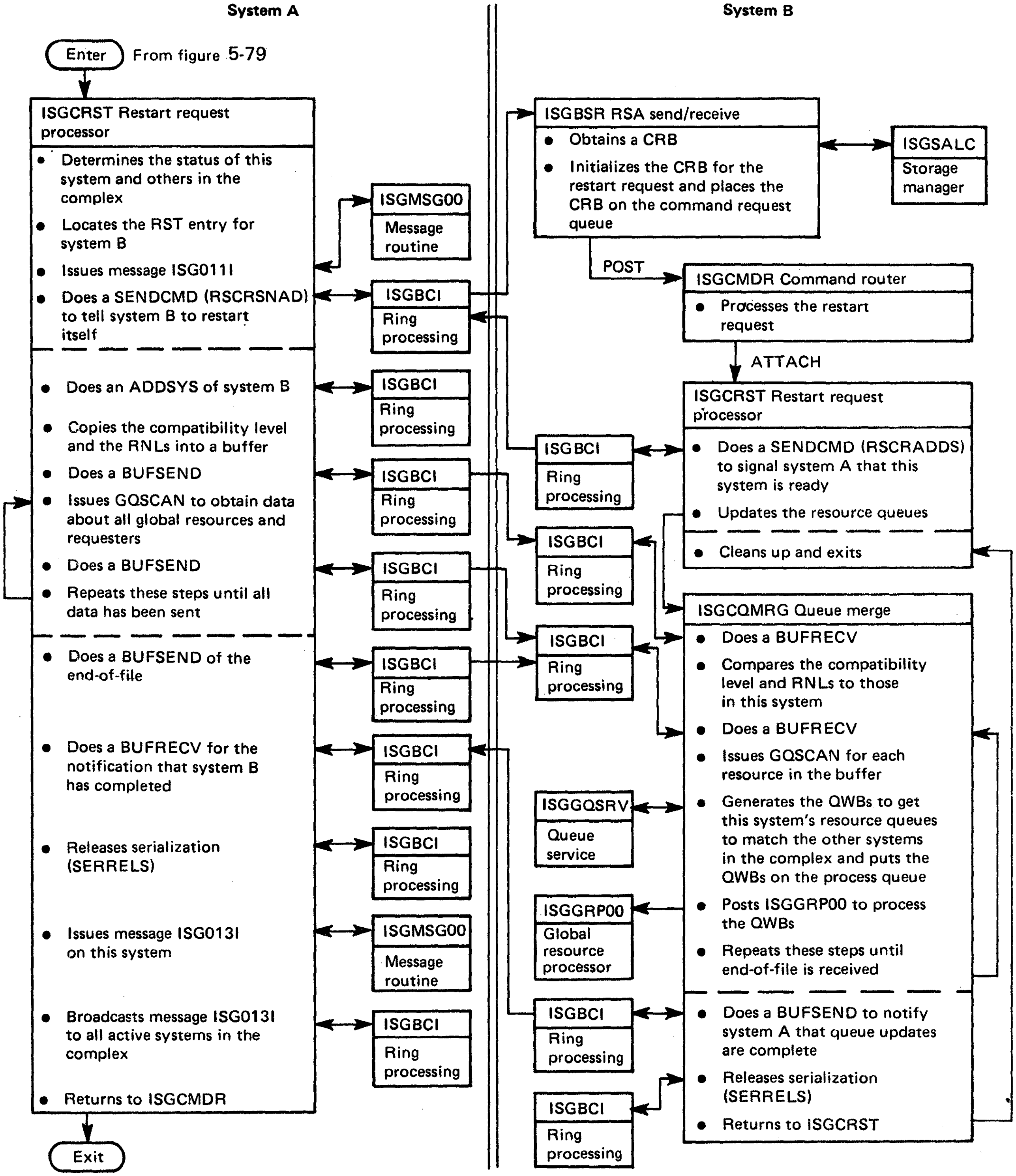

System A
System B
Enter
From figure 5-79
ISGCRST Restart request processor
• Determines the status of this system and others in the complex
• Locates the RST entry for system B
• Issues message ISG011I
• Does a SENDCMD (RSCRSNAD) to tell system B to restart itself
• Does an ADDSYS of system B
• Copies the compatibility level and the RNLs into a buffer
• Does a BUFSEND
• Issues GQSCAN to obtain data about all global resources and requesters
• Does a BUFSEND
• Repeats these steps until all data has been sent
• Does a BUFSEND of the end-of-file
• Does a BUFRECV for the notification that system B has completed
• Releases serialization (SERRELS)
• Issues message ISG013I on this system
• Broadcasts message ISG013I to all active systems in the complex
• Returns to ISGCMDR
Exit
ISGMSG00
Message routine
ISGBCI
Ring processing
ISGBCI
Ring processing
ISGBCI
Ring processing
ISGBCI
Ring processing
ISGBCI
Ring processing
ISGBCI
Ring processing
ISGBCI
Ring processing
ISGMSG00
Message routine
ISGBCI
Ring processing
ISGBSR RSA send/receive
• Obtains a CRB
• Initializes the CRB for the restart request and places the CRB on the command request queue
ISGSALC
Storage manager
POST
ISGCMDR Command router
• Processes the restart request
ATTACH
ISGCRST Restart request processor
• Does a SENDCMD (RSCRADDS) to signal system A that this system is ready
• Updates the resource queues
• Cleans up and exits
ISGBCI
Ring processing
ISGBCI
Ring processing
ISGBCI
Ring processing
ISGCQMRG Queue merge
• Does a BUFRECV
• Compares the compatibility level and RNLs to those in this system
• Does a BUFRECV
• Issues GQSCAN for each resource in the buffer
• Generates the QWBs to get this system's resource queues to match the other systems in the complex and puts the QWBs on the process queue
• Posts ISGGRP00 to process the QWBs
• Repeats these steps until end-of-file is received
• Does a BUFSEND to notify system A that queue updates are complete
• Releases serialization (SERRELS)
• Returns to ISGCRST
ISGGQSRV
Queue service
ISGGRP00
Global resource processor
ISGBCI
Ring processing
ISGBCI
Ring processing

Figure 5-84. Module Flow for VARY GRS(x), RESTART to Restart Another System

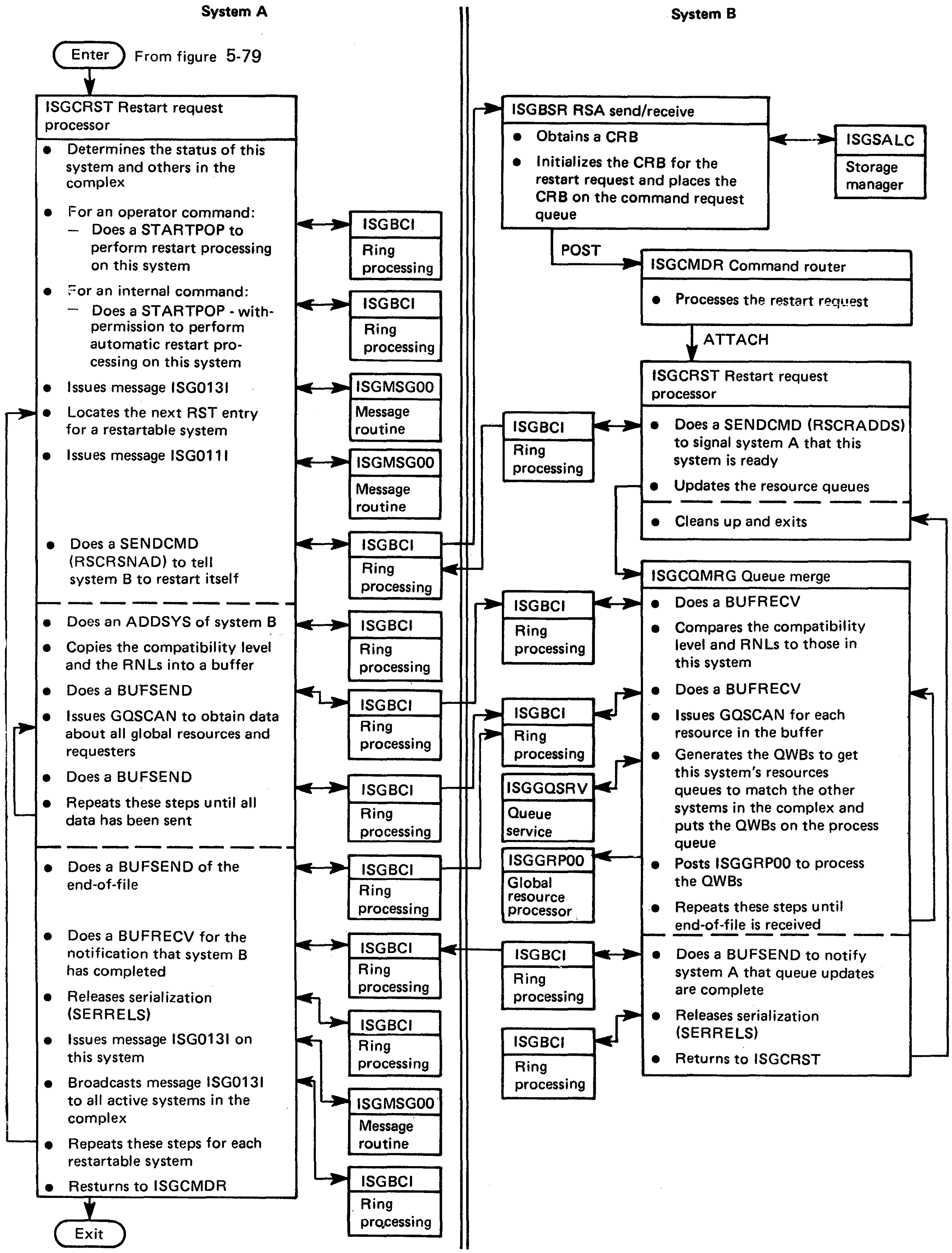

System A
System B
Enter
From figure 5-79
ISGCRST Restart request processor
● Determines the status of this system and others in the complex
● For an operator command:
— Does a STARTPOP to perform restart processing on this system
● For an internal command:
— Does a STARTPOP - with-permission to perform automatic restart processing on this system
● Issues message ISG013I
● Locates the next RST entry for a restartable system
● Issues message ISG011I
● Does a SENDCMD (RSCRSNAD) to tell system B to restart itself
● Does an ADDSYS of system B
● Copies the compatibility level and the RNLs into a buffer
● Does a BUFSEND
● Issues GQSCAN to obtain data about all global resources and requesters
● Does a BUFSEND
● Repeats these steps until all data has been sent
● Does a BUFSEND of the end-of-file
● Does a BUFRECV for the notification that system B has completed
● Releases serialization (SERRELS)
● Issues message ISG013I on this system
● Broadcasts message ISG013I to all active systems in the complex
● Repeats these steps for each restartable system
● Resturns to ISGCMDR
Exit
ISGBCI Ring processing
ISGBCI Ring processing
ISGMSG00 Message routine
ISGMSG00 Message routine
ISGBCI Ring processing
ISGBCI Ring processing
ISGBCI Ring processing
ISGBCI Ring processing
ISGBCI Ring processing
ISGBCI Ring processing
ISGBCI Ring processing
ISGMSG00 Message routine
ISGBCI Ring processing
ISGBSR RSA send/receive
● Obtains a CRB
● Initializes the CRB for the restart request and places the CRB on the command request queue
ISGSALC Storage manager
POST
ISGCMDR Command router
● Processes the restart request
ATTACH
ISGCRST Restart request processor
● Does a SENDCMD (RSCRADDS) to signal system A that this system is ready
● Updates the resource queues
● Cleans up and exits
ISGCQMRG Queue merge
● Does a BUFRECV
● Compares the compatibility level and RNLs to those in this system
● Does a BUFRECV
● Issues GQSCAN for each resource in the buffer
● Generates the QWBs to get this system's resources queues to match the other systems in the complex and puts the QWBs on the process queue
● Posts ISGGRP00 to process the QWBs
● Repeats these steps until end-of-file is received
● Does a BUFSEND to notify system A that queue updates are complete
● Releases serialization (SERRELS)
● Returns to ISGCRST
ISGBCI Ring processing
ISGBCI Ring processing
ISGBCI Ring processing
ISGGQSRV Queue service
ISGGRP00 Global resource processor
ISGBCI Ring processing
ISGBCI Ring processing

Figure 5-85. Module Flow for VARY GRS(ALL), RESTART to Restart All Systems

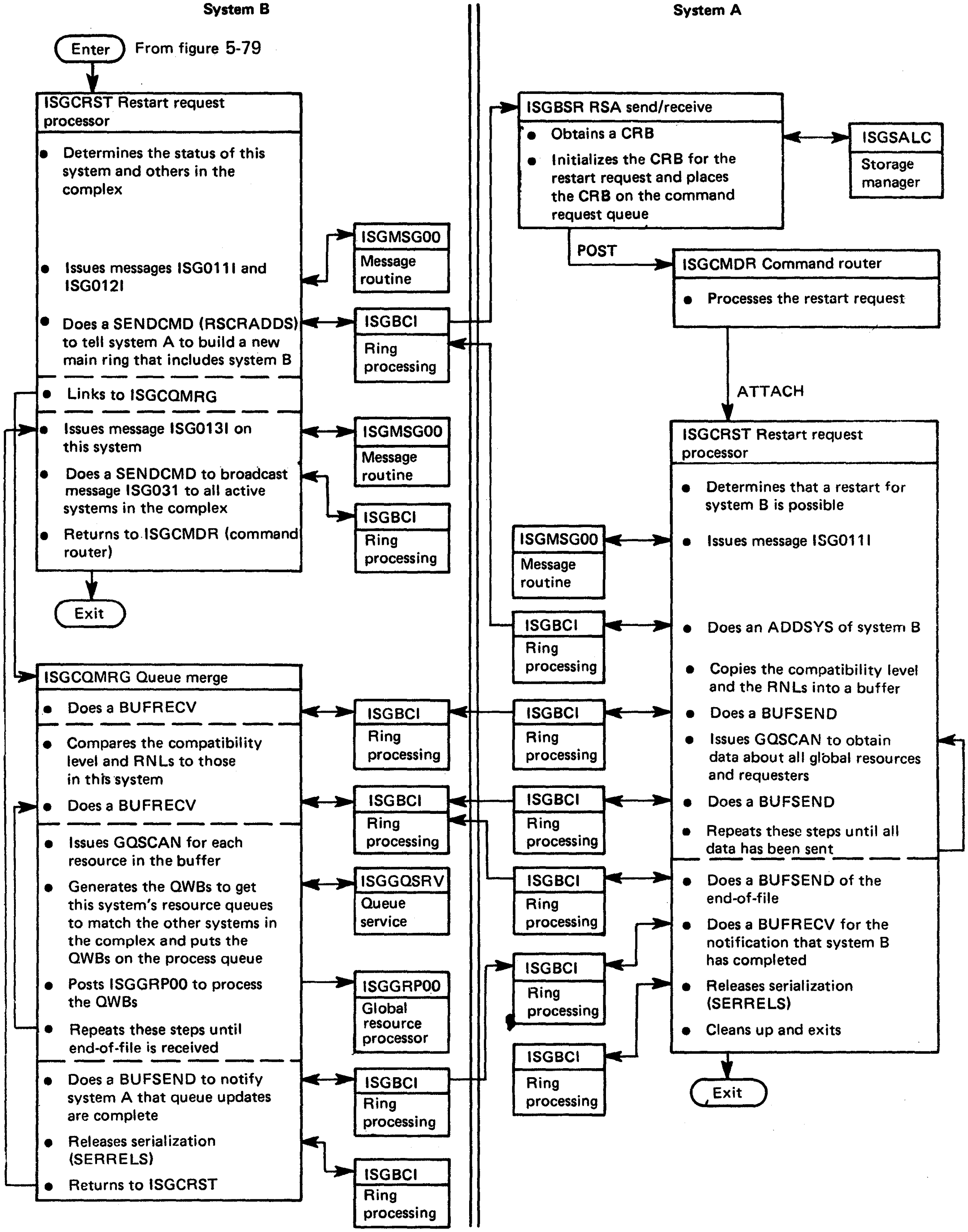

System B
System A
Enter
From figure 5-79
ISGCRST Restart request processor
● Determines the status of this system and others in the complex
● Issues messages ISG011I and ISG012I
● Does a SENDCMD (RSCRADDS) to tell system A to build a new main ring that includes system B
● Links to ISGCQMRG
● Issues message ISG013I on this system
● Does a SENDCMD to broadcast message ISG031 to all active systems in the complex
● Returns to ISGCMDR (command router)
Exit
ISGMSG00
Message routine
ISGBCI
Ring processing
ISGMSG00
Message routine
ISGBCI
Ring processing
ISGCQMRG Queue merge
● Does a BUFRECV
● Compares the compatibility level and RNLs to those in this system
● Does a BUFRECV
● Issues GQSCAN for each resource in the buffer
● Generates the QWBs to get this system's resource queues to match the other systems in the complex and puts the QWBs on the process queue
● Posts ISGGRP00 to process the QWBs
● Repeats these steps until end-of-file is received
● Does a BUFSEND to notify system A that queue updates are complete
● Releases serialization (SERRELS)
● Returns to ISGCRST
ISGBCI
Ring processing
ISGBCI
Ring processing
ISGGQSRV
Queue service
ISGGRP00
Global resource processor
ISGBCI
Ring processing
ISGBCI
Ring processing
ISGBSR RSA send/receive
● Obtains a CRB
● Initializes the CRB for the restart request and places the CRB on the command request queue
ISGSALC
Storage manager
POST
ISGCMDR Command router
● Processes the restart request
ATTACH
ISGCRST Restart request processor
● Determines that a restart for system B is possible
● Issues message ISG011I
● Does an ADDSYS of system B
● Copies the compatibility level and the RNLs into a buffer
● Does a BUFSEND
● Issues GQSCAN to obtain data about all global resources and requesters
● Does a BUFSEND
● Repeats these steps until all data has been sent
● Does a BUFSEND of the end-of-file
● Does a BUFRECV for the notification that system B has completed
● Releases serialization (SERRELS)
● Cleans up and exits
Exit
ISGMSG00
Message routine
ISGBCI
Ring processing
ISGBCI
Ring processing
ISGBCI
Ring processing
ISGBCI
Ring processing
ISGBCI
Ring processing
ISGBCI
Ring processing

Figure 5-86. Module Flow for VARY GRS(x), RESTART by a System Not in the Main Ring

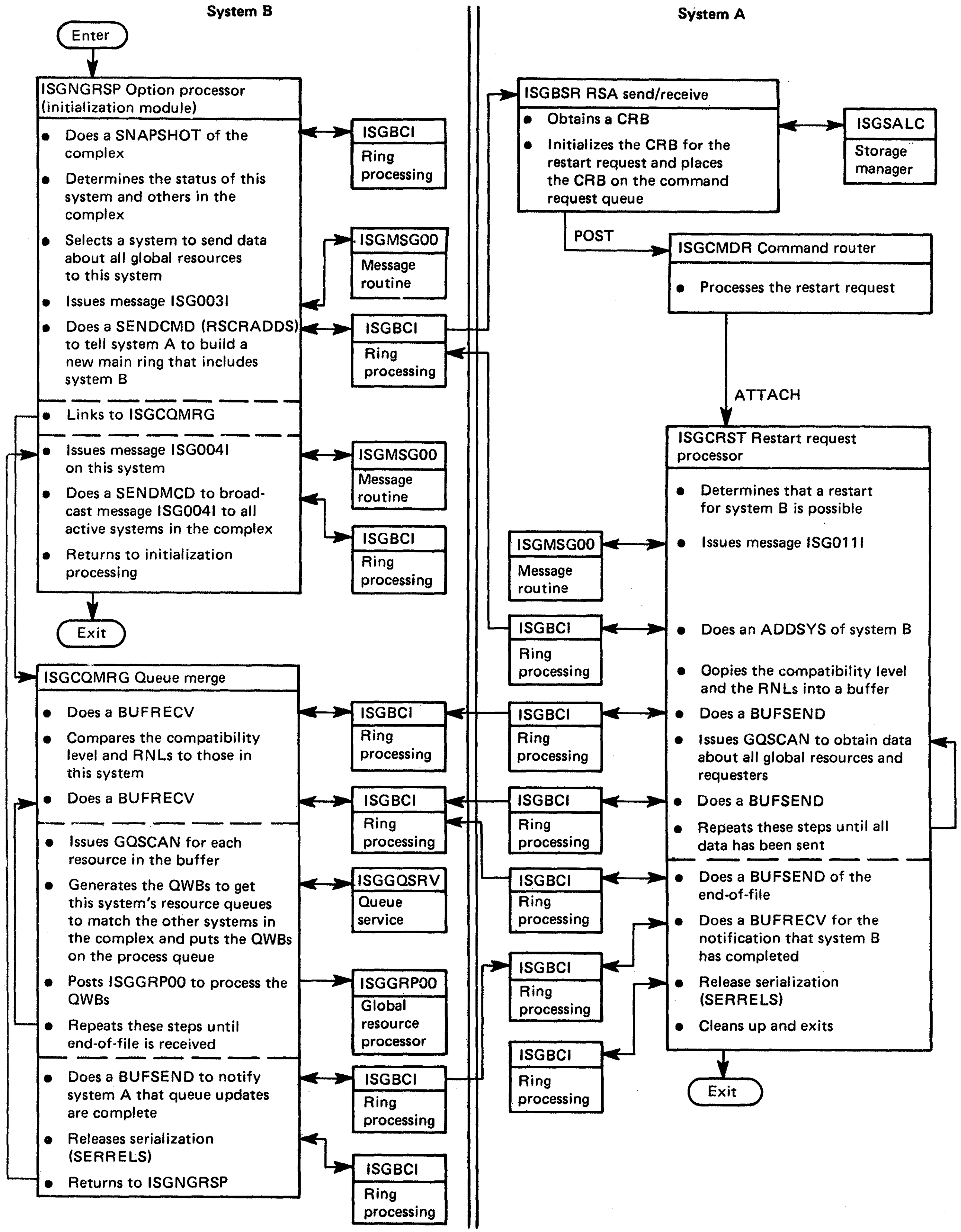

System B
System A
Enter
ISGNGRSP Option processor (initialization module)
• Does a SNAPSHOT of the complex
• Determines the status of this system and others in the complex
• Selects a system to send data about all global resources to this system
• Issues message ISG003I
• Does a SENDCMD (RSCRADDS) to tell system A to build a new main ring that includes system B
• Links to ISGCQMRG
• Issues message ISG004I on this system
• Does a SENDMCD to broad-cast message ISG004I to all active systems in the complex
• Returns to initialization processing
Exit
ISGBCI
Ring processing
ISGMSG00
Message routine
ISGBCI
Ring processing
ISGMSG00
Message routine
ISGBCI
Ring processing
ISGCQMRG Queue merge
• Does a BUFRECV
• Compares the compatibility level and RNLs to those in this system
• Does a BUFRECV
• Issues GQSCAN for each resource in the buffer
• Generates the QWBs to get this system's resource queues to match the other systems in the complex and puts the QWBs on the process queue
• Posts ISGGRP00 to process the QWBs
• Repeats these steps until end-of-file is received
• Does a BUFSEND to notify system A that queue updates are complete
• Releases serialization (SERRELS)
• Returns to ISGNGRSP
ISGBCI
Ring processing
ISGBCI
Ring processing
ISGGQSRV
Queue service
ISGGRP00
Global resource processor
ISGBCI
Ring processing
ISGBCI
Ring processing
ISGBSR RSA send/receive
• Obtains a CRB
• Initializes the CRB for the restart request and places the CRB on the command request queue
ISGSALC
Storage manager
POST
ISGCMDR Command router
• Processes the restart request
ATTACH
ISGCRST Restart request processor
• Determines that a restart for system B is possible
• Issues message ISG011I
• Does an ADDSYS of system B
• Copies the compatibility level and the RNLs into a buffer
• Does a BUFSEND
• Issues GQSCAN to obtain data about all global resources and requesters
• Does a BUFSEND
• Repeats these steps until all data has been sent
• Does a BUFSEND of the end-of-file
• Does a BUFRECV for the notification that system B has completed
• Release serialization (SERRELS)
• Cleans up and exits
Exit
ISGMSG00
Message routine
ISGBCI
Ring processing
ISGBCI
Ring processing
ISGBCI
Ring processing
ISGBCI
Ring processing
ISGBCI
Ring processing
ISGBCI
Ring processing

Figure 5-87. Module Flow for Join Processing at Initialization Time

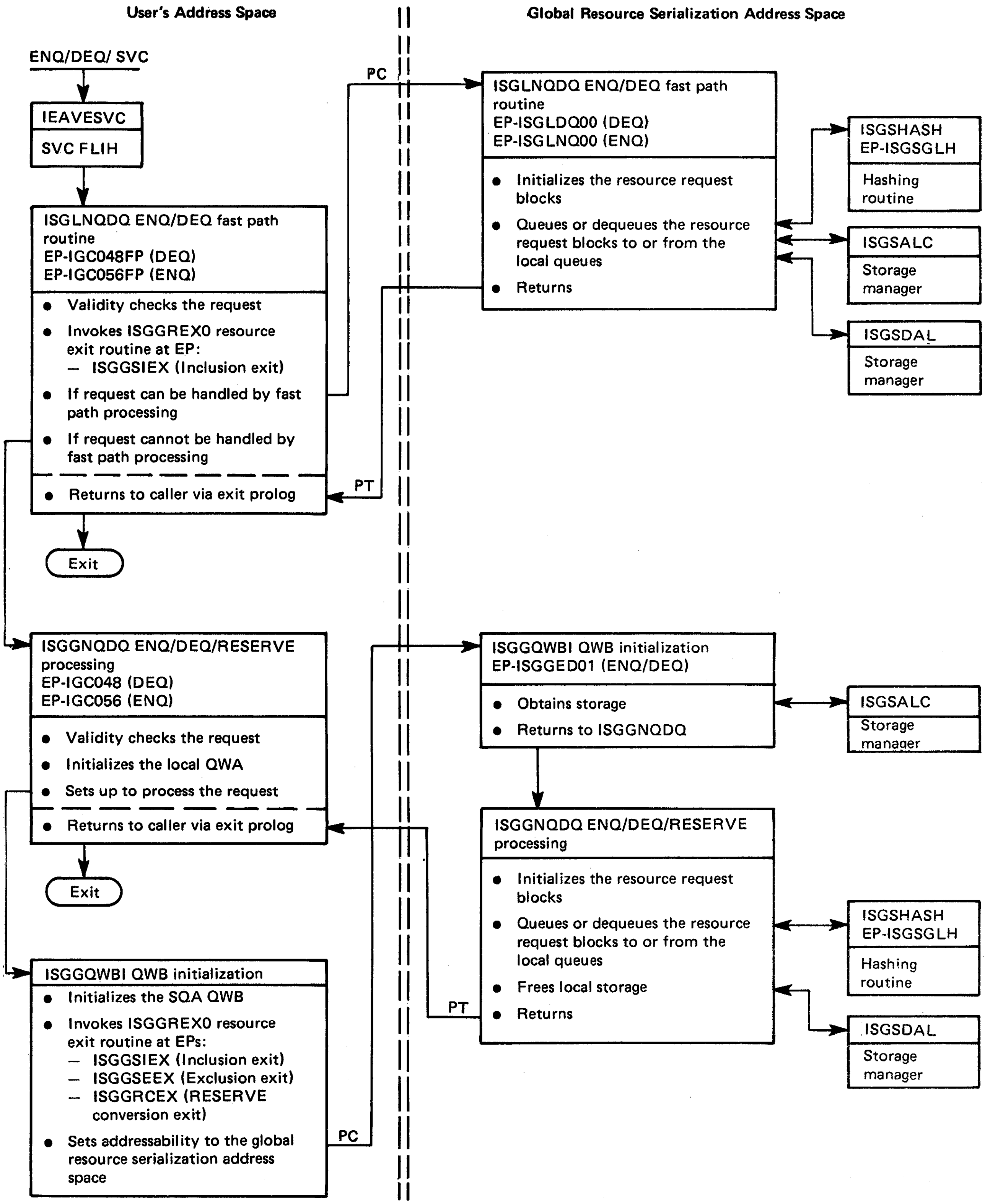

User's Address Space
Global Resource Serialization Address Space
ENQ/DEQ/ SVC
IEAVESVC
SVC FLIH
ISGLNQDQ ENQ/DEQ fast path routine
EP-IGC048FP (DEQ)
EP-IGC056FP (ENQ)
● Validity checks the request
● Invokes ISGGREX0 resource exit routine at EP:
– ISGGSIEX (Inclusion exit)
● If request can be handled by fast path processing
● If request cannot be handled by fast path processing
● Returns to caller via exit prolog
Exit
PC
PT
ISGLNQDQ ENQ/DEQ fast path routine
EP-ISGLDQ00 (DEQ)
EP-ISGLNQ00 (ENQ)
● Initializes the resource request blocks
● Queues or dequeues the resource request blocks to or from the local queues
● Returns
ISGSHASH
EP-ISGSGLH
Hashing routine
ISGSALC
Storage manager
ISGSDAL
Storage manager
ISGGNQDQ ENQ/DEQ/RESERVE processing
EP-IGC048 (DEQ)
EP-IGC056 (ENQ)
● Validity checks the request
● Initializes the local QWA
● Sets up to process the request
● Returns to caller via exit prolog
Exit
ISGGQWBI QWB initialization
● Initializes the SQA QWB
● Invokes ISGGREX0 resource exit routine at EPs:
– ISGGSIEX (Inclusion exit)
– ISGGSEEX (Exclusion exit)
– ISGGRCEX (RESERVE conversion exit)
● Sets addressability to the global resource serialization address space
PC
ISGGQWBI QWB initialization
EP-ISGGED01 (ENQ/DEQ)
● Obtains storage
● Returns to ISGGNQDQ
ISGSALC
Storage manager
ISGGNQDQ ENQ/DEQ/RESERVE processing
● Initializes the resource request blocks
● Queues or dequeues the resource request blocks to or from the local queues
● Frees local storage
● Returns
PT
ISGSHASH
EP-ISGSGLH
Hashing routine
ISGSDAL
Storage manager

Figure 5-88. Module Flow for ENQ/DEQ Mainline - Local Resource Request

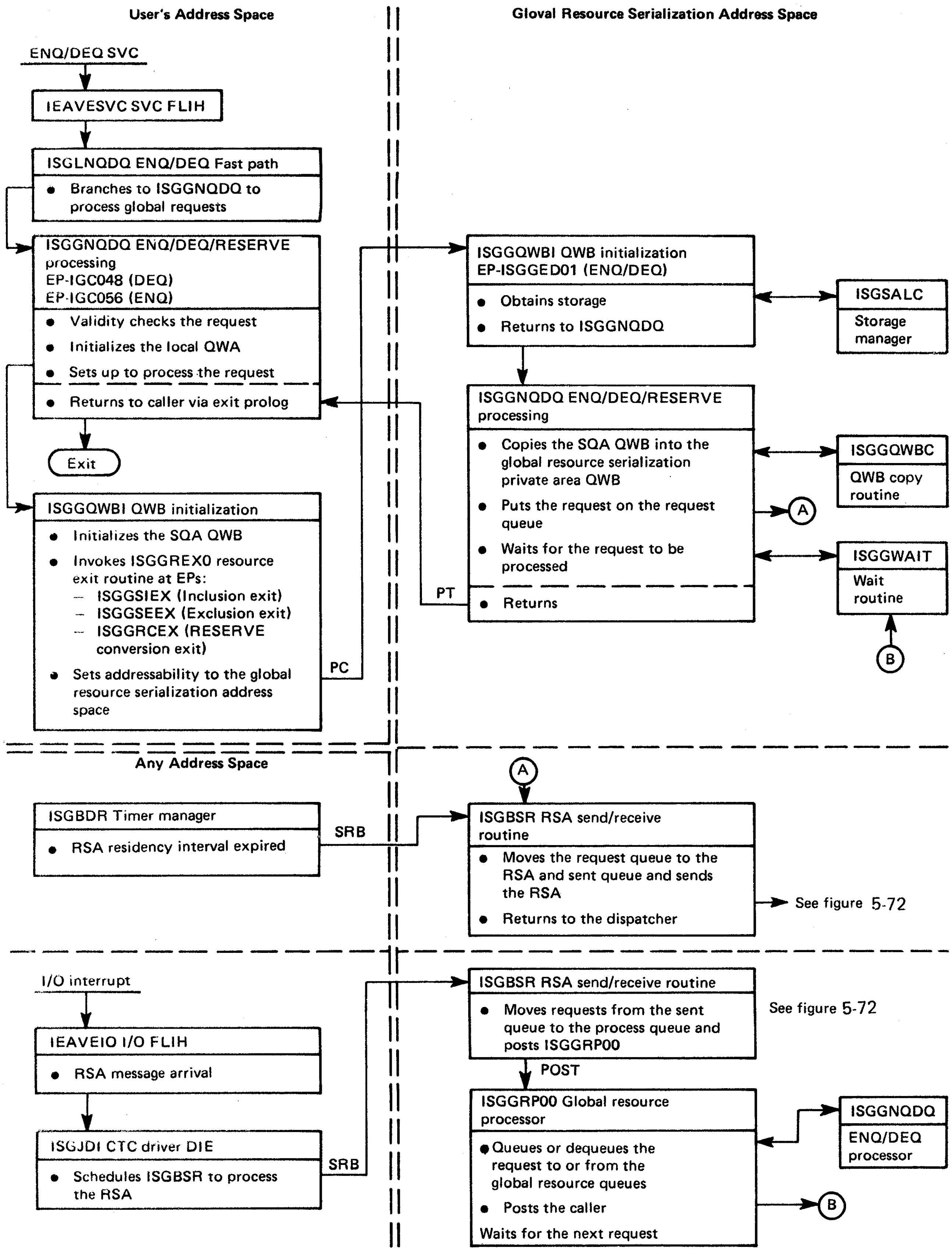

User's Address Space
Gloval Resource Serialization Address Space
ENQ/DEQ SVC
IEAVESVC SVC FLIH
ISGLNQDQ ENQ/DEQ Fast path
• Branches to ISGGNQDQ to process global requests
ISGGNQDQ ENQ/DEQ/RESERVE processing
EP-IGC048 (DEQ)
EP-IGC056 (ENQ)
• Validity checks the request
• Initializes the local QWA
• Sets up to process the request
• Returns to caller via exit prolog
Exit
ISGGQWBI QWB initialization
• Initializes the SQA QWB
• Invokes ISGGREX0 resource exit routine at EPs:
– ISGGSIEX (Inclusion exit)
– ISGGSEEX (Exclusion exit)
– ISGGRCEX (RESERVE conversion exit)
• Sets addressability to the global resource serialization address space
PC
ISGGQWBI QWB initialization
EP-ISGGED01 (ENQ/DEQ)
• Obtains storage
• Returns to ISGGNQDQ
ISGSALC
Storage manager
ISGGNQDQ ENQ/DEQ/RESERVE processing
• Copies the SQA QWB into the global resource serialization private area QWB
• Puts the request on the request queue
• Waits for the request to be processed
• Returns
PT
ISGGQWBC
QWB copy routine
A
ISGGWAIT
Wait routine
B
Any Address Space
ISGBDR Timer manager
• RSA residency interval expired
SRB
A
ISGBSR RSA send/receive routine
• Moves the request queue to the RSA and sent queue and sends the RSA
• Returns to the dispatcher
See figure 5-72
I/O interrupt
IEAVEIO I/O FLIH
• RSA message arrival
ISGJDI CTC driver DIE
• Schedules ISGBSR to process the RSA
SRB
ISGBSR RSA send/receive routine
• Moves requests from the sent queue to the process queue and posts ISGGRP00
See figure 5-72
POST
ISGGRP00 Global resource processor
• Queues or dequeues the request to or from the global resource queues
• Posts the caller
Waits for the next request
ISGGNQDQ
ENQ/DEQ processor
B

Figure 5-89. Module Flow for ENQ/DEQ Mainline - Global Resource Request

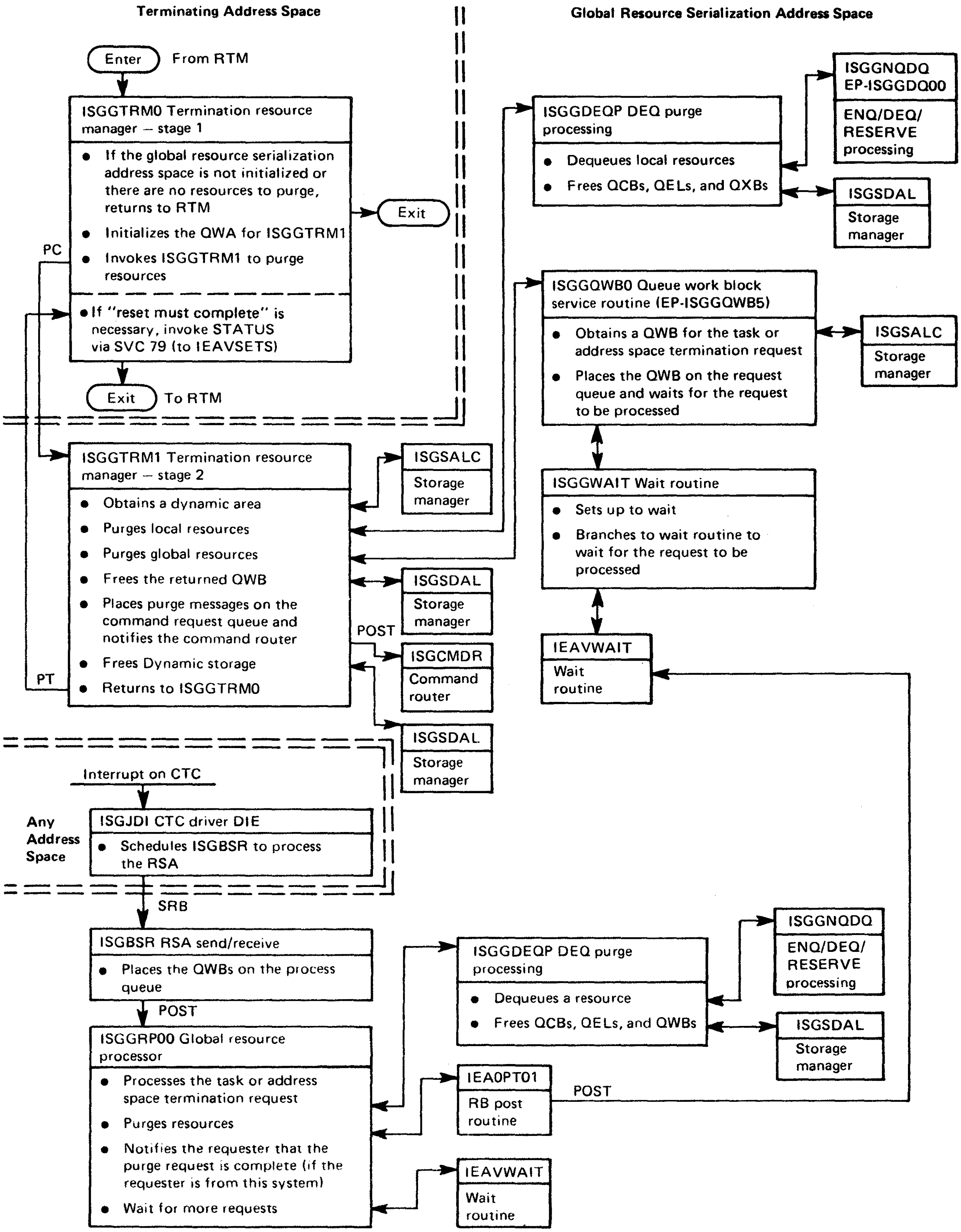

Terminating Address Space
Global Resource Serialization Address Space
Enter
From RTM
ISGGTRM0 Termination resource manager – stage 1
● If the global resource serialization address space is not initialized or there are no resources to purge, returns to RTM
● Initializes the QWA for ISGGTRM1
● Invokes ISGGTRM1 to purge resources
● If "reset must complete" is necessary, invoke STATUS via SVC 79 (to IEAVSETS)
Exit
Exit
To RTM
PC
PT
ISGGTRM1 Termination resource manager – stage 2
● Obtains a dynamic area
● Purges local resources
● Purges global resources
● Frees the returned QWB
● Places purge messages on the command request queue and notifies the command router
● Frees Dynamic storage
● Returns to ISGGTRM0
ISGSALC
Storage manager
ISGSDAL
Storage manager
POST
ISGCMDR
Command router
ISGSDAL
Storage manager
ISGGDEQP DEQ purge processing
● Dequeues local resources
● Frees QCBs, QELs, and QXBs
ISGGNQDQ EP-ISGGDQ00
ENQ/DEQ/ RESERVE processing
ISGSDAL
Storage manager
ISGGQWB0 Queue work block service routine (EP-ISGGQWB5)
● Obtains a QWB for the task or address space termination request
● Places the QWB on the request queue and waits for the request to be processed
ISGSALC
Storage manager
ISGGWAIT Wait routine
● Sets up to wait
● Branches to wait routine to wait for the request to be processed
IEAVWAIT
Wait routine
Interrupt on CTC
Any Address Space
ISGJDI CTC driver DIE
● Schedules ISGBSR to process the RSA
SRB
ISGBSR RSA send/receive
● Places the QWBs on the process queue
POST
ISGGRP00 Global resource processor
● Processes the task or address space termination request
● Purges resources
● Notifies the requester that the purge request is complete (if the requester is from this system)
● Wait for more requests
ISGGDEQP DEQ purge processing
● Dequeues a resource
● Frees QCBs, QELs, and QWBs
ISGGNQDQ
ENQ/DEQ/ RESERVE processing
ISGSDAL
Storage manager
IEAOPT01
RB post routine
POST
IEAVWAIT
Wait routine

Figure 5-90. Module Flow for the Termination Resource Manager

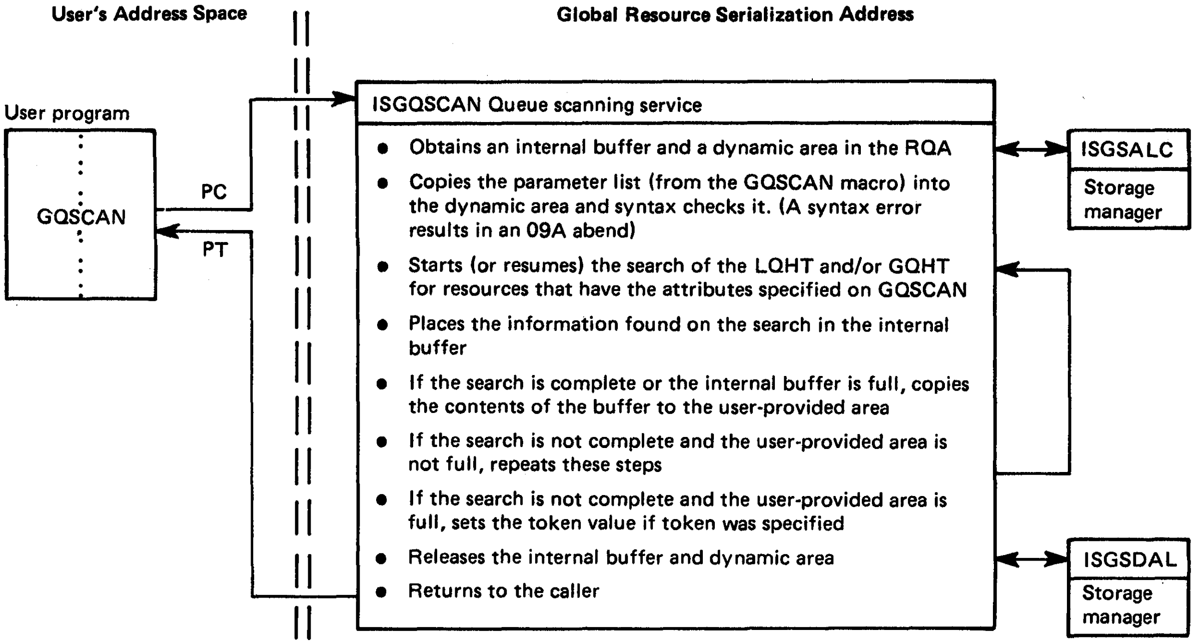

User's Address Space
Global Resource Serialization Address
User program
GQSCAN
PC
PT
ISGQSCAN Queue scanning service
Obtains an internal buffer and a dynamic area in the RQA
Copies the parameter list (from the GQSCAN macro) into the dynamic area and syntax checks it. (A syntax error results in an 09A abend)
Starts (or resumes) the search of the LQHT and/or GQHT for resources that have the attributes specified on GQSCAN
Places the information found on the search in the internal buffer
If the search is complete or the internal buffer is full, copies the contents of the buffer to the user-provided area
If the search is not complete and the user-provided area is not full, repeats these steps
If the search is not complete and the user-provided area is full, sets the token value if token was specified
Releases the internal buffer and dynamic area
Returns to the caller
ISGSALC
Storage manager
ISGSDAL
Storage manager

Figure 5-91. Module Flow for Queue Scanning Services

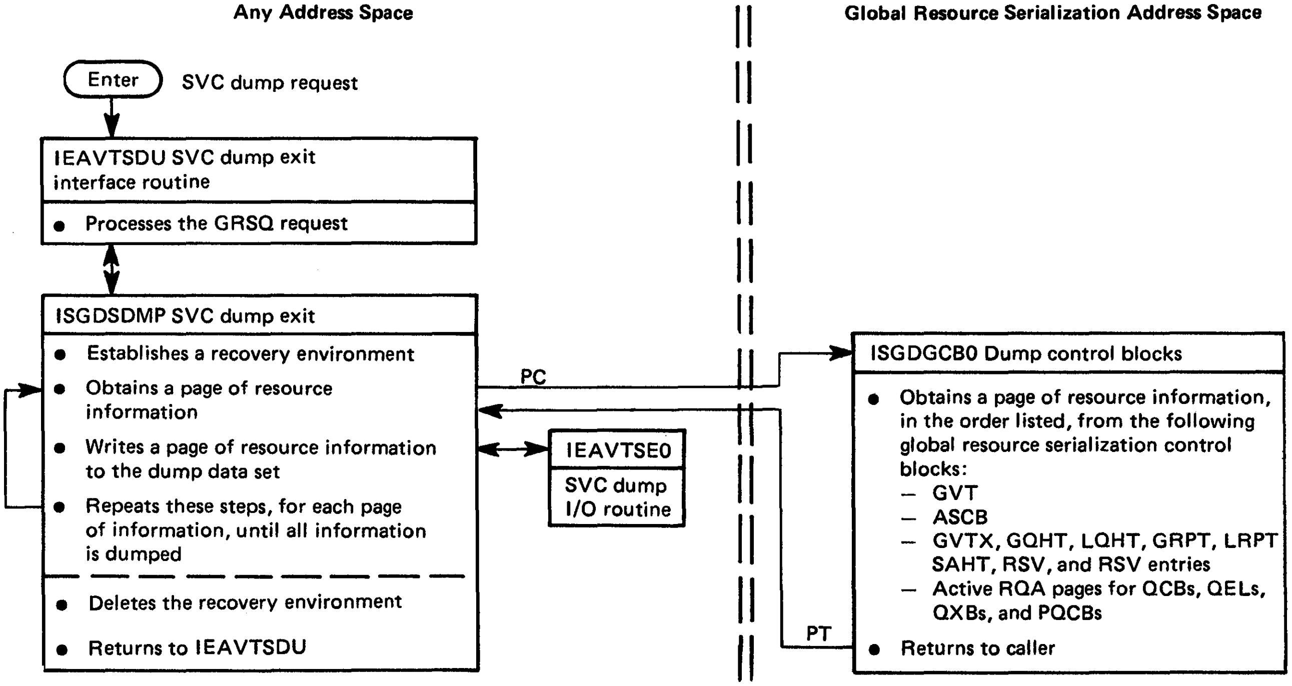

Any Address Space
Global Resource Serialization Address Space
Enter
SVC dump request
IEAVTSDU SVC dump exit interface routine
Processes the GRSQ request
ISGDSDMP SVC dump exit
Establishes a recovery environment
Obtains a page of resource information
Writes a page of resource information to the dump data set
Repeats these steps, for each page of information, until all information is dumped
Deletes the recovery environment
Returns to IEAVTSDU
PC
PT
IEAVTSEO
SVC dump I/O routine
ISGDGCB0 Dump control blocks
Obtains a page of resource information, in the order listed, from the following global resource serialization control blocks:
– GVT
– ASCB
– GVTX, GQHT, LQHT, GRPT, LRPT SAHT, RSV, and RSV entries
– Active RQA pages for QCBs, QELs, QXBs, and PQCBs
Returns to caller

Figure 5-92. Module Flow for Dump Support - SVC Dump

Diagnostic Aids

This topic contains diagnostic aids to help you solve problems with global resource serialization. It contains the following sections:

- Check on Enabled Wait During IPL
- System Indicators
- Probe Points
- CTC Processing Debugging Hints
- Ring Processing Debugging Hints
- ENQ/DEQ/RESERVE Processing Debugging Hints
- Storage Management Debugging Hints
- Serialization
- Recovery Routines
- SYS1.LOGREC Recording

Check on Enabled Wait During IPL

If an enabled wait occurs during IPL processing, you can make the following check to determine if the wait was due to missing entries in the SYSTEMS exclusion RNL.

- Check the request queue in the GVT (GVTREQQ) for QWBs.
- Compare the resource name identified in the PEL portion of the QWB to the entries in the SYSTEMS exclusion RNL and SYSTEM inclusion RNL.
- If the RNLs indicate that the resource name identifies a global resource, the requester of that resource must wait until master scheduler initialization completes before the requester is granted control of the resource.
- If the requester must complete processing prior to master scheduler initialization completing, the resource name must be added to the SYSTEMS exclusion RNL.

System Indicators

The following indicators, when set to one, have these meanings:

GVT indicators:

GVTGRSNA - Global resource serialization is not active. (Only local requests can be processed.)

GVTNCMDR - Global resource serialization commands cannot be processed.

GVTGQDMG - Global resource queues have been damaged. This system will reject VARY GRS,RESTART commands.

GVTNCOMM - CTC-driver and ring processing functions are not operative.

GVTNREQS - Requests cannot be put on the command request queue.

GCL indicators:

GCLINOP - CTC processing will not allow use of this CTC because a software error occurred and the control blocks of this CTC (GCL or RSL) might be damaged.

GCLIOERR - CTC processing will not allow use of this CTC because an I/O error occurred on this CTC.

GCLOFFLN - CTC processing will not allow use of this CTC because the CTC has been varied offline.

Probe Points

The following probe points are useful to help you debug global resource serialization problems or set SLIP traps.

1. Probe point for obtaining the RSA message that this system received:

   Module: ISGBSR
   Label: RECVPT1
   Data: - RSAPTR (register 6) points to the RSA.
   - Register 4 contains the length of the RSA.
   - Register 13 points to the RSV.
   - RSVIBFOR (RSV+X'8C') points to the received RSA.

2. Probe point for obtaining the RSA message that this system sent:

   Module: ISGBSR
   Label: SENDPT1
   Data: - Register 13 points to the RSV.
   - RSVOBFOR (RSV+X'90') points to the sent RSA.

3. Probe point for obtaining the QWB that is to processed (the first QWB on the process queue):

   Module: ISGGRP00
   Label: GRPNXTPQ
   Data: - Register 3 points to the GVT.
   - GVTPRCQF (GVT+X'40') points to the QWB to be processed.

CTC Processing Debugging Hints

The following debugging hints help you isolate problems in the CTC processing subcomponent.

1. Field GCLWGCQF of the GCL is the write queue of the corresponding GCL (representing a CTC) and points to a write GCQ when the write queue is not empty. GCLWGCQF is zero when the write queue is empty.

2. Field GCLCNTS is bumped by one before the STARTIO for a SENDBUF or SENDBUF-IMMEDIATE. Field GCLCNTC is bumped by one when the SENDBUF or SENDBUF-IMMEDIATE completes. Therefore, by comparing these two count fields you can determine if a write operation is in progress.

3. Field GCLRGCQF is the read queue of the corresponding GCL and points to a read GCQ when the read queue is not empty. GCLRGCQF points to a dummy GCQ (located in the GCV) when the read queue is empty.

4. The address in field GCLRGCQF is a word-multiple address when the GCL does not have a read channel program in progress. The address is bumped by one when a read channel program is started. Therefore, by checking the low order bit in GCLRGCQF you can determine if a read channel program is in progress.

5. Field GCLTRACE contains the last 15 CCW operation codes sensed from the corresponding CTC. In a dump, the acronym TRC1 appears a short distance before this field. The occurrence of an EE or ED operation code in this area

indicates that the system taking the dump sensed a broken channel program that was started by the system at the opposite end of the CTC.

Ring Processing Debugging Hints

The following debugging hints help you isolate problems in the ring processing subcomponent.

1. Field RSVIBFOR points to the RSA input buffer. Field RSVMRLRL contains the length of the last RSA received.

2. Field RSVOBFOR points to the RSA output buffer. Field GCBLNBUF of the RSA output GCB contains the length of the last RSA sent or the length of the RSA that soon will be sent. Field RSVGCBOP points to the RSA output GCB.

3. Field RSARCSEQ of the RSA is the RSA send count, which is a number that is bumped by one each time the RSA is sent. By comparing RSARCSEQ in the input buffer to RSARCSEQ in the output buffer, you can determine if the system that took the dump was holding the RSA at the time of the dump. Also, by comparing RSARCSEQ values in dumps taken by different systems, you can determine which system last received the RSA before a failure.

4. When a system is in the main ring, field RSVRSASC contains the RSA send count of the last RSA sent by this system (if the system is not holding the RSA) or the send count of the RSA that will soon be sent by this system (if the system is holding the RSA). RSVRSASC is set to zero when a system does main ring cleanup.

5. Subroutine CLNUFAIL (in module ISGBCI) does the main ring cleanup. When a system does main ring cleanup after a main ring disruption, CLNUFAIL copies field RSVRSASC to an entry in the RSVENTY table, and also marks entries in the RSVENTY table to show which systems were in the main ring at the time of the disruption and which RSA was last received before the disruption. Because main ring cleanup is serialized by the ISGBCI-ENQ-resource, cleanup might not occur immediately after the main ring disruption because another task might be holding the ISGBCI-ENQ-resource at the time of the disruption.

ENQ/DEQ/RESERVE Processing Debugging Hints

The following debugging hints help you isolate problems in the ENQ/DEQ/RESERVE processing subcomponent.

1. The queue work areas (QWAs) used by ENQ/DEQ mainline processing contain information that is useful in solving ENQ/DEQ/RESERVE problems. There are two QWAs: one for local resource processing (the local QWA pointed to by GVTLQWA), and the other for global resource processing (the global QWA pointed to by GVTGQWA).

   The QWA is divided into the following major areas:

   QWABASIC - This is the basic section of the QWA. It contains the information required by the mainline routine to process the resource request. For example, it indicates whether or not the request is authorized, whether global resources are part of the request, and whether the request is an ENQ or DEQ. This is also the only section of the QWA that can be mapped to the SVRB extended save area or the RMPL work area.

QWARSA - This is the first request save area section of the QWA. It contains the information required to process a global or local resource request. This section is moved to the QWBHRSA field and later restored to the QWARSA field by module ISGGRP00. It exists in the QWABASIC section of the QWA.

QWARSA2 - This is the second request save area section of the QWA. It contains the information needed to process a global or local resource request. This section contains the requester's job name, SYSID, ASID, and ASCB address. This data is moved to the QWBHRSA2 field and later restored to the QWARSA2 field by module ISGGRP00. It exists in the QWARDA section of the QWA.

QWARDA - This is the request data area section of the QWA. It contains the counts of the types of resources being processed, and the addresses of internal control blocks.

Work/Save areas - This series of general work/save areas follows the QWARDA area in the QWA and are used by the resource request processing routines. These areas are used to save register contents.

QWATRMRM - This work area section of the QWA is used by the termination resource manager. It contains information used by ISGGTRM0 and ISGGTRM1 to process a termination request.

When a local resource is being processed, the QWABASIC section of the QWA is moved to the SVRB extended save area when the requester of the resource must be suspended because the resource is not immediately available. QWABASIC information is then referenced in the SVRB extended save area following the notification that the resource is available.

When a global resource is being processed, the QWABASIC section of the QWA is always moved to the SVRB extended save area because the global resource requester is always suspended.

After the requester is notified (via cross memory post) that the requested resource is available, the data in the SVRB extended save area is copied back to the QWABASIC section of the QWA. This information in QWABASIC is then used to complete the processing of the request.

The main point to consider about the QWA is that whenever an ENQ/DEQ/RESERVE requester is suspended, the SVRB extended save area contains useful information that can be used in debugging. An important piece of information in the QWABASIC section of the SVRB extended save area is the QWB address used to define a global resource request. By locating this QWB (pointed to by QWAQWBA), you can find the data presented to ENQ/DEQ/RESERVE processing in the original request. If this field in the QWA is zero, then a local resource is being processed.

2. ENQ/DEQ/RESERVE processing uses two types of QWBs to process resource requests: the SQA QWB (pointed to by GVTSQWB), and the global resource serialization address space QWBs (pointed to by QXBQWB, GVTREQQ, and GVTPRCQF).

   When a local resource is being processed, the SQA QWB is used. When a global resource is being processed, the SQA QWB is used only until the global

resource serialization private area QWBs are constructed. The following shows
the process in which the resource data is passed between ISGGNQDQ and ISGGRP00.

- The requester's PEL is moved to the SQA QWB.

- The local QWA is initialized.

- Information in the QWA and SQA QWB is moved to the global resource serialization private area QWBs.

- The QWABASIC section of the local QWA is moved to the SVRB extended save area.

- The global resource serialization private area QWBs are placed on the request queue. (These QWBs are subsequently moved to the process queue by ring processing routines.)

- The ring processing function notifies ISGGRP00 that work (QWBs) is now available on the process queue.

- ISGGRP00 moves the QWBHRSA and QWBHRSA2 fields to the global QWARSA and QWARSA2 fields respectively.

- ISGGRP00 processes the requests and notifies the requester (ISGGNQDQ SVRB) when the resource request is satisfied.

- ISGGNQDQ restores the local QWA from the QWABASIC section of the SVRB extended save area. It then locates the global resource serialization private area QWBs defining this request from the restored QWABASIC section. This address is then used to restore the QWARSA from the QWBHRSA.

3. Prior to master scheduler initialization completing, any global resource requests placed on the request queue that are required for IPL processing will cause an enabled wait state. To prevent this from occurring, any global resource requests required during IPL processing before master scheduler initialization has completed should be placed in the SYSTEMS exclusion RNL.

ENQ/DEQ/RESERVE Termination Resource Manager Debugging Hints

The following debugging hints help you isolate problems in the ENQ/DEQ/RESERVE termination resource manager function:

1. For normal and abnormal task termination, ISGGTRM0 receives control from RTM in either the address space of the terminating task or the address space of the master scheduler. In either case, ISGGTRM0 issues a PC to ISGGTRM1 in the global resource serialization address space to process the request. The input resource manager parameter list (RMPL, which is pointed to by register 1 on entry) defines the type of termination request.

2. ISGGTRM0 uses the local QWA to store information related to its processing. QWABASIC is initialized with common resource processing information and QWATRMRM is initialized with information related to the task or address space being purged. For the format of this data, refer to the QWA in the *Debugging Handbook.*

3. If only local resources are being purged, the ENQ/DEQ cross memory services lock (CMSEQDQ) is held to provide serialization for the local QWA.

4. If global resources need to be purged, then the data stored in the QWA must be preserved during this process. ISGGTRM1 saves this data in the dynamic area before calling ISGGQWB5. Register 9 in ISGGTRM1 points to the dynamic area. The information in the dynamic area includes the QWARSA, QWAASCB, QWATRMRM, QWAJOBNM, GVTXLSMP, and RUB (register update block).

Storage Management Debugging Hints

The following debugging hints help you isolate problems in the storage management subcomponent.

1. Most global resource serialization control blocks reside in the global resource serialization address space. Pools of control blocks are maintained in resource pools as defined by two resource pool tables (RPTs), the local RPT and the global RPT. RPTs, in turn, address pool extension blocks (PEXBs) that define the control blocks (cells) for global resource serialization. (For an overview of these control blocks, see Figure 5-66.)

   Each PEXB is 4K bytes in length and contains multiple cells for control blocks of the same type and size. Listed below are the global resource serialization control blocks that are defined within a PEXB. (The RPT indexes are described in the following hint.)

Control Block	RPT Index	Name	Attributes
QCB	1	queue control block - size 1	local or global
QCB	2	queue control block - size 2	local or global
QCB	3	queue control block - size 3	local or global
QEL	4	queue element	local or global
QXB	5	queue extension block	local or global
QWB	6	queue work block	global only
HWKA	6	huge work area	local only
TWKA	7	tiny work area	local or global
PQCB	8	placeholder QCB	local or global
MRB	9	message request block	local or global
CRB	10	command request block	global only

   The RPT header contains either the acronym LRPT (local RPT) or GRPT (global RPT). Also, in the PEXB headers, the PEXBs addressed by each RPT contain the acronym PEXB as well as the acronym for one of the control blocks listed above. This information is useful when you are scanning the RQA in a dump listing to locate a particular control block, or when you find an address of an unknown control block. From the information in the PEXB, you can determine the type of control block (defined by the acronym) and whether or not the control block is in use by global resource serialization.

The control block is in use if it is not chained to the available cell chain in the PEXB header.

The available chain is double-headed (PEXFRST and PEXLAST) and single-threaded (PEXNCELL). Note that the first four bytes of each cell are used to chain available cells together.

2. A storage manager parameter list (SMPL) is the input to the storage manager allocation (ISGSALC) and deallocation (ISGSDAL) routines. The SMPL describes the number and type of control blocks requested. The type of control block is defined by an RPT index value in the SMPL. The RPT indexes (defined in the ISGRPT and ISGSMPL mapping macros) are used to index into the RPT to locate the RPT entry (RPTE) for the control block in question.

3. The QCB is defined in three sizes: size 1 for those with an RNAME of 24 bytes or less, size 2 for those with an RNAME of 44 bytes or less, and size 3 for those with an RNAME of 255 bytes or less. Each QCB has a unique index corresponding to the three sizes.

4. The sequence in which the storage manager allocates control blocks is:

   - When a request is received, ISGSALC attempts to satisfy the request from the queue of active PEXBs that are chained from RPTEFPXB and RPTELPXB. If, while scanning the active PEXB queue, ISGSALC finds a PEXB with no available cells, the PEXB is rechained to the end of the active PEXB queue.

   - If sufficient PEXBs are not available on the active queue, ISGSALC searches the inactive PEXB queue that is chained from RPTEIAPQ. If available, the inactive PEXB is moved to the front of the active PEXB queue and the required cells are obtained from this PEXB.

   - If the inactive PEXB chain is empty and the request is still not satisfied, an additional page is obtained from the RQA. A new PEXB is then constructed and chained to the front of the active queue.

   - If the RQA has been completely assigned, then the storage manager issues abend 09A with a reason code of 8104.

5. A bit map in the RQA defines each page of the RQA. When the storage manager attempts to allocate a control block and no active or inactive PEXB is found, the RQA bit map is searched for an available page. (The address of the bit map is in GVTXBTMP and the length of the bit map is in GVTXBTML.) The storage manager allocates control blocks from the high end of the RQA for global resources and the low end for local resources. Therefore, for global resources, the search proceeds from the high order bit in the bit map to the low order bit. For local resources, the search proceeds from the low order bit in the bit map to the high order bit. When a page is allocated in the RQA, the corresponding bit in the bit map is set to 1. When a page is deallocated from the RQA, such as a PEXB, the corresponding bit in the bit map is set to 0. By scanning the bits in the bit map, you can determine the number and locations of all allocated control blocks in the RQA. (The address of the RQA is in GVTXRQA.)

6. You can locate a PEXB header by zeroing the low order 12 bits of the cell (or control block) address. The PEXB header contains the addresses of the first and last available cells in this PEXB. The header also contains pointers to the previous and next PEXBs for this control block. By scanning the queue of available cells (pointed to by PEXFRST), you can determine if a particular control block is allocated to a function or has been released.

   When cells are returned to the storage manager, they are placed at the end of the available chain. When cells are assigned by the storage manager, they are assigned from the front of the queue. This ensures that a history of cell usage is maintained within the PEXBs because the oldest are used first.

   When all cells within a PEXB have been freed, the PEXB is moved to the front of the chain of available PEXBs (that is, the inactive PEXBs pointed to by RPTEIAPQ). Therefore, a history of PEXBs is not maintained. Whenever the count of inactive PEXBs (maintained in GVTXIACT) equals the count in RPTIACNT, all inactive PEXBs defined by this PRT are released. The storage manager deallocation routine (ISGSDAL) schedules ISGSPRLS to perform the page release function (via a branch entry to IEAVPSIB).

7. Control blocks in the RQA are not fixed. Instead, global resource serialization relies on the storage isolation function of SRM to ensure that the real frames associated with these virtual pages remain in storage until a critical storage shortage is encountered. (Refer to the *Initialization and Tuning Guide* for information about storage isolation.)

8. With the exception of the QWB, all global control blocks are serialized with the global resource serialization local lock. All local resources and the QWB are serialized with the ENQ/DEQ cross memory services lock (CMSEQDQ).

Serialization

When GRS=NONE is specified, all required global resource serialization resources are serialized with the CMSEQDQ lock.

When GRS=START or GRS=JOIN is specified, the following chart summarizes the serialization of the resources used by global resource serialization.

CMSEQDQ	Local	CS	Resource
X			Local hash table
	X		Global has table
	X		SYSID/ASID hash table
X			Local ASCB QEL queue
	X		Global ASCB QEL queue
X			Local storage management pools
	X		Global storage management pools
X			Storage management QWB pools
		X	Request queue
	X		Process queue
X			Local QWA
	X		Global QWA

Legend:

CMSEQDQ - ENQ/DEQ cross memory services lock

Local - Global resource serialization local lock

CS - Compare and Swap instruction

Recovery Routines

The recovery routines for the global resource serialization subfunctions are:

Recovery Routine	Subfunction
ISGBERCV - ESTAE ISGBFRCV - FRR	Ring processing
ISGCRCV - ESTAE ISGCRET0 - FRR ISGCRET1 - FRR	Command processing
ISGDSDMP (EP-ISGDSDRV) - ESTAE ISGDSNAP (EP-ISGDSNRV) - ESTAE	Dump support
ISGGEST0 - ESTAE ISGGFRR0 - FRR	Request (ENQ/DEQ/RESERVE) processing
ISGGQSRV (EP-ISGGRTRY) - FRR	Global queue services
ISGJRCV - FRR	CTC processing
ISGCRCV - ESTAE	WTO/WTOR message processing
ISGCRCV - ESTAE	Initialization
ISGQSCNR - FRR	Queue scanning services
ISGGFRR0 - FRR ISGSMI (EP-ISGSMIFR) - FRR	Storage management

SYS1.LOGREC Recording

All global resource serialization recovery routines (except ISGGEST0) record the following information in the SDWA:

SDWAMODN	- Load module name
SDWACSCT	- CSECT name - Date of compilation - Product/PTF number
SDWACID	- Component identifier (SCSDS) - Subcomponent identifier
SDWAREXN	- Recovery routine name

Additional information is recorded in the variable recording area (SDWAVRA) in the key-length-data format as described in the following topics.

Recorded by ISGBERCV

ISGBERCV records the following in the SDWAVRA:

- The REPL and its address. (The REPL contains execution footprints. Also, if the failing module was working with a particular RSL, the REPL contains the address of that RSL.)

- The RSC being processed at the time of failure and its address. (Recorded only if ISGBC1 was the failing module.)

- Six words copied from the UCB of the CTC that encountered the timeout condition. (Recorded only if ISGBCI is the failing module and the abend reason code is 620C.)

Recorded by ISGBFRCV

ISGBFRCV records the following in the SDWAVRA:

- The RVR and its address. (The RVR contains execution footprints. Also, if the failing module was working with a particular RSL, the RVR contains the address of that RSL.)

- The ISGBSR entry point that encountered the failure.

- The addresses of the RSLs used to receive and send the RSA.

- Field RSVCRSAT of the RSV, which indicates whether a ring processing function was being performed at the time of the failure. Also, field RSVCPHNO, which indicates the phase of the function being performed.

- The addresses of the RSA input buffer and output buffer, plus six words from the beginning of each buffer.

If the failure occurred for entry point ISGBSRRI, the following is also recorded:

- The address of the RSL.
- The device address of the CTC represented by that RSL.
- The RSL flags: RSLLKSF, RSLLKIF, and RSLBFCTC.

Recorded by ISGCRCV

ISGCRCV records the following in the SDWAVRA:

- The contents of the CRWALEIB field (LOGREC error information) when ISGCRCV begins recovery processing.

- The parameter list passed to ISGBCI if the error exit routine determined that the failure occurred during a call to ISGBCI. (ISGCRCV invokes exit routines in failing modules as a part of its recovery processing.)

- The contents of the CRWALEIB field when ISGCRCV completes processing.

For each CRWA on the chain, ISGCRCV repeats the recording noted above. Therefore, multiple CRWALEIB fields might be recorded.

Recorded by ISGCRET0

ISGCRET0 (at entry point ISGCR0RV) records the following in the SDWAVRA:

- The FRR parameter list. (Refer to the PARMAREA structure in module ISGCRET0.)

Recorded by ISGCRET1

ISGCRET1 (at entry point ISGCR1RV) records the following in the SDWAVRA:

- The FRR parameter list. (Refer to the PARMAREA structure in module ISGCRET1.)

Recorded by ISGDSDMP

ISGDSDMP (at entry point ISGDSDRV) records the following in the SDWAVRA:

- The contents of the DEPL (ESTAE parameter list for SDUMP).

Recorded by ISGDSNAP

ISGDSNAP (at entry point ISGDSNRV) records the following in the SDWAVRA:

- The ESTAE parameter list. (Refer to the PARMAREA structure in module ISGDSNAP.)

Recorded by ISGGEST0

ISGGEST0 does not request recording to SYS1.LOGREC. Nothing is copied into SDWAVRA.

Recorded by ISGGFRR0

ISGGFRR0 records the following in the SDWAVRA:

- The contents of the QFPL (ENQ/DEQ FRR parameter list).

- The contents of the output data area (ODA) if the queue verifier routine detects queue damage. (Refer to module IEAVEQV0 for the mapping of the ODA.)

- Internal processing flags. (Refer to the FLAGS structure in module ISGGFRR0.)

- Resource damage flags. (Refer to the DAMAGE structure in module ISGGFRR0.)

Recorded by ISGGQSRV

ISGGQSRV (at entry point ISGGRECV) records the following in the SDWAVRA:

- The error information block (EIB), which is local to ISGGQSRV.

Recorded by ISGJRCV

ISGJRCV records the following in the SDWAVRA:

- The CTC unit address.
- The address of the IOSB.
- The IOSB fields: IOSFLA, IOSFLB, IOSFLC, IOSCOD, IOSCSW, IOSSNS, and IOSUSE.
- The address of the GCQ.
- The first five words of the GCQ.
- The contents of GCL.

Recorded by ISGQSCNR

ISGQSCNR records the following in the SDWAVRA:

- The contents of QFPL1 (queue scanning services FRR parameter list).
- The input parameter list (built by the GQSCAN macro) to ISGQSCAN, if it is available.
- The original system completion code and reason code describing the error.
- The control block cell type and address, if the control block was found not valid.
- Internal recovery status flags. (Refer to the RCVYSTFG structure in module ISGQSCNR.)

Note: ISGQSCNR does not record the 09A abend code issued by ISGQSCAN.

Recorded by ISGSMI

ISGSMI (at entry point ISGSMIFR) records the following in the SDWAVRA:

- The FRR parameter list. (Refer to the PARMAREA structure in module ISGSMI.)
- The original system completion code and reason code (in SDWAGR15) describing the error.

Appendix A. Process Flows

This appendix describes the flow of various MVS processes. These processes are described in the following chapters:

- RSM Processing for Page Faults
- Swapping
- EXCP/IOS
- GETMAIN/FREEMAIN
- VTAM Process
- TSO

RSM Processing for Page Faults

This chapter describes the important aspects of the RSM component's page-fault processing. Figure A-1 outlines the major functions in this processing.

Note: When page fault assist (PFA) microcode is installed and active, initial page faults are usually resolved by PFA without presenting an interruption to the software. (Also note that PFA does not make entries in the trace table.)

During page fault processing, several important tests are made. The following describes what these tests are, where they are made, and what they mean during the course of the RSM page fault process.

IEAVPIX Tests

IEAVPIX performs the following:

- Checks the PGTE to ensure that PGTPVM is still on after the SALLOC lock has been obtained. This is done because in an MP environment the other processor might have validated the PGTE (turned PGTPVM off) between the time this processor page-faulted and the time the SALLOC lock was obtained.
- Checks the PGTE to ensure that PGTPAM is on. If it is not, this is a logical protection violation.
- For a first reference, a PFTE is allocated directly, if one is available on the available frame queue; the page is cleared to zeros and the PGTE is validated. A first reference is one where the PGTE contains X'0019' or X'0009' and there is no auxiliary storage assigned or there is no deferred PCB for the page.
- If reclaim from the available frame queue is possible, IEAVPIX allocates the PFTE and validates the PGTE.
- If IEAVPIX cannot allocate a PFTE, it initializes a PCB and calls IEAVGFA.

IEAVGFA Tests

IEAVGFA performs the following:

- Checks the XPTE to determine if any other requests for this page are outstanding. If XPTDEFER is on, other paging requests (PCBs) for this page have been deferred. (The PCB is on the GFA defer queue anchored in the PVT.) Normally, the fact that a paging request is currently outstanding is indicated by PFTPCBSI, but in the defer case there is no frame and therefore no PFTE is yet associated with the request. Collect any deferred requests for this page (using a match on VBNO and also on the ASID if the page is in the private area) and relate them to the current request.
- Analyzes the current request and any related requests for the requirements of the frame that is needed for the requests.

- Checks to see if the XRBN from the PGTE does not represent a first reference case. If it is not a first reference, it can be determined if page I/O is in progress for the page, of if the frame has been used for another purpose since last backing the VBN. If no I/O is in progress, the frame is acceptable to the current request and its related requests, and the frame still contains the requested VBN; then the frame is reclaimed and is available immediately.

- If PFTPCBSI is on, checks to see if a PCB can be found on an I/O queue. The VBN0 value is used to search for the correct PCB on the I/O queue to be searched. If no PCB is found, IEAVGFA issues a C0D abend to record the error. If a PCB is found, a PCB relate function is performed if the I/O frame is acceptable to the current request and its related requests.

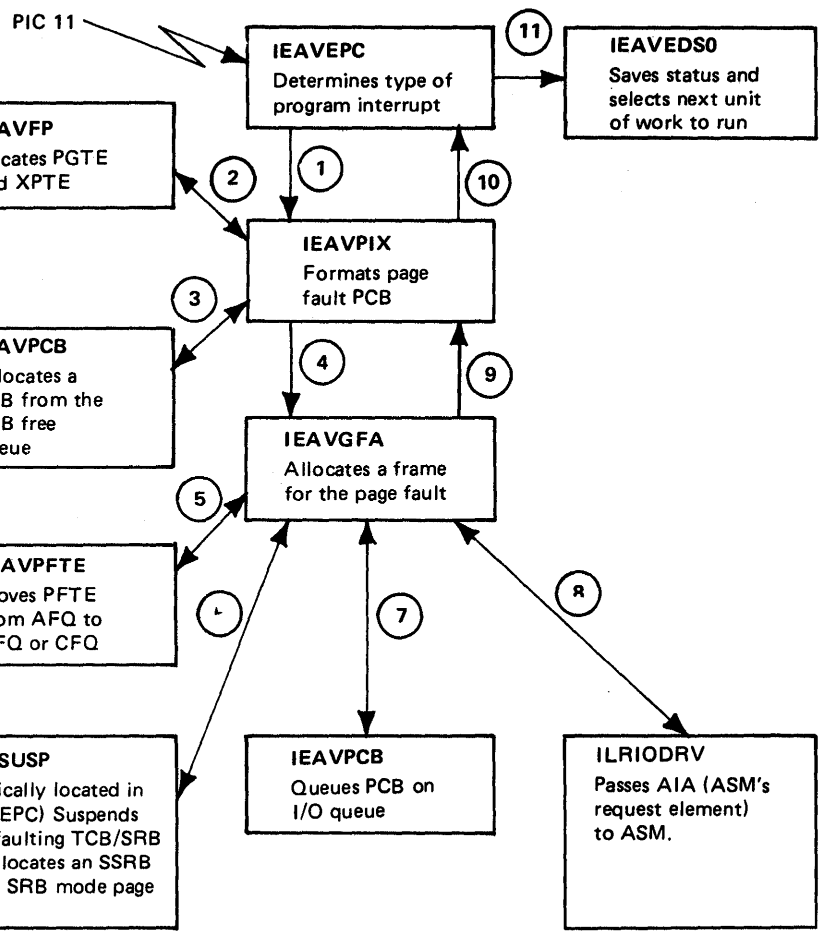
- If an old frame with or without I/O in progress cannot be found that is acceptable to the current request and its related requests, a frame is selected from the front of the AFQ. The PFTE is filled in and it is queued on either the common or the local frame queue. The XPTE (XPTXAV) is now checked to see if the paging data sets contain a copy of this virtual page. If XPTXAV is on, a page in operation is started; if it is off, the frame is cleared to zeroes.

- If the AFQ is empty, the request is deferred by placing the current PCB and any related PCBs on the PCB defer queue (PVTGFADF) of the PVT. The XPTDEFER flag is set in the XPTE.

- If a page-in is needed, the XRBN of the allocated frame is placed in the PCB in the AIA (which is always physically adjacent to the PCB) and the AIA is passed to ASM. Processing then exits as shown by steps 9, 10 and 11 in Figure A-1.

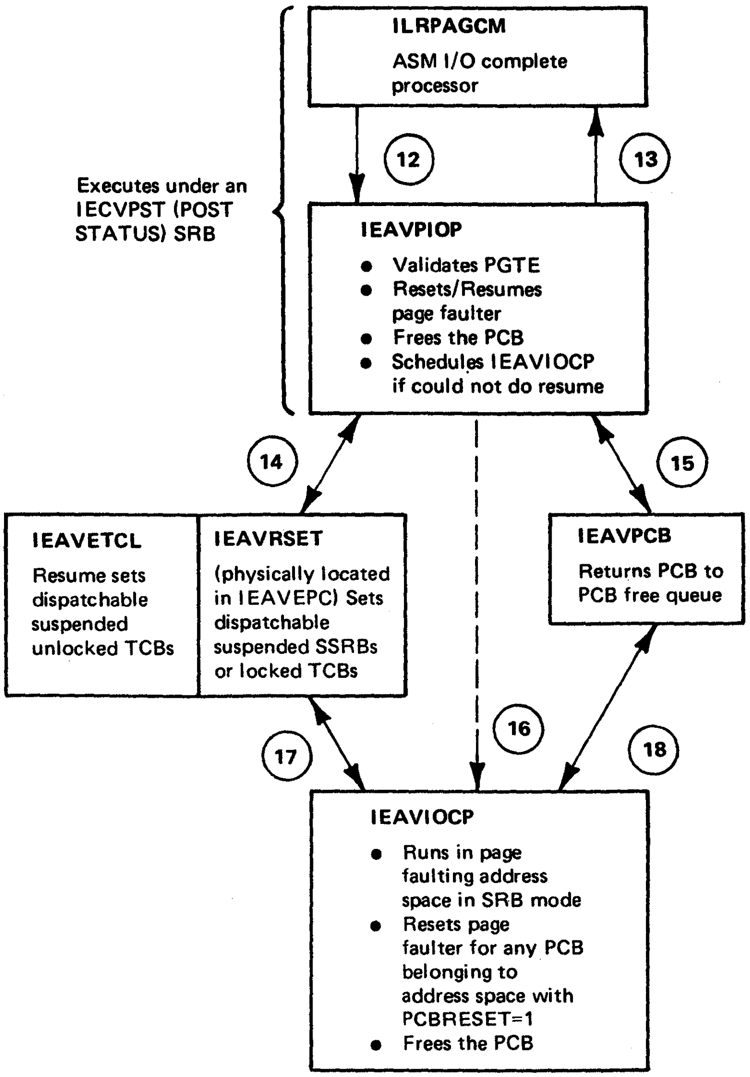
IEAVPIOP Tests

IEAVPIOP receives control from ASM and IEAVPIOP is passed the AIA when I/O has completed. IEAVPIOP checks for an I/O error and marks the PCB I/O complete. If necessary, IEAVPIOP indicates an I/O error in the PCB. IEAVPIOP checks PCBFREAL to determine if the reason for the page-in still exists. If PCBFREAL = 1, the page-in has been NOPed for some reason (such as FREEMAIN) and the frame is sent to the AFQ. If PCBFREAL = 0, the PGTE is validated. IEAVPIOP will then RESET/RESUME the suspended page faulter and the PCB is returned to the free queue. If RESUME is unsuccessful, then IEAVPIOP will leave the PCB on the I/O queue and SCHEDULE IEAVIOCP.


PIC 11
IEAVEPC
Determines type of program interrupt
11
IEAVEDSO
Saves status and selects next unit of work to run
IEAVFP
Locates PGTE and XPTE
2
1
10
IEAVPIX
Formats page fault PCB
3
IEAVPCB
Allocates a PCB from the PCB free queue
4
9
IEAVGFA
Allocates a frame for the page fault
5
IEAVPFTE
Moves PFTE from AFQ to LFQ or CFQ
6
7
8
IEAVSUSP
(physically located in IEAVEPC) Suspends page faulting TCB/SRB and allocates an SSRB for an SRB mode page fault
IEAVPCB
Queues PCB on I/O queue
ILRIODRV
Passes AIA (ASM's request element) to ASM.

Note: Circled numbers indicate the sequence of processing.

Figure A-1 (Part 1 of 2). Page Fault Process Flow


ILRPAGCM
ASM I/O complete processor
12
13
Executes under an IECVPST (POST STATUS) SRB
IEAVPIOP
● Validates PGTE
● Resets/Resumes page faulter
● Frees the PCB
● Schedules IEAVIOCP if could not do resume
14
15
IEAVETCL
Resume sets dispatchable suspended unlocked TCBs
IEAVRSET
(physically located in IEAVEPC) Sets dispatchable suspended SSRBs or locked TCBs
IEAVPCB
Returns PCB to PCB free queue
16
17
18
IEAVIOCP
● Runs in page faulting address space in SRB mode
● Resets page faulter for any PCB belonging to address space with PCBRESET=1
● Frees the PCB

Note: Circled numbers indicate the sequence of processing.

Figure A-1 (Part 2 of 2). Page Fault Process Flow

IEAVIOCP Tests

IEAVIOCP runs in SRB mode and searches the local and common PCB queues looking for I/O complete PCBs. Once found, IEAVIOCP calls IEAVRSET for any I/O complete PCBs with PCBRESET = 1. The reset function (IEAVRSET in IEAVEPC) is responsible for making the suspended work (TCB/SSRB) redispatchable.

IEAVIOCP searches the local and common PCB queues looking for I/O complete PCBs. Once found, IEAVIOCP calls IEAVRSET for any I/O complete PCBs with PCBRESET = 1. The reset function (IEAVRSET in IEAVEPC) is responsible for making the suspended work (TCB/SSRB) redispatchable. IEAVIOCP validates the PGTE for any I/O-complete PCB with a non-zero PCBVBN, with PCBFREAL = 0, and without an I/O error (PCBIOERR = 0). When this is done, IEAVIOCP returns the PCB to the free queue.

Because IEAVIOCP is queue-driven, it might not be able to get the local lock when it requests it. In such a case, it can be held in suspension by a page faulter whose PCB is on the queue IEAVIOCP is working on. Therefore, up to two SRBs can be scheduled for IEAVIOCP at one time. If IEAVIOCP does not hold the local lock and discovers an I/O-complete PCB that needs to be reset and for which reset requires the local lock (PCBLLHLD = 0, PCBSRBMD = 0, PCBPEX = 1, an unlocked TCB page fault), it can call IEAVOPBR to reschedule itself (exit to dispatcher). IEAVIOCP continues its scan of the PCB queues, doing any work possible before it exits to the dispatcher.

Swapping

This chapter describes the major considerations and decisions of the swapping processes (swap-in and swap-out).

Swap-In Process

The numbers in the following descriptions correlate to the circled numbers in Figure A-2.

1 - 2 SRM schedules IEAVSWIN and passes it the address of the ASCB in SRBPARM. IEAVSWIN obtains working-set size (SPCTWSSZ) + 1 PCBs. It then scans the SPCT LSQA entries and fills in a PCB for each entry. Next IEAVSWIN scans the stage one pageable page entries. Finally, IEAVSWIN scans the fix entries. For private area fix entries, it builds a stage one PCB. For common area fixes, it adds the SPCT fix count to the PFTE fix count. For common area fixes not in storage, it builds a PCB. Next, IEAVSWIN scans the SPCT segment entries and builds a PCB for each bit map entry. It then returns unused PCBs to the PCB free queue and calls IEAVGFA. If enough frames are not available for the stage one pages, IEAVGFA returns a code of eight to IEAVSWIN and sets PCBRETRY. IEAVSWIN notifies SRM via a SYSEVENT SWINFL to try the swap-in later.

3 - 4 IEAVGFA allocates frames for both stage one and stage two PCBs and then calls ASM to start swap-in I/O.

5 After swap-in I/O completes, the IEAVSWIN root exit IEAVSIRT is called by IEAVPIOP with stage one PCBs chained from the root PCB. IEAVSWIN does the following:

- Updates PFTFXCT if any fix counts are greater than 255
- Sets ASTESTD
- Fills in SGTEs in non-translate mode
- Fills in PGTEs in non-translate mode

6 IEAVSIRT calls IEAVPCB to free the root and all stage one PCBs.

7 IEAVSIRT calls ASCBCHAP to put the ASCB back on the ASCB queue.

8 IEAVSIRT calls status to start both quiesceable and non-quiesceable SRBs.

9 IEAVSIRT obtains an SRB from the RSM cell pool and schedules IEAVSWPP into the swapped-in address space so it can post the region control task. SWINPOST posts RCT's ASCBQECB to restore the address space and to start the I/O for stage two pages. Note that stage two frames are allocated at the same time as stage one frames.

10 IEAVGFA allocates frames for stage two PCBs and then calls ASM to start paging I/O.

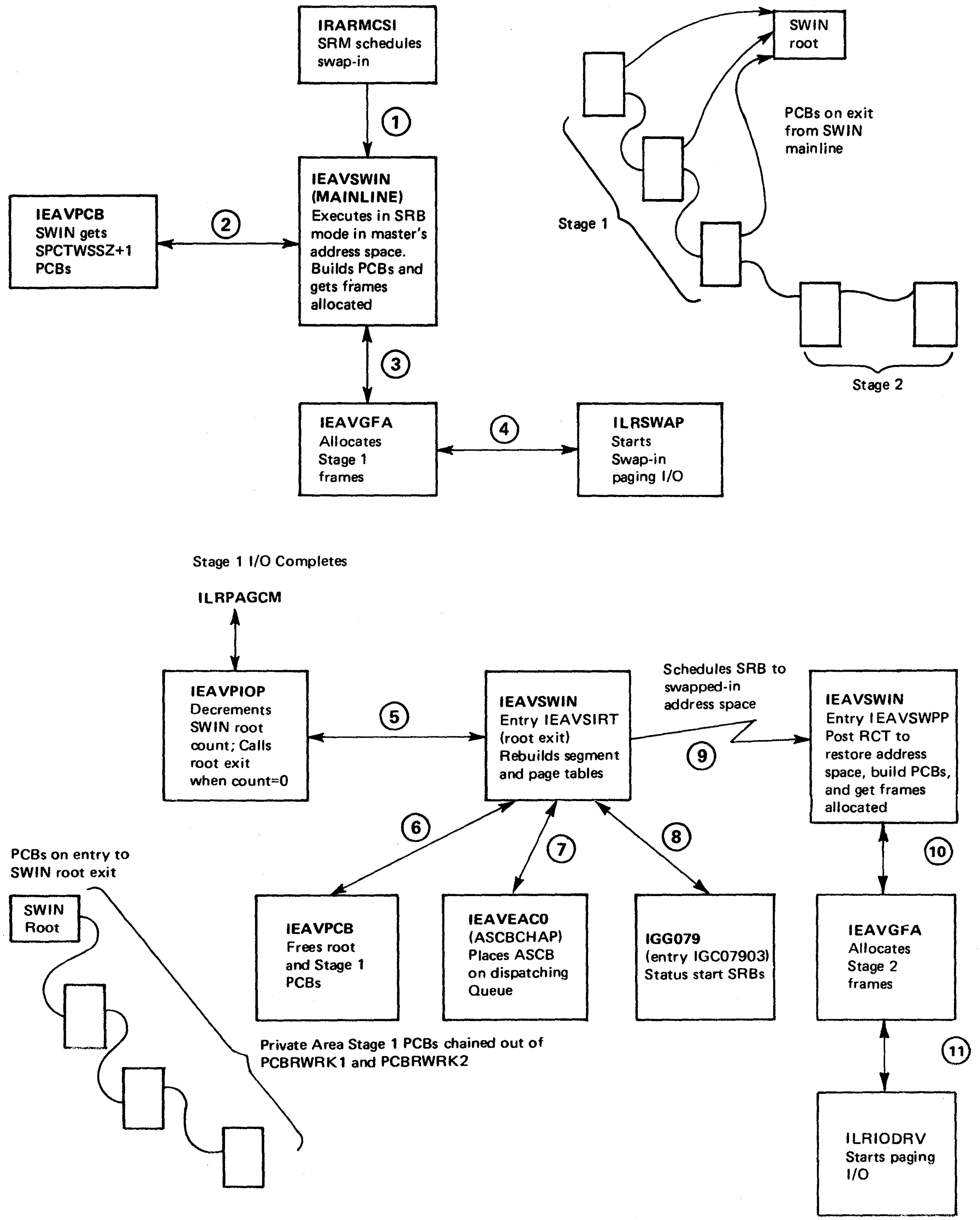


Figure A-2. Swap-In Process Flow

Swap-Out Process

IEAVAR02

SRM (IRARMCSO) posts the region control task (RCT) to swap out the address space. RCT is responsible for:

- IOS purge processing: I/O requests that have been requested or started are purged or quiesced, respectively.
- Halting all tasks in the address space with the exception of its own task.
- Preventing system SRBs from executing.

IEAVSOUT

The numbers in the following descriptions correlate to the circled numbers in Figure A-3.

1 IEAVSOUT receives control from RCT and calls STATUS (IEAVSSNQ) to stop non-quiesceable SRBs.

2 IEAVSOUT gets enough PCBs to page out every private area page in the address space plus one to be used as a swap out root.

3 IEAVSOUT clears the swap control table (SPCT) LSQA stage one pageable page, and fix entries (SPCTSWPE), and all bits in the bit maps in the segment entries (SPCTSEGE). Prior to this, the SPCT reflects the status of the address space at the last swap-out. SPCTSEGEs provide a mechanism to check how many and which segments are obtained via GETMAIN in an address space because there is a SPCTSEGE for each private area segment that is obtained by GETMAIN.

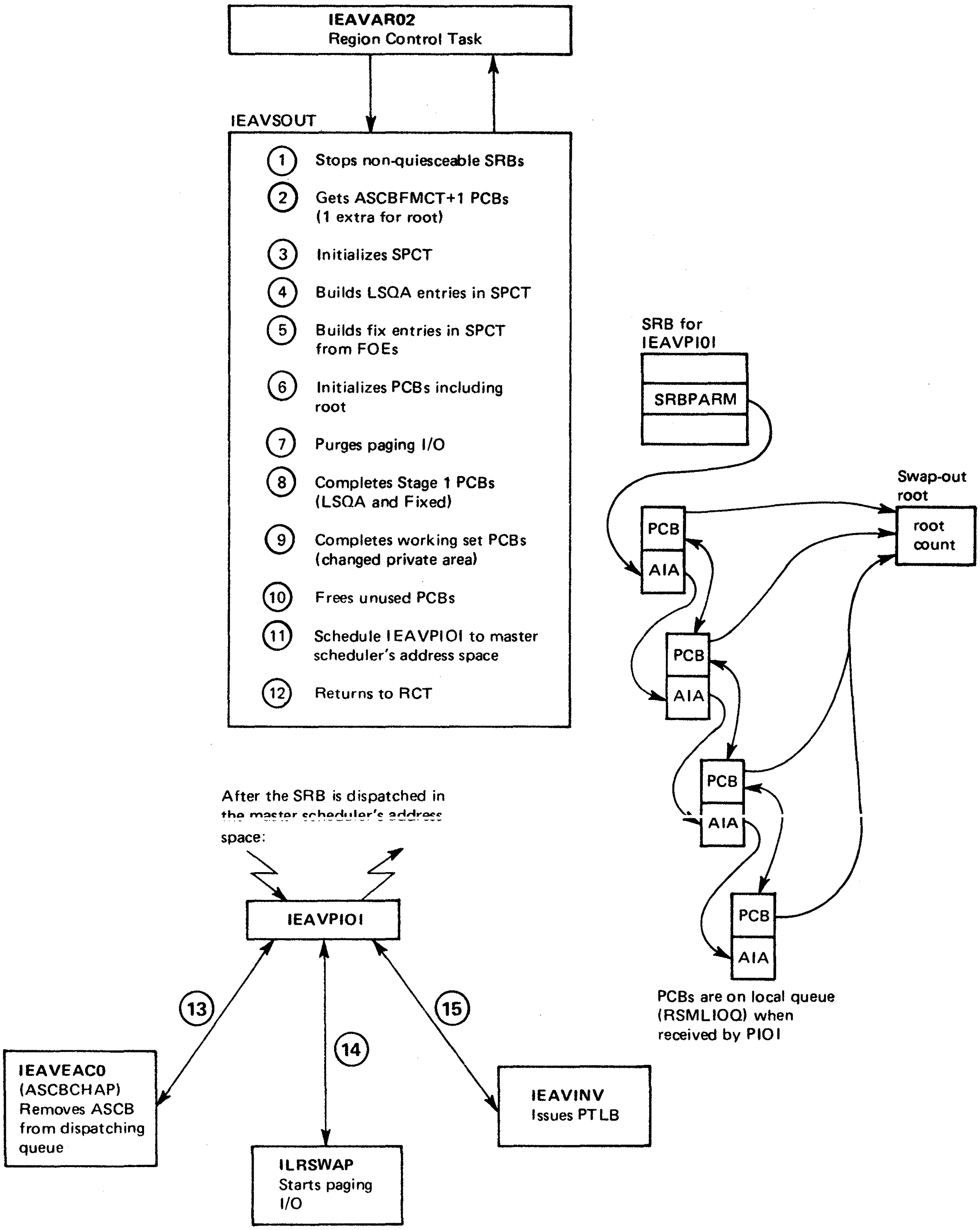
4 IEAVSOUT builds a six-byte LSQA entry for each frame on the LSQA frame queue.

5 IEAVSOUT builds an eight-byte fix entry for each page (private or common) that has an FOE on any TCB in this address space. The fix count is added into the fix entry SPCTSWPE. Note that fixes done without a TCB address supplied do not have FOEs.

6 IEAVSOUT initializes a root PCB to zero. It initializes the remaining PCBs, which might be used to swap-out a page as follows:

Partially initialized Swap-Out PCB

	PCB	
X‘00’	FF000000	- Not currently queued flags
	00000000	
X‘08’	06aaaaaa	- Root and output flags, and
	00000000	address of root PCB
X‘10’	80000000	- Free real frame flag
X‘14’	80000000	- Swap-out flag
	00000000	
	00000000	
X‘20’	aaaaaaaa	- Address of ASCB
X‘24’	00000000	- Start of AIA
	00000000	
X‘2C’	18C00000	- Swap-out and write flags
	00000000	
	00000000	
	00000000	
	00000000	



1. Stops non-quiesceable SRBs
2. Gets ASCBFMCT+1 PCBs (1 extra for root)
3. Initializes SPCT
4. Builds LSQA entries in SPCT
5. Builds fix entries in SPCT from FOEs
6. Initializes PCBs including root
7. Purges paging I/O
8. Completes Stage 1 PCBs (LSQA and Fixed)
9. Completes working set PCBs (changed private area)
10. Frees unused PCBs
11. Schedule IEAVPIOI to master scheduler's address space
12. Returns to RCT

SRB for
IEAVPIOI
SRBPARM
Swap-out
root
root
count
PCB
AIA
PCB
AIA
PCB
AIA
PCB
AIA
PCBs are on local queue (RSMLIOQ) when received by PIOI
After the SRB is dispatched in the master scheduler's address space:
IEAVPIOI
13
14
15
IEAVEACO
(ASCBCHAP)
Removes ASCB from dispatching queue
ILRSWAP
Starts paging I/O
IEAVINV
Issues PTLB

Figure A-3. Swap-out Process Flow

7 IEAVSOUT purges paging for this address space on the common PCB I/O queue, local PCB I/O queue, and the GFA deferred queue. The processing is to post users waiting on fixes, reset page faulters, and to NO-OP the PCB. (Fix entries are made for PCBs found for private area zero TCB fixes.) The NO-OP process makes the PCB look like a cancelled page load PCB; that is, no notification (RESET/POSTING) is to be done and the frame is to be freed. PCBs on the GFA defer queue are removed. The only exceptions here are for zero TCB fixes for which no entries could be made in the SPCT (GETMAIN for SQA failed). These PCBs remain unchanged and the fixed frame remains fixed throughout the swap.

8 IEAVSOUT fills XRBNs and VBNs into PCBs for each LSQA stage one pageable page or fix entry now in the SPCT. Even unchanged fixed or LSQA pages are paged out. If a one-stage swap is to be done, unchanged recently referenced pageable pages are also paged out.

9 IEAVSOUT next initializes a PCB for each changed page on the local frame queue for which a stage one pageable page PCB was not built. It then sets a bit in the bit map (SPCTBITM) for all pages that are to be paged in as part of stage two swap-in. The steal is based on a comparison of a criterion number passed by SRM in OUXBSTC to PFTUIC.

10 IEAVSOUT returns any unused PCBs to the free queue. This marks where on the free queue the swap-out began.

11 IEAVSOUT schedules an SRB for IEAVPIOI, releases the SALLOC lock, and returns to RCT (IEAVAR02), which waits for ASCBQECB to be posted by swap-in (IEAVSWIN). Because release of the SALLOC lock enables the processor, an address space is often swapped-out before RCT has gotten a chance to wait. When analyzing a stand-alone dump, you will see the following if the above case is true:

- The RCT is dispatcl.able.
- There is no wait count in RBWCF.
- There are no frames allocated to storage (ASCBFMCT=0).
- The address space is not on the ASCB dispatchability queue.

Do not consider this situation a problem.

IEAVPIOI

IEAVPIOI receives control in the master scheduler's address space. It calls ASCBCHAP to remove the ASCB from the dispatching queue, calls ASM with the string of AIAs passed to it from IEAVSOUT via the SRBPARM field, and calls IEAVINV to PTLB and exits.

EXCP/IOS

Figure A-4 is an overview of the I/O process through MVS using EXCP as the driver. The following outline correlates to this process.

1. Problem program issues GET/PUT (implied wait).

2. Problem program branches to access method.

3. Access method issues SVC 0 (EXCP) to EXCP front end.
   or
   Access method issues SVC 114 (EXCPVR) to EXCP front end.

4. EXCP front end:

   a. Validates request.
   b. Builds RQE.
   c. Queues related requests.
   d. If a VIO data set, goes to window intercept processor.
   e. Builds SRB/IOSB.
   f. If a virtual user, gets TCCW and BEB.
   g. Branches to PAGE FIX appendage (if specified and not a V = R region).
   h. Branch returns.
   i. If EXCPVR request, fixes pages from PAGE FIX appendage.
   j. Fixes DEB for V = R user if not already fixed.
   k. If a DASD device, branches to END of EXTENT appendage (if seek address is out of specified extent).
   l. Branch returns.
   m. Branches to START I/O appendage if specified.
   n. Branch returns.
   o. If virtual user: translates CCWs, fixes pages for buffers, and builds IDAL.
   p Issues START I/O macro (branch entry to IOS front end).

5. IOS front end.

   a. Builds IOQ.
   b. Selects physical path (channel scheduling).
   c. If path available, adds prefix CCWs and issues SIO; otherwise, queues IOQ on LCH.
   d. Restarts all queued I/Os to available channels.
   e. Branch returns to EXCP front end and branch returns from EXCP front end to problem program WAIT.

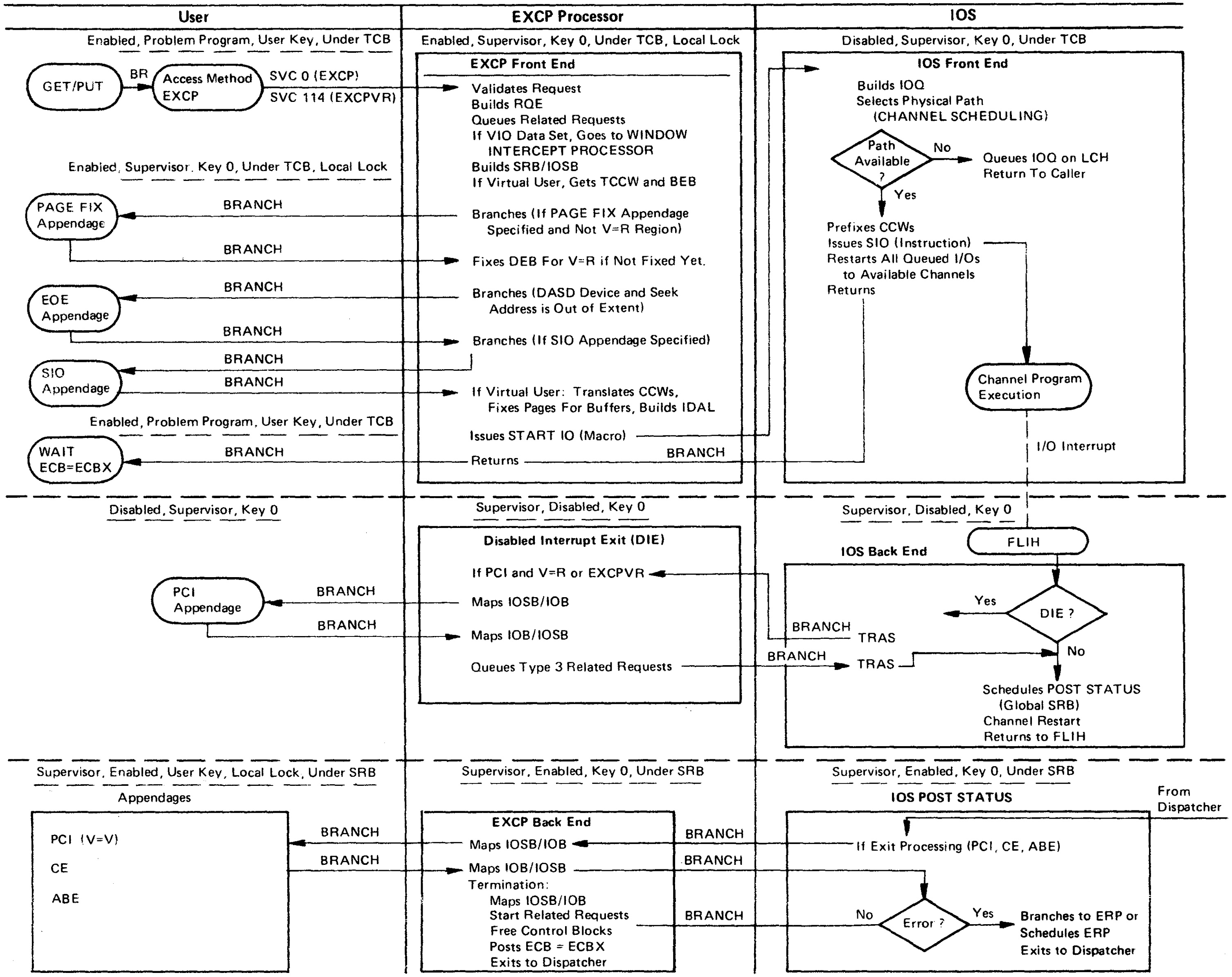

User
EXCP Processor
IOS
Enabled, Problem Program, User Key, Under TCB
Enabled, Supervisor, Key 0, Under TCB, Local Lock
Disabled, Supervisor, Key 0, Under TCB
GET/PUT
BR
Access Method EXCP
SVC 0 (EXCP)
SVC 114 (EXCPVR)
EXCP Front End
Validates Request
Builds RQE
Queues Related Requests
If VIO Data Set, Goes to WINDOW INTERCEPT PROCESSOR
Builds SRB/IOSB
If Virtual User, Gets TCCW and BEB
Branches (If PAGE FIX Appendage Specified and Not V=R Region)
Fixes DEB For V=R if Not Fixed Yet.
Branches (DASD Device and Seek Address is Out of Extent)
Branches (If SIO Appendage Specified)
If Virtual User: Translates CCWs, Fixes Pages For Buffers, Builds IDAL
Issues START IO (Macro)
Returns
BRANCH
IOS Front End
Builds IOQ
Selects Physical Path (CHANNEL SCHEDULING)
Path Available ?
No
Queues IOQ on LCH
Return To Caller
Yes
Prefixes CCWs
Issues SIO (Instruction)
Restarts All Queued I/Os to Available Channels
Returns
Channel Program Execution
I/O Interrupt
Enabled, Supervisor, Key 0, Under TCB, Local Lock
PAGE FIX Appendage
BRANCH
BRANCH
EOE Appendage
BRANCH
BRANCH
SIO Appendage
BRANCH
BRANCH
Enabled, Problem Program, User Key, Under TCB
WAIT ECB=ECBX
BRANCH
Disabled, Supervisor, Key 0
Supervisor, Disabled, Key 0
Supervisor, Disabled, Key 0
Disabled Interrupt Exit (DIE)
If PCI and V=R or EXCPVR
Maps IOSB/IOB
Maps IOB/IOSB
Queues Type 3 Related Requests
PCI Appendage
BRANCH
BRANCH
FLIH
IOS Back End
DIE ?
Yes
No
BRANCH
TRAS
BRANCH
TRAS
Schedules POST STATUS (Global SRB)
Channel Restart
Returns to FLIH
Supervisor, Enabled, User Key, Local Lock, Under SRB
Supervisor, Enabled, Key 0, Under SRB
Supervisor, Enabled, Key 0, Under SRB
Appendages
PCI (V=V)
CE
ABE
BRANCH
BRANCH
EXCP Back End
Maps IOSB/IOB
Maps IOB/IOSB
Termination:
Maps IOSB/IOB
Start Related Requests
Free Control Blocks
Posts ECB = ECBX
Exits to Dispatcher
BRANCH
BRANCH
BRANCH
IOS POST STATUS
From Dispatcher
If Exit Processing (PCI, CE, ABE)
Error ?
No
Yes
Branches to ERP or Schedules ERP
Exits to Dispatcher

Figure A-4. IOS/EXCP Process Flow

6. IOS back end (entry from I/O FLIH) entered as a result of I/O interrupt.

   a. If DIE is specified:

      1) TRAS (translates address space - to get addressability to control blocks in originating address space).
      2) Branch enters DIE.
      3) If PCI and V=R or EXCPVR, maps IOSB to IOB and branch enters PCI appendage.
      4) PCI processing.
      5) Branch returns ot DIE.
      6) Maps IOB to IOSB.
      7) Queues type-3-related requests.
      8) Branch return to IOS front end.
      9) TRAS (returns to addressability at time of interrupt).

   b. Schedules POST STATUS ﬄglobal“ (means POST STATUS will be entered via dispatcher).

   c. Branches to channel restart to start queued IOQEs on LCHs.

   d. Returns to FLIH.

   e. If system was in SRB mode, loads PSW for SRB or returns to dispatcher.

7. IOS POST STATUS (scheduled from IOS back end).

   a. If PCI, CE or ABE appendages specified:

      1) Branch enters EXCP back end.
      2) Maps IOSB to IOB.
      3) Branch enters appropriate appendage.
      4) Appendage processing.
      5) Branch returns to EXCP back end.
      6) Maps IOB to IOSB.
      7) Branch returns to IOS POST STATUS.

   b. If error, schedules ERP. (See 8.)

   c. Branches to EXCP back end for termination processing.

      1) Maps IOSB to IOB.
      2) Starts related requests.
      3) Unfixes buffer pages.
      4) Posts ECB (the one after the GET/PUT).
      5) Exits to the dispatcher.

8. ERP interface.

   a. If IBM ERP, get ERP work area.
   b. If DASD (IECVDERP), branch to ERP.
   c. If non-DASD, schedule ERP loader (IECVERPL) under SIRB. Use stage II exit effector to queue SRB to ASXBFSRB. Set stage II exit effector switch in ASCB.

GETMAIN/FREEMAIN

This chapter describes the processing for virtual storage requests in terms of GETMAIN processing and FREEMAIN processing. The flow through the GETMAIN/FREEMAIN process is complicated and the VSM control block structure should be understood prior to following this process. This process flow is not intended to explain how GETMAIN/FREEMAIN works but to provide an understanding of the important considerations of virtual storage management, how the important control blocks are manipulated, and the common subroutines of VSM.

GETMAIN Processing

The following describes the processing required to satisfy a given GETMAIN.

1. A problem program issues an SVC 10 GETMAIN for subpool 0 for 256 bytes.

2. GETMAIN (entry at IGC010) saves the TCB addresses in LDA, sets GETMAIN/FREEMAIN's FRR (module IEAVGFRR), sets up the length and subpool ID for common processing routines, saves the caller's mode in LDARQSTA, and goes to the common GETMAIN routine, GMCOMM1.

3. GMCOMM1 goes to routine UG4SP0 to find the SPQE for the requested subpool 0, PGTSPQE2 is called to search the TCBMSS chain. If no SPQE is found, PGTSPQE2 saves the address of the previous SPQE on the chain in SPQE SAVE.

   UG4PS0 then calls routine GETSPQEC to get a 16-byte element to build and chain-in an SPQE for the requested subpool. The 16-byte blocks for internal control blocks are obtained via GETMAINC (a simple GETCELL function). After obtaining the SPQE, module IEAVGM04 is branched to and processing continues at label IEAVGM4G.

4. At label ROUND, the request is rounded up to a doubleword boundary.

5. GMCOMM calls GFRECORE to search the FQEs pointed to by each DQE for the appropriate subpool. A first-fit algorithm is used to find a free element large enough to satisfy the request. ***Exception:*** LSQA/SQA requests for 4096 bytes or less are not satisfied across page boundaries because the request can be for page or segment tables that must reside in contiguous real storage.

6. If storage is found in an FQE, GFRECORE calls GFQEUPDT to maintain and update the FQE chain. (Control is passed to step 9.)

7. If storage is not found in an FQE, GFRECORE determines the number of 4K-blocks that are required and calls G4KSRCH to satisfy the request.

8. G4KSRCH performs the following functions:

   a. Calls FBQSRCH to search the appropriate FBQE chain to find 4K bytes of free space. (For problem program subpools, TCBPQE points to PQE which points to FBQE.) Once found, FBQSRCH removes the space from the FBQE and, if the FBQE is empty, frees it via an internal FREEMAIN (FMAINB in module IEAVGM00) or an internal freecell (FMAINC in module IEAVGM00).

b. Acquires a DQE and chains it onto the DQE chain anchored in the SPQE.

c. Calls RSM (IEAVFP1) to locate the page table entry (PGTE) and the external page table entry (XPTE) of the new 4K-block. Then at label SETUPPTE it initializes both the GETMAIN-assigned flag (PGTPAM) in the PGTE and the XPTPROT (protection key) in the XPTE (+0). *Note:* This is the only place XPTPROT is set.

d. Updates the SMF region usage fields of the TCT (task control table).

e. Creates an FQE and chains it from the DQE that was just built.

f. Returns to GMCOMM1.

9. GMCOMM1 places the address of the allocated storage in register 1 and sets the return code. Then GMCOMM1 performs housekeeping of any areas chained from FMAREAS in the LDA, deletes the FRR, and passes control to the EXIT prologue.

FREEMAIN Processing

The following is a logic flow of the FREEMAIN process when a problem program issues an SVC 10 requesting 256 bytes from subpool 0.

1. Upon entry at IGC010, FREEMAIN:

   a. Saves the TCB address in LDA.
   b. Establishes the FRR (IEAVGFRR).
   c. Saves the callers mode in LDARQSTA.
   d. Sets up the length and subpool ID for common processing.
   e. Passes control to FMCOMM1.

2. FMCOMM1 passes control to FMCOM because the request is not to free an entire subpool. FMCOM calls PGTSPQE2 to locate the SPQE. When the appropriate SPQE is found, control passes to module IEAVGM04 (at label IEAVGM4F) where processing continues. The associated DQEs are searched to locate the one DQE that describes the area to be freed.

3. The routine at label QELOCATE ensures that the area is not already described in an FQE (if it is, the requestor is abnormally terminated). Subroutine CREATFQE obtains a 16-byte element for an FQE, then builds the FQE and adds it to the proper FQE chain. *Note:* If possible, FQEs are combined if the new free space is adjacent to free space described by an existing FQE.

4. If less than 4K bytes are freed, FREEMAIN has completed its task and control is passed to the EXIT prologue.

5.

   a. If all space described by a DQE has become free, FREEMAIN frees the FQE and DQE and notifies RSM (IEAVRELV) that a page(s) can be released.

   b. If a virtual page is freed, FREEMAIN frees the FQE (and adjusts the DQE if the free pages exist at either end of the described area) and notifies RSM (IEAVRELV) to release the page(s).

   c. If the free page exists in the middle of the area described by the DQE, FREEMAIN obtains a new DQE and the two DQEs will now describe the area (essentially the area has been split into two parts). FREEMAIN updates the associated FQEs and notifies RSM (IEAVRELV) to release the page(s).

      Note: RSM invalidates the PGTE(s) for the associated pages being freed and calls ASM to release the auxiliary storage copy of the page. If a page table has become completely free, IEAVGM04 is passed the PGT address, which is queued from a field in the LDA (FMAREAS) to be freed at exit time. FMAREAS is really a list of items no longer required to describe virtual storage.

6. After restructuring the DQEs, MRELEASE returns virtual space to the appropriate FBQEs. If possible, MRELEASE places 4K blocks of storage in an existing FBQE; if not, it builds a new FBQE and includes it in the existing FBQE chain.

7. FREEMAIN returns to FMCOMM1A, which performs FMAREAS bookkeeping, deletes the FRR, and returns to the caller.

   Note: FMAREAS anchors a one-way chain of areas to be freed. The area itself contains the address of the next area at offset +0 and the subpool's ID and length at offset +4. These areas are not freed immediately, because freeing them might cause register save area overlays on the double recursion into FREEMAIN processing.

VTAM Process

The following shows the logic flow through the VTAM component into IOS and out to the 3705 when an application issues a SEND request. This description includes the major module flow and the control blocks required in order to process the request. Note that this is a general processing flow; additional modules not shown can be entered depending on options and device type. Figure A-5 illustrates the system modes at various stages of the VTAM processing.

1. The application program issues the VTAM SEND macro, passing an RPL (request parameter list), which points to the data that is to be sent.

2. The SEND macro branches to a VTAM interface routine (ISTAICIR).

3. ISTAICIR determines that this is a non-authorized request and issues the VTAM SVC. This is a type 1 SVC (SVC 124).

4. The type 1 SVC routine (ISTAPC22) obtains an MQL (MPST queueing element), places the address of the user RPL in it, and issues the TPQUE macro to queue the MQL to the TPIO PAB for the application's address space.

5. The TPQUE macro (normally) issues the TPSCHED macro in order to schedule the TPIO PAB.

6. The TPSCHED macro invokes ISTAPC32, which queues the TPIO PAB to the memory process scheduling table (MPST), and schedules an SRB to execute ISTAPC55.

7. ISTAPC32 returns to ISTAPC22.

8. ISTAPC22 issues a Type 1 exit back to ISTAICIR.

9. ISTAICIR determines if the request was synchronous or asynchronous. If it was synchronous, is issues the WAIT macro. If it was asynchronous, it returns control to the application program.

10. When the SRB is dispatched, ISTAPC55 dequeues the TPIO PAB from the MPST, obtains a component recovery area (CRA) from the large pageable (LP) pool and passes control to ISTAPC57.

11. ISTAPC57 formats the request parameter header (RPH) (within the CRA), dequeues the MQL from the TPIO PAB, and passes control to ISTAPC23.

12. ISTAPC23 release the MQL, obtains a Copy RPL (CRPL) from the CRPL pool and copies the user RPL into it.

13. ISTAPC23 then issues the TPQUE macro to queue the CRPL to the control layer outbound PAB in the appropriate FMCB and schedules control layer processing.

14. ISTAPC23 then issues the VTAM TPEXIT macro, which passes control to ISTAPC31.

15. ISTAPC31 recognizes that there is more work to do (control layer processing) and passes control to ISTAPC57.

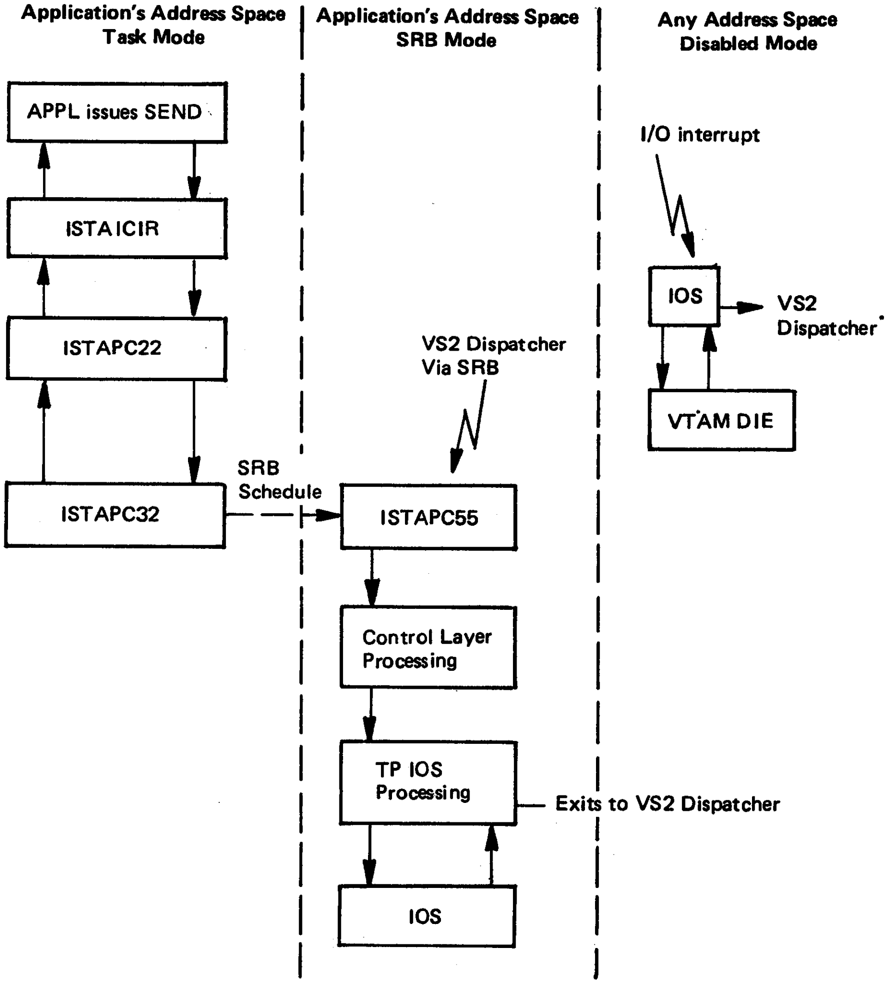


Figure A-5. VTAM SEND Process Flow

16. ISTAPC57 reformats the RPH (within the same CRA) for processing by the control layer.

17. ISTAPC57 then passes control to the control layer (ISTDCC00).

18. ISTDCC00 recognizes that this is a SEND request, obtains a logical channel program block (LCPB) from the CRPL pool, and invokes ISTRCC22.

19. ISTRCC22 sets up the logical channel command words (LCCWs) in the LCPB from the options in the CRPL and issues the TPQUE macro to queue the LCPB to the TPIOS outbound PAB in the FMCB and schedule TPIOS processing.

20. ISTRCC22 then passes control to ISTCDD00, which issues the TPEXIT macro.

21. The TPEXIT macro passes control to ISTAPC31, which recognizes that there is more work to do (TPIOS processing) and passes control to ISTAPC57.

22. ISTAPC57 reformats the RPH (within the same CRA) for TPIOS processing and passes control to ISTZAF1B.

23. Within TPIOS, ISTZDFA0 allocates the fixed I/O buffer; ISTZDFC0 and ISTZDFD0 move the user data to the I/O buffer.

24. Once the data is moved from the user's buffer, TPIOS invokes a routine (ISTRCFY0) which calls ISTAICPT. ISTAICPT copies the CRPL back to the user's RPL, frees the CRPL, and POSTs the ECB complete.

25. ISTRCFY0 then frees the LCPB and returns control to TPIOS.

26. TPIOS then invokes ISTZEMBB, which obtains the UCB lock for the 3705 and checks the ICNCB (intermediate controller node control block) to see if there is an active channel program currently executing for the 3705.

27. If the 3705 is busy, ISTZEMBB queues the I/O buffer to the ICNCB write queue, releases the UCB lock, and returns to TPIOS. (Go to step 29.)

28. If the 3705 is not busy, ISTZEMBB calls ISTZEMAB, which issues the STARTIO macro to IOS and then returns to ISTZEMBB, which returns to TPIOS. The IOSB, which is the interface to IOS, physically resides within the ICNCB.

29. After ISTZEMBB returns to TPIOS, TPIOS issues the TPEXIT macro, which invokes ISTAPC31.

30. ISTAPC31 recognizes that there is nothing more to do and calls ISTAPC58.

31. ISTAPC58 frees the CRA an exits to the VS2 dispatcher.

32. Sometime later, an I/O interrupt occurs as a result of the write channel program completing.

33. IOS passes control to the VTAM DIE (disable interrupt exit) (ISTZFM3B).

34. ISTZFM3B frees the I/O buffer and returns to IOS, indicating that POST STATUS should not be scheduled.

35. IOS exits to the dispatcher.

TSO

Following are some of the more important processes involved with the TSO/TIOC/TCAM interface portion of MVS. The processes are:

- Time Sharing Initialization
- LOGON Processing
- TSO Line Drop Processing
- TMP and Command Processor Interface
- TSO Command Processor Recovery
- TSO Terminal I/O Overview
- TSO/TIOC Terminal I/O Diagnostic Techniques
- TSO Attention Processing

Time Sharing Initialization

The system operator issues the MODIFY command (F TCAM, TS=START) to initialize the time sharing system. Terminal I/O control (TIOC) logic is documented in *OS/VS TCAM Level 10 Logic.*

The major functions that occur during time sharing initialization are:

1. The SYS1.PARMLIB member IKJPRMxx is read to determine the TIOC buffer size and number, the maximum number of time sharing users allowed to be logged on at one time, and thresholds for the maximum number of TIOC buffers a single user can use at one time.

2. The main control block for the time sharing system (TIOC reference table - TIOCRPT) is initialized. This control block points to the free queue of TIOC buffers and has status flags indicating whether the system is in an LWAIT (out of TIOC buffers). The TIOCRPT also points to a pool of terminal status blocks.

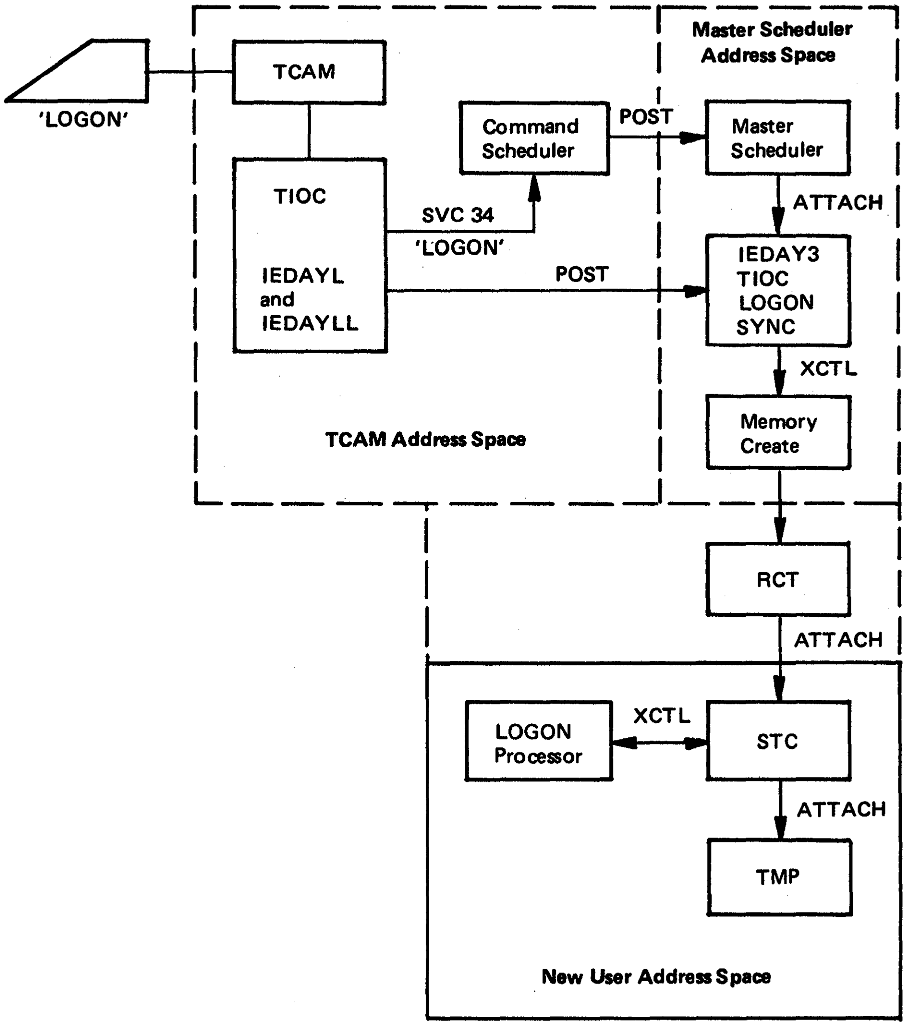
3. The pool of terminal status blocks (TSBs) is built. The number of TSBs is determined by the maximum user parameter in IKJPRMxx. A TSB is assigned to a user during logon processing. The TSB connects the ASCB of the user to the terminal-name table entry of the terminal. From the terminal-name table entry, TCAM can locate the terminal table entry for the user and hence the address of the destination QCB. The TSB contains input and output queues for TIOC buffers that are used by the time sharing user.

   The TSB also contains status indicators that record whether the user is in an input wait (TGET issued and no TIOC buffer on TSB input queue) or an output wait (maximum number of TIOC buffers used for output).

4. The TIOC buffer pool is built. The number and size of the buffers is determined from IKJPRMxx. If no parmlib member was specified on the MODIFY TCAM command, SYS1.PARMLIB is searched for the default parmlib member name - IKJPRM00. If this member is not found, standard default values are used.

5. The 'TSO HAS BEEN INITIALIZED' message is issued (via WTO).

Terminal User Issues LOGON



Note: Details of this process are shown in part 2 of this figure.

Figure A-6 (Part 1 of 2). Overview of Logon Processing

LOGON Scheduling

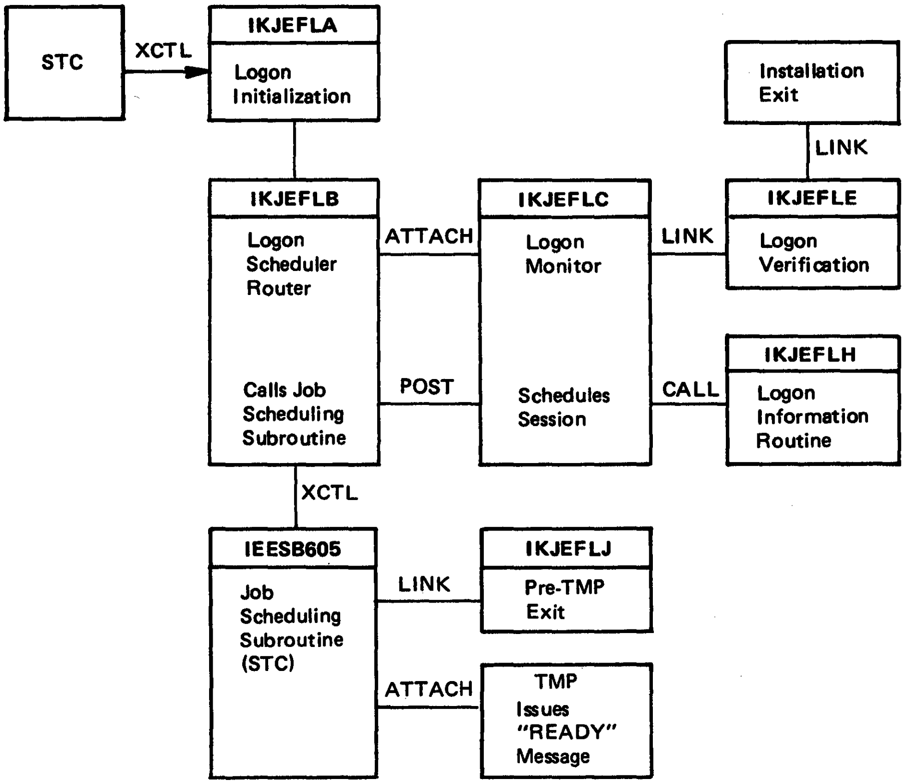


Figure A-6 (Part 2 of 2). Overview of Logon Processing

LOGON Processing

The major functions of LOGON processing are:

1. TCAM handles line I/O and routes the buffer to the TSO message handler. The message handler routes the buffer to various functional routines. One of these is logon.

2. The logon routine receives control from the TSO message handler as a result of the expansion of the LOGON macro. Logon routes the buffer to TSINPUT so that logon scheduling may retrieve it via a TGET SVC. TIOC logon then issues an SVC 34 to notify the master scheduler that logon processing should be started. TIOC then issues QTIP 10 to initialize control blocks. **Note:** QTIP is the TIOC code invoked when SVC 101 is issued. It performs functions related to communication between the TCAM and TSO user address spaces. The specific function it is to perform is indicated by an entry code (for example, QTIP 10). A table of entry codes, their callers, the functions performed, and the modules that provide the function is contained in *OS/VS TCAM Level 10 Logic.*

   QTIP issues an XMPOST to inform the master scheduler that TIOC initialization is complete and that memory create may begin. TIOC then returns to the message handler for final buffer disposition. If logon fails or is terminated, TIOC is notified so that the appropriate error message can be issued.

3. TSINPUT invokes QTIP to move the contents of the TCAM buffer to the TIOC buffers. This data can then be accessed using TGET services.

4. The master scheduler recognizes that a logon has been requested and attaches TIOC synchronization. This routine waits until QTIP signals with a post that memory create can begin. Once an address space has been initialized for the logon request, the region control task is the first task to be dispatched.

5. Region control establishes an ESTAE routine, attaches the dump task, attaches started task control, and waits for one of the following:

   - An attention request signaled by TIOC via XMPOST
   - A swap request signaled by SRM
   - A termination request

6. Started task control recognizes that logon is requested and passes control to logon initialization (IKJEFLA).

7. Logon initialization opens the UADS and broadcast data sets, initializes control blocks, and calls logon scheduling (IKJEFLB).

8. The logon load module contains four service modules. One, IKJEFLP0, contains the default values for the number of seconds requested between 'LOGON PROCEEDING' messages and the number of logon attempts allowed before automatic logoff. Both values are sysgen options on the TSO macro. The logon scheduler attaches the logon monitor (IKJEFLC). The scheduler and monitor now begin parallel processing. WAITs and POSTs are used when synchronization is required.

9. The logon monitor (IKJEFLC) builds the environment control table (ECT), sets the first element of the input stack to indicate terminal input, and links to logon verification.

10. Logon verification (IKJEFLE) calls the user's pre-prompt exit if it was coded. Logon verification makes the following checks:

    - Determines (via ENQ) if the userid is in use.
    - Checks the user's password, account number, and procedure name.
    - Checks the performance group requested in the LOGON command.

    Logon verification prompts the user for missing parameters if required parameters do not have defaults in the UADS. After all required parameters have been obtained, verification builds the JOB and EXEC statement images for the session. The EXEC statement contains the name of a logon procedure specified in the UADS or the LOGON command.

11. Logon verification posts the logon scheduler when the parameters are complete and the job can be scheduled. The scheduler's job now is to cause the broadcast messages to be listed at the terminal at the same time that the user's job is being scheduled. To do this, it posts the monitor task and then XCTLs to the initiator, passing it the JCL that has been created.

12. The logon monitor regains control when signaled by the logon scheduler, attaches the LISTBC command processor to write broadcast messages to the terminal, and then waits for a post from a special initiator logon routine. This post signals that final processing can be completed.

13. The initiator uses the TSO internal reader to send the logon job to JES2. JES2 reads the user's procedure from the procedure library specified by the &TSU job class parameter and changes the JCL to internal text. This is placed on the spool data set. Once this processing has completed, the initiator requests the user's job by ID and completes initiation and allocation. Initiation finally gives control to a special TSO routine (pre-TMP exit, IKJEFLJ). This routine posts the logon monitor and issues a WAIT. The logon monitor then terminates. This causes the initiator task to regain control. The logon monitor is then detached. Once the monitor is detached, the initiator attaches the TMP and waits.

14. The TMP (specified as IKJEFT01 in the LOGON PROC on the EXEC statement) performs initialization and then issues a PUTGET to write the 'READY' message and request a command from the user. This PUTGET results in a TPUT to send READY and a TGET to request terminal input. The user is now in an input wait. This signals SRM to perform a swap-out until input is available.

Figure A-7 shows TCAM's organization after a TSO logon. The following are detailed descriptions of the logon process including information on control block manipulation. The numbers in parentheses correlate to the numbers in the preceding summary of the logon process.

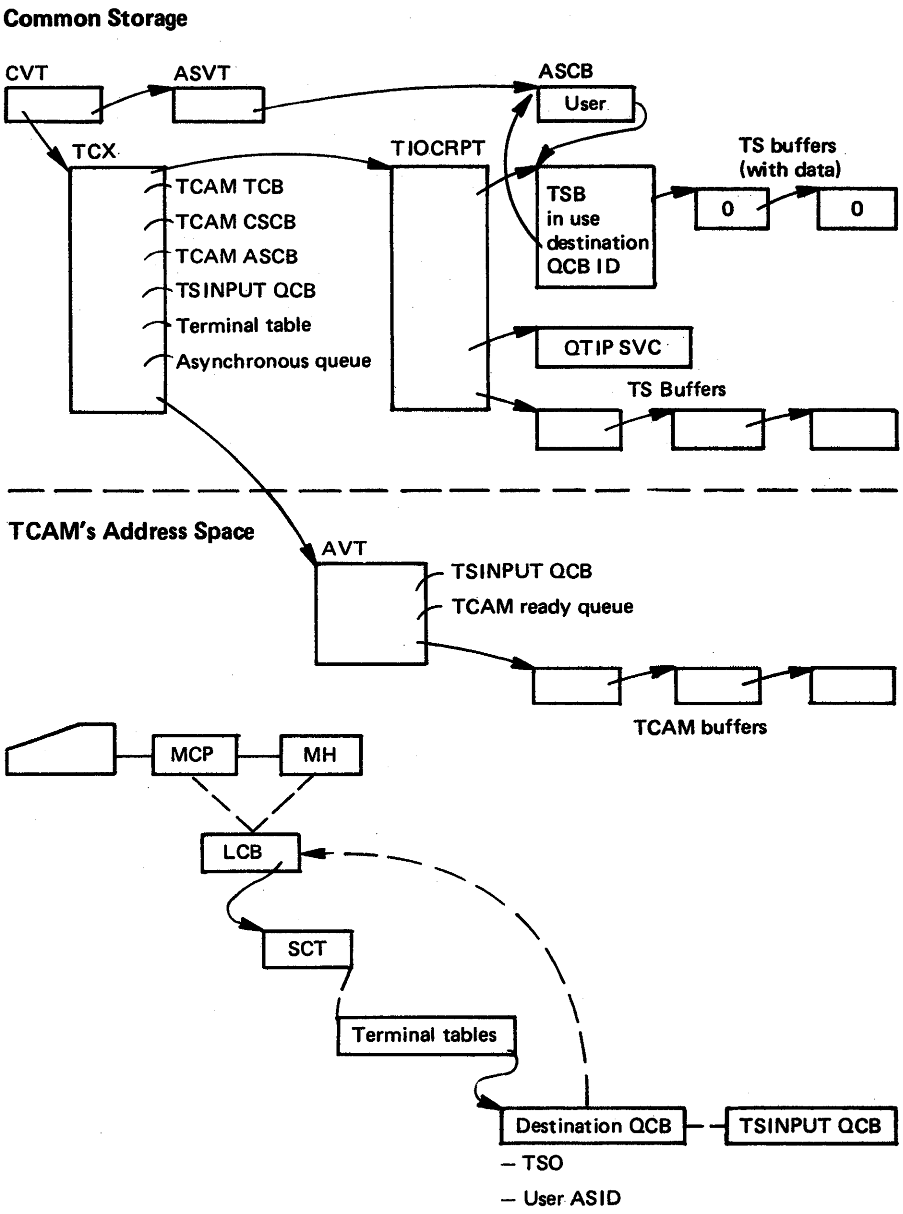


Figure A-7. TCAM Organization After a TSO Logon

TIOC Logon Processing (2):

- Checks the maximum user count in TIOCRPT.

- Issues SVC 34 'LOGON'.

- Places the returned ASID in the QCB for this line.

- Calls QTIP (entry 10) to find and initialize the TSB.

  - Puts TSB address in the ASCB for the user's address space.
  - Puts the ASCB address in the TSB.
  - Updates the user count.
  - Puts the UCB address in the TSB.
  - XMPOSTs 'TIOC SYNC'.

- Sets the QCB to indicate TSO.

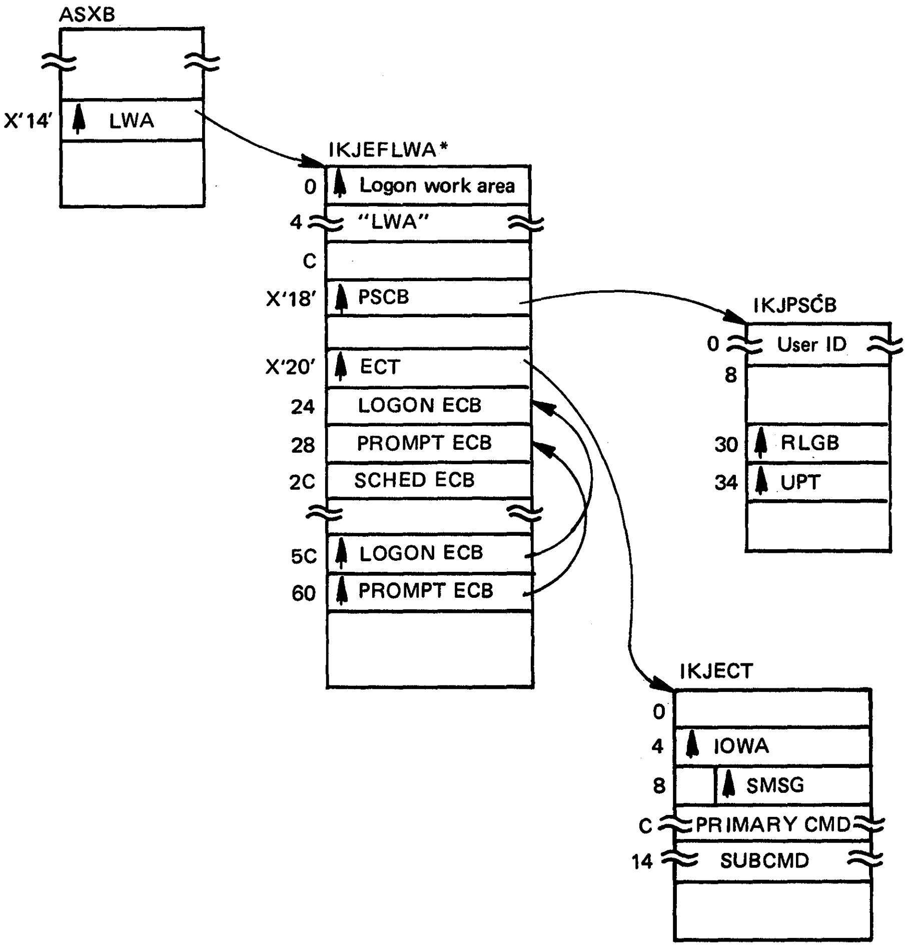
- Pass the logon message buffer to TSINPUT QCB (which is now available to system logon processing via GETLINE).

Logon Initialization (IKJEFLA) (7): Logon initialization uses the address of the ASCB as input and does the following:

- Ensures SYS1.UADS and SYS1.BRODCAST data sets are allocated.
- Gets the LWA (logon work area) from the LSQA. (See Figure A-8.)
- Puts the LWA address in the ASXB.
- Gets the JSEL (job scheduling entrance list) from the LSQA.
- Puts the CSCB and ASCB addresses in the JSEL.
- Gets the JSXL (job scheduling exit list) from the LSQA.
- Puts the LWA address in the JSXL. JSXL contains pointers to the PRE-TMP, POST-TMP, and PRE-FREEPART exits.
- Puts the JSXL address in the JSEL.
- Gets the UPT (user profile table) from subpool 230.
- Issues BLDL for the installation exit routine (Release 2 only).
- Gets the PSCB (protected step control block) from subpool 230.
- Puts the PSCB address in the LWA.
- Puts the UPT address in the PSCB.
- Gets the re-logon buffer from subpool 230.
- Puts the re-logon address in the PSCB.
- Calls the logon scheduler router.

Logon Scheduler Router (IKJEFLB) (8):

- Frees subpool 0.
- Attaches the logon monitor.
- Posts the monitor with the 'schedule' code.
- Waits for the 'what to do' post from the monitor.



*The logon work area (IKJEFLWA) is a 148-byte area that is created by IKJEFLA and is pointed to by ASXB and JSXL. It contains control block pointers, entrance lists, and parameter lists that are required for logon/logoff.

Figure A-8. Logon Work Area

Logon Monitor (IKJEFLC) (9):

- Switches the storage key to '8'.
- Gets the STAX work area from subpool 1.
- Gets the ECT (environment control table) from subpool 1.
- Puts the ECT address in the LWA.
- Invokes the STACK macro (input is to come from the terminal).
- Gets the new CSCB (command scheduler control block) from the SQA.
- Sets the CSCB to indicate to the master scheduler that the job is:
  - swappable
  - terminal job
  - cancellable
  - TSO
- Gets the local and CMS locks.
- Puts the CSCB address in the ASCB.
- Frees the local and CMS locks.
- Issues the MGCR macro to remove the old CSCB from the chain.
- Puts the new CSCB pointer in the JSCB and JSEL.
- Issues the MGCR macro to add the new CSCB to the chain.
- Issue the STAX macro to set up attention handling.
- Links to logon verification.

Logon Verification (IKJEFLE) (10):

- Calls the installation exit (if necessary).

- Issues GETLINE or uses the installation supplied buffer containing the logon parameters.

- Calls the command scan service routine to ensure that input is the LOGON or LOGOFF command (assumes LOGON).

- Calls PARSE for logon parameter parsing.

- Indicates no password required for the UADS.

- Issues ENQ on the UADS (prevents the ACCT CP from changing UADS).

- Opens the UADS.

- Issues FIND for the userid member (userid is taken from the logon parameter).

- Places the userid in the PSCB (protected step control block).

- Posts the logon scheduler.

- Waits for the post from the logon scheduler.

Logon Scheduler (IKJEFLB):

- Enqueues (via ENQ) on SYSIKJUA.USERID.
- Posts logon verification.
- Waits for logon verification.

Logon Verification (IKJEFLE):

- Dequeues (via DEQ) from UADS.
- Puts the userid in the CSCB.
- Puts the userid in the ASCB.
- Enqueues (via ENQ) on UADS.
- Finds userid member.
- Dequeues (via DEQ) on UADS.
- Reads UADS.
- Issues check.
- Places the parameter in the proper control block.
- Places the password in the TSB.
- Places the procname in the CSCB.
- Places the region size in the PSCB.
- Informs SRM of the performance group.
- Builds the JCL:
  //USERID JOB 'account#',REGION=region size
  //procname EXEC procname, PERFORM=performance group
- Issues the 'LOGON IN PROGRESS' message to the terminal.
- Closes the UADS.
- Clears 'NO PASSWORD' in the JSCB.
- Dequeues (via DEQ) from UADS.
- Posts the logon scheduler to schedule the session. (11)
- Waits for the logon scheduler.
- Sends the broadcast messages (via the information routine). (12)
- Issues the 'LOGON IN PROGRESS' messages until posted by the initiator.
- Frees subpool 78.

Logon Scheduler (IKJEFLB) (11):

- Sets up the interface to JSS.
- Posts the logon monitor.
- XCTLs to JSS (initiator).

Job Scheduling Subroutine (IEESB605) (13):

- Calls the PRE-TMP exit.

PRE-TMP Exit (IKJEFLJ):

- Posts the monitor task to terminate.
- Moves the PSCB from (unaccountable) subpool 230 storage to (accountable) subpool 252 storage. The PSCB address is placed in the active JSCB.
- Moves the UPT and the re-logon buffer to 0 (allows updating by CPs).
- Returns to the initiator.

Initiator:

- Attaches TMP (PARM='xxx...' is passed).

TMP (14):

- Issues "READY" message.
- Requests terminal input.

LOGON Scheduling Diagnostic Aids

The following two figures contain information that can be used for diagnosing problems that occur during logon scheduling.

Field Name and Contents		Name of Executing Module	Common Name of Module
LWAINX1	=1	IKJEFLD	Installation Exit (written by installation)
LWALA	=1	IKJEFLA	LOGON Initialization
LWALB	=1	IKJEFLB	LOGON Scheduling
LWALC	=1	IKJEFLC	LOGON Monitor
LWALE	=1	IKJEFLE	LOGON/LOGOFF Verification
LWALEA	=1	IKJEFLEA	Parse/Scan Interface
LWALI	=1	IKJEFLI	Installation Interface
LWALH	=1	IKJEFLH	LOGON Synchronizer
LWALL	=1	IKJEFLL	LOGOFF Processing
LWALGM	=1	IKJEFLGM	LOGON Message Handler
LWALJ	=1	IKJEFLJ	Pre=attach Exit
LWALK	=1	IKJEFLK	Post-attach Exit
LWALG	=1	IKJEFLG	Attention Exit
LWALGB	=1	IKJEFLGB	LOGON Monitor Recovery
LWALS	=1	IKJEFLS	LOGON Scheduling Recovery and Retry
LWALTBC	=1	IKJEFLH	Mail and Notices Processing
LWAMCK		IKJEFLGB	ABEND was a machine check
LWAPCK		IKJEFLGB	ABEND was a program check
LWAPHASE	=0	Any LOGON module except IKJEFLH	LOGON/LOGOFF Verification
LWAPHASE	=1	IKJEFLH	LOGON Synchronizer
LWAPSW		IKJEFLGB	Console Restart key depressed
LWATNBT		IKJEFLG	Attention Routine

Figure A-9. LOGON Work Area Bits That Indicate the Currently Executing Module

Module Issuing POST	Module Being Posted	Location of ECB	Post Code	Condition of Module Issuing POST	Action Taken by Module Being Posted
IKJEFLB	IKJEFLC	field LWASECB in LWA	16	Ready to invoke job scheduling subroutine (IEESB605).	Invoke LOGON information routine (IKJEFLH).
			24	Terminating for LOGOFF or for unusual termination of LOGON monitor (IKJEFLC).	Perform clean-up operations and terminate.
IKJEFLC	IKJEFLB	field LWAPECB in LWA	12	Termination or attention requested.	Issue DEQ on user identification.
			16	Verified and processed the LOGON parameters.	Schedule a terminal session.
			24	Processing a LOGOFF command.	Terminate.
IKJEFLE	IKJEFLB	field LWAPECB in LWA	8	Authorized the user identification.	Issue ENQ on user identification.
			12	Error processing.	Issue DEQ on user identification.
IKJEFLJ	IKJEFLH	field LWASECB in LWA	20	Detects that the initiator is ready to attach the TMP.	Finish LISTBC processing; return to caller.
IKJEFLH	IKJEFLJ	field LWAPECB in LWA	20	Finished LISTBC processing.	Terminate so the initiator can attach the TMP.

Figure A-10. LOGON Scheduling Post Codes

TSO Line Drop Processing

The following description corresponds to the overview of line drop processing shown in Figure A-11.

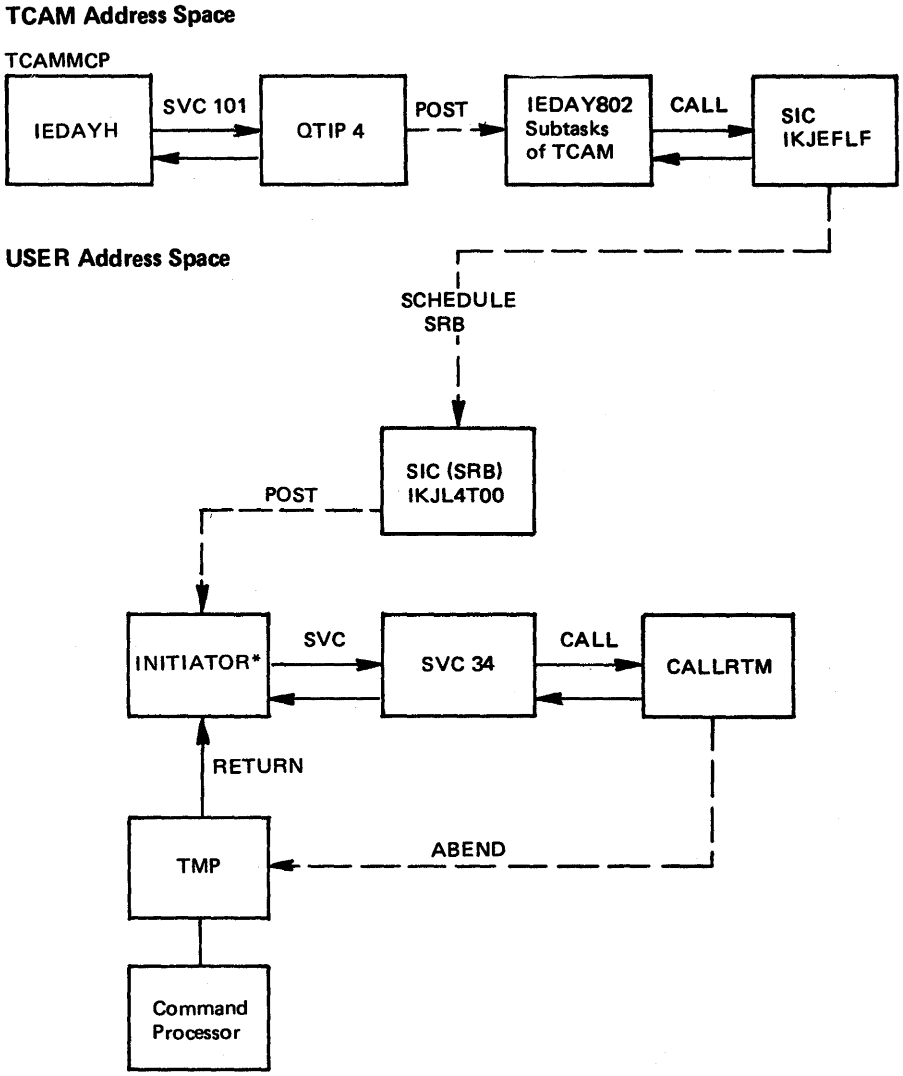
IEDAYH (Part of TCAM MCP):

- Gets control from the TCAM dispatcher when either of the following occurs:
  - A hang up on a monitoring channel program or a message generation.
  - Each input or output message ends.
- Tests for and handles several kinds of errors. If it discovers the line has dropped, it begins terminating the user. Each of the following is considered a line drop:
  - Entry because of a hang up on a monitoring channel program or a message generation
  - A 3705 control unit error - indicated in the SCB (station control block)
  - A permanent terminal error - indicated in the SCB
  - A countable error and an appropriate number of retries have been done - indicated in the SCB
- If a line drops, issues a QTIP 4 (SVC 101, entry code 4).

QTIP 4 (IEDAYHH):

- TSBHUNG=1.
- Issues QTIP 28 to free the TCAM buffers.
- If the reconnect time limit is 0 (in TIOCRPT), then branch enters SIC (system-initiated-cancel IKJEFLF) with code 622; upon return, returns to caller.
- For a non-0 RECONLIM:
  - Sets TSBMINL equal to the reconnect time lımıt.
  - If TIOCTECB (in TIOCRPT) is posted, then increases the value in TSBMINL by one. Otherwise, posts TIOCTECB (which IEDAY802, running as a subtask of TCAM, is waiting for).
- Returns.

LINE DROP IN TSO ENVIRONMENT



*Upon return, continues with normal logoff.

Figure A-11. Overview of TSO Line Drop Process

IEDAY802 (subtask of TCAM): Keeps track of users whose lines have dropped and, if the time limit expires before they come back, terminates the address space. IEDAY802 does the following:

- Waits for TIOCTECB.

- Sets the one-minu. timer.

- Invokes QTIP 27 (IEDAY88) SVC 101, entry 27 which scans the TSBs for TSBHUNG=1 and TSBMINL"0.

  - If so, QTIP 27 decreases TSBMINL by 1.

  - If TSBMINL is now 0, QTIP 27 branch enters SIC (system-initiated-cancel) with code 622.

  - QTIP 27 returns a code of 0 if any users have time left or a code of 4 if all users have been cancelled.

- If the return code is 0, IEDAY802 goes to the one-minute timer.

- If the return code is 4, IEDAY802 waits for TIOCTECB.

SIC (system-initiated cancel):

IKJEFLF schedules an SRB in the address space to be terminated, passes a completion code (622 for line drop), and returns to the caller.

IKJL4T00 runs under the SRB scheduled by IKJEFLF and gets control the next time the address space is dispatched. IKJL4T00 does the following:

- If TMP is in control, skips to POST.
- Issues STATUS STOP for TCB= (IWAIT/OWAIT dispatchability bits).
- Issues QTIP 24, which sets TSBCANC=1 and removes OWAIT for other address spaces TPUTing to this user.
- POST cancels the ECB in the CSCB (IKJL4T00 branch enters POST with completion code 622). The initiator (IEFSD263) waits for the ECB while TMP is in control.
- If TMP is not in control, issues STATUS START for the logon scheduler and monitor tasks.
- Exits.

Initiator (IEFSD263):

- Waits for the CANCEL ECB and ATTACH ECB of the TMP task.
- When the CANCEL ECB is posted, issues SVC 34 to abnormally terminate the user.

SVC 34:

- Issues CALLRTM, which sets the resume PSW of the TMP task to point to an SVC D instruction and forces the TMP task to be dispatchable.

SVC D (RTM2):

- Oversees the termination of the TMP task and all daughter tasks.
- When the TMP task terminates, its attach ECB is posted, giving the initiator control again.

Initiator:

Processing continues the same as for normal logoff except:

- IKJEFLK, the POST-TMP exit module, issues QTIP 24.
- IKJEFLC issues the "session cancelled" message before the logon scheduler XCTLs to the STC termination.
- If the line drops, IEDAY8, the TIOC resource manager, does not force the remaining messages out.

TMP and Command Processor Interface

The following is a description of the TMP and command processor flow.

1. The TMP is attached by the initiator as a result of a logon command from a terminal user or the execution of a batch job. Logon initialization establishes the STAE environment to handle abends and the STAX exit to handle attention interrupts.

2. The TMP mainline routine receives control and determines which buffer to obtain. This can be either:

   a. The logon buffer (from PART= on the EXEC statement of the logon procedure)

   b. The command buffer, as a result of a PUTGET

   c. The buffer obtained by the attention prolog

   d. The buffer obtained by the STAI exit

3. If the current input is the command buffer, the TMP must check for five special cases as follows:

   a. PUTGET is responsible for checking for a '?' in the first buffer position in response to a mode message. When one is detected PUTGET immediately issues the next available second-level message. This TMP should never receive a '?' in a buffer, but if the user enters a 'b̸?' (blank ?), PUTGET passes the buffer through to the TMP.

   b. A null line.

   c. TEST command without operands.

   d. TIME command.

e. If scan determines that the data in the buffer is not one of these special cases and that the data begins with an alphabetic character and is less than eight bytes, the TMP issues an ATTACH for the command name. Prior to ATTACH processing a search is conducted (through MLPA, LPA, joblib, LNKLSTxx, respectively) to assure a successful ATTACH. If the ATTACH is not successful, the TMP assumes a CLIST and attaches the EXEC CP to search the user's command procedure library. If the TMP does not locate either a command or a command procedure whose name is the same as that found in the input buffer, a 'COMMAND NOT FOUND' message is issued to the terminal.

4. If the command processor was attached, the TMP waits for an ECB list containing the following ECBs:

   a. STAI ECB: The TMP's STAI exit routine posts this when a command processor abnormally terminates and does not recover with its own STAE routine.

   b. Attention exit ECB: The TMP's attention exit routine posts this when it gains control. It gains control when the user enters an attention interrupt and the TMP exit is the current level exit. For more details, see the discussion of "TSO attention processing" later in this chapter.

   c. STOP/MODIFY ECB: This ECB is posted if a stop userid is requested by the system operator.

   d. Command processor ECB: This is the ECB specified in the attach of the command processor. It is posted when the processor terminates.

5. If the command processor ECB is posted, the TMP repeats step 2 to determine what action to take.

6. If the attention exit or STAI ECB is posted, the TMP does one of the following:

   a. If a 'ɓ?' was entered in response to the mode message, the TMP sends second level messages to the terminal.

   b. If a null line was entered, TMP returns control to the command processor. If an attention interrupt occurred, the TMP continues normal processing. If an abend occurred, the TMP takes a dump.

   c. If TEST was entered without operands, the TMP links to TEST and places the interrupted command processor under test control. When TEST processing is ended, the TMP detaches the current command and prompts the user with a 'READY' to enter a new command.

   d. If the TIME command was entered, the TMP displays the current time and prompts the user for a new command. In this case, the user can exercise any of the preceding options or enter a new command.

   e. If the user enters a new command or exercises one of the preceding options, the TMP detaches the current command and issues a PUTGET requesting new input.

The following common control blocks are used for communication among the TMP, command processors, and service routines (PUTGET, PARSE, etc.):

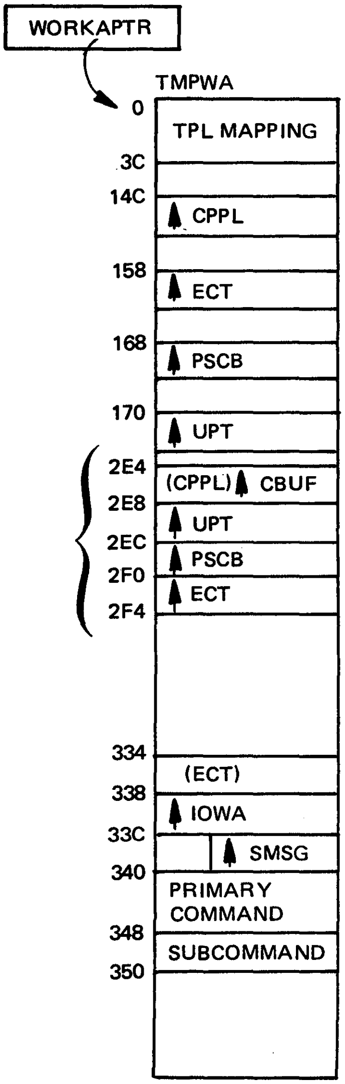
IKJTMPWA (TMP Work Area)

Created by: IKJEFT01

Length: 1076 bytes

Pointed to by: TMPWAPTR, WORKAPTR

Function: Provides communication among TMP modules. Contains register save areas, parameter lists for TEST and TMP, ABEND exit routines, and mappings of macros commonly used by TMP modules.



IKJCPPL (Command Processor Parameter List)

Created by: IKJEFT01

Length: 16 Bytes

Pointed to by: Register 1

Function: Provides parameters for the command processor.

IKJECT (Environment Control Table)

Created by: IKJEFT01

Length: 40 bytes

Pointed to by: TPL, CPPL

Function: Provides communication among the TMP, CP, and service routines. Contains current command/subcommand names, pointers to work areas and second-level message chains, and return codes.

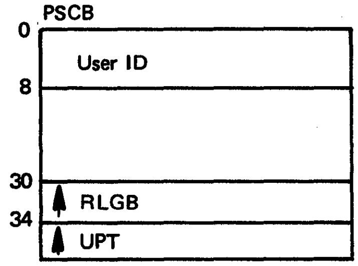
IKJPSCB (Protected Step Control Block)

Created by: IKJEFLA

Length: 72 bytes

Pointed to by: LWA, CPPL

Function: Contains information from UADS, control bits, and accounting data for the user ID. (This accounting data is controlled by the installation via the ACCOUNT command.)



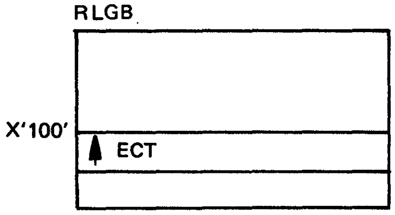
IKJRLGB (Re-Logon Buffer)

Created by: IKJEFLA

Length: 264 bytes

Pointed to by: PSCB

Function: Contains the LOGON/LOGOFF command entered at the terminal at the end of the session.



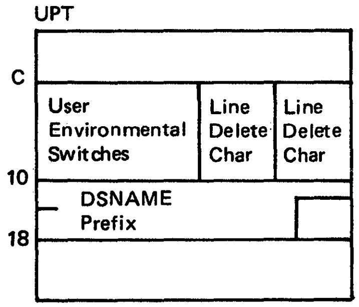
IKJUPT (User Profile Table)

Created by: IKJEFLA

Length: 24 bytes

Pointed to by: PSCB, CPPL

Function: Contains information stored in UADS that is used by LOGON/LOGOFF, the TMP, and the command processors. (This information is all controlled by the installation via the PROFILE command.)



TSO Command Processor Recovery

The following describes the TSO command processors. Figure A-12 summarizes their recovery activity.

ACCOUNT

The STAE exit routine for ACCOUNT flushes the input stack and posts the ACCOUNT ECB before returning to continue abend processing. ACCOUNT attaches the HELP command processor, specifying for a STAI exit routine the same name as the STAE exit routine.

EDIT

The ESTAE exit routine for EDIT flushes the input stack, stops automatic line prompting, and frees any acquired storage still remaining. The EDIT work area, mapped by IKJEBECA, can be located in a dump to obtain certain data on the EDIT session. The pointer to the communication area is passed between routines in register 0. By convention, most routines keep the pointer in register 9 during execution. A description of IKJEBECA can be found in the data areas microfiche (*Data Areas*).

LOGON

The ESTAE exit routine for LOGON dequeues from the userid, closes the UADS data set, and detaches IKJEFLC. The LOGON work area, mapped by IKJEFLWA, can be located in the dump (field ASXBLWA in the ASXB) to obtain certain information on the session. A description of IKJEFLWA can be found in the data areas microfiche.

LOGON also has an ESTAI exit routine, which dequeues from the userid, closes the UADS data set, cancels the attention exit, and frees subpools 1 and 78.

OPERATOR

The STAE exit routine for OPERATOR stops all active monitor function if the abend is caused by a DETACH with STAE. OPERATOR also has a STAI exit routine that is the same name as the STAE exit routine.

The SVC 100 parameter list, mapped by IKJEFFIB and passed to the OPERATOR command processor, can be located in the dump and certain data about the session can be obtained. A description of IKJEFFIB can be found in the data areas microfiche.

OUTPUT

Before returning to continue abend processing, the ESTAE exit routine for OUTPUT closes any data sets that are being processed. The OUTPUT work area, mapped by IKJOCMTB, can be located in a dump (while OUTPUT is in control) and certain data about the session can be obtained.

OUTPUT attaches the HELP command processor specifying a STAI exit routine. The STAI exit routine simply returns to continue abend processing.

SUBMIT

The SUBMIT command processor runs under the STAI environment established by SVC 100. This STAI routine closes the INTRDR data set before it returns to continue abend processing. The SVC 100 parameter list, mapped by IKJEFFIB and passed to the SUBMIT command processor, can be located in the dump and certain data on the session can be obtained. A description of IKJEFFIB can be found in the data areas microfiche.

Command Processor	STAE ESTAE	STAI/ ESTAI	RETRY	SDUMP	LOGREC	Messages
ACCOUNT	STAE	STAI				IKJ56554I IKJ56554I
EDIT	ESTAE		X			See Note 1
LOGON	ESTAE	ESTAI	X	X X		IKJ56452I IKJ56451I IKJ56452I IKJ56406I
OPERATOR	STAE	STAI	X X			IKJ55004I IKJ55004I
OUTPUT	ESTAE	STAI		See Note 2	See Note 3	IKJ56318I
SUBMIT		STAI				IKJ56294I

Notes:

1. *Abend codes B37, D37, and E37 point to IKJ52427I, IKJ52428I; the others point to IKJ52422I. If the data set is modified, abend codes point to IKJ52555I.*
2. *SDUMP is issued for all abends except for DETACH with STAE, codes 437, 913, and 422.*
3. *LOGREC is written to except for DETACH with STAE.*
4. *An effective trapping and problem solving technique for TSO command processors is to stop the error processing in the appropriate error recovery routine.*

Figure A-12. Summary of Command Processor Recovery Activity

TSO Terminal I/O Overview

Terminal I/O flow is divided into two parts, input flow and output flow. This overview highlights each at the SVC level.

TS/TCAM uses the services of three SVCs to communicate between the user's address space and the TCAM address space:

1. ***TGET/TPUT (SVC93):*** The TMP and command processors issue this SVC to move data form the user's buffer to an interface buffer in CSA (TIOC buffer).
2. ***QTIP (SVC 101):*** This SVC is a set of multipurpose routines that perform functions for both the user address space and the TCAM address space. For example, QTIP is used by TCAM to move data from a TCAM buffer to an interface (TIOC) buffer and is also used by TGET/TPUT to move data from a user's buffer to a TIOC buffer.
3. ***STCC (SVC 94):*** This SVC is a set of routines used to update TCAM control blocks from the user's address space. For example, the user can use

the terminal command to change a terminal characteristic. This is communicated to TCAM via SVC 94.

TS/TCAM data flow also requires a logical connection between a terminal, a line, and an address space. This is accomplished as follows:

- The terminal macro in the user's MCP establishes the connection between a terminal name and a destination (destination QCB).
- At TCAM initialization, OPEN establishes the connection between the destination and a physical terminal (a line control block is connected to the terminal name table via an index into the table).
- Logon processing establishes the connection between the destination QCB and the user's address space (the destination QCB contains the ASID of the user and the user's terminal status block (TSB) contains an index to the TCAM terminal name table). Also, a user's TSB and ASCB point to each other. The station's control block contains the address of the TSINPUT QCB.

Terminal I/O flow also requires the use of two special TCAM subtasks: TSINPUT and TSOUTPUT. TSOUTPUT acts as the router for all messages coming from time sharing users. TSOUTPUT is responsible for editing output messages as it moves the data from the time sharing interface buffers (TIOC buffers) in CSA to the TCAM buffers in the TCAM address space. Once TSOUTPUT has moved data to the TCAM buffers, the buffer is routed to the output side of the message handler and then written to the terminal.

TSOUTPUT also runs as a subroutine of TCAM. TSOUTPUT is the first subroutine in control of the disk I/O QCB in a TCAM system that supports time sharing.

Terminal Output Flow

Assume that a user has logged on, the TMP has been initialized, and a PUTGET has been issued by the TMP to put out a 'READY' message and request input from the terminal user. The following now occurs:

1. The TMP invokes the services of the PUTGET service routine, which issues a TPUT and then a TGET (both SVC 93s). TPUT performs the following basic functions:

   a. Obtains a TIOC buffer from the pool of free buffers. If a buffer is not available or the user has passed the output buffer limit (OWAITHI parameter in IKJPRM00), the user is placed in an output wait (the appropriate flag is set in the TSB).

   b. If a buffer is available, the 'READY' message is moved from the user's buffer to the TIOC buffer.

   c. The user's terminal status block is placed on TCAM's asynchronous ready queue. (A special element at TSB+X'40' is used.)

   d. An XMPOST is done to alert TCAM.

   e. Control is returned to PUTGET.

2. When the TCAM address space is dispatched, and the MCP TCB regains control, TCAM searches its asynchronous ready queue and discovers the user's TSB. However, because this is a TS/TCAM system, TSOUTPUT receives control instead of the disk I/O routine. TSOUTPUT performs the following functions:

   a. Builds TCAM buffers from basic TCAM buffer units.

   b. Uses QTIP services to move the TIOC buffer from the TSB header queue (queue of complete output messages) to the TSB output trailer queue (queue of TIOC buffers being moved).

   c. Uses special TIOC edit routines (not QTIP) to move and edit data from the TIOC buffer to the TCAM buffer.

   d. Once the data has been moved into the TCAM buffers, the TCAM buffers are routed to the output side of the message handler and are then written to the terminal. After the message is successfully written, the TIOC buffers are freed via a subsequent call to TSOUTPUT.

Terminal Input Flow

The following process can run in parallel with step 2 in the preceding section, "Terminal Output Flow." It starts when control is returned to PUTGET as described at the end of step 1 in that section.

1. PUTGET issues a TGET to obtain input. TGET (SVC 93) performs the following functions:

   a. Checks to determine if there is an input buffer on the user's terminal status input queue. TCAM normally allows users of remote terminals to enter input while the current input is being processed. Therefore, it is possible that input could be 'stacked' and an input buffer found on the TSB input queue. However, TCAM does not allow local devices to 'stack' input. In this case, assume a local device and no buffer on the TSB input queue.

   b. Therefore, the TGET notifies SRM that an input WAIT has been entered and sets the appropriate flag in the TSB (IWAIT condition).

   c. SRM eventually performs a swap-out on the user.

2. The user now enters a new command at the display station and hits 'ENTER'. TCAM handles the interrupt, associates it via the LCB to a terminal name table index, terminal table entry, and destination QCB.

3. The TCAM buffer is routed to the input side of the appropriate message handler (determined from the DCB for the line). The message handler normally translates the data from line code to EBCDIC. The message handler must locate the destination QCB of the terminal that issued the message and also check that the terminal is logged on to time sharing. If it is logged on, the message handler routes the buffer to TSINPUT as the common input destination for all time sharing messages.

4. TSINPUT performs the following functions:

   a. From the ASID value in the terminal's destination QCB, TSINPUT determines which address space should receive a particular message.

   b. TSINPUT obtains a TIOC buffer from the free buffer pool. If no TIOC buffers are available, the TCAM buffer is chained from a special queue in the TSINPUT QCB until TIOC buffers are made available. In this case, the time sharing system is placed in an LWAIT (out of TIOC buffers).

   c. If a TIOC buffer is available, TSINPUT uses the services of QTIP to move data from the TCAM buffer to the TIOC buffer. Most line control characters and all 3270 buffer control characters are edited out of the message during this move.

   d. SRM is notified that the user is no longer in an input wait and may be swapped in.

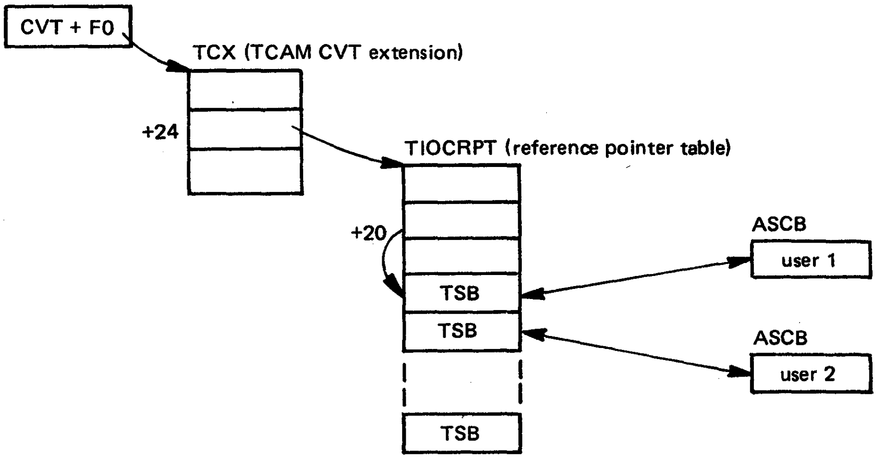
   e. The TCAM buffer is routed to the buffer disposition routine for final processing.

5. Once the TCAM buffer has been freed and final cleanup has been performed on the line, TCAM searches for additional work on the work-to-do queues. If there is none, TCAM enters a wait.

6. Once SRM has swapped-in the user, TGET regains control. Using QTIP, TGET moves the data from the TIOC buffer to the user's buffer.

TSO/TIOC Terminal I/O Diagnostic Techniques

For terminal hangs or interlocks involving TSO terminal I/O, a good place to start is at the TSB and TIOCRPT. The TSBs are physically contiguous and adjacent to the TIOCRPT (all in CSA), as shown below:



TIOCRPT is described in the *Debugging Handbook*. TSB is described in *Data Areas* (microfiche). TIOC is described in *OS/VS TCAM Level 10 Logic*.

TSBOWIP and TSBWOWIP are used to serialize TPUTs to a user. TSBOWIP is set at the start of a TPUT SVC, while that SVC holds the local and CMS locks.

If another TPUT is issued before OWIP is reset, then WOWIP is set and the issuer of the second TPUT is put in OWAIT.

The task that has "seized the TSB" (that is, set OWIP) can be determined by checking TSBCTCB. (TSBTJIP and TSBTJOW serve approximately the same function for cross-memory TPUTs.)

TSO Attention Processing

The following section summarizes the process of TSO attentions. The numbers in parentheses correlate to the numbers in Figure A-13.

TCAM Channel End Appendage (1)

- Ensures TCAM is active.
- Finds the element associated with this terminal.
- Places the element on the asynchronous queue.
- TCAM dispatcher merges the asynchronous queue to the ready queue and give control to the message handler.
- TCAM recognizes the following forms of terminal attention interrupts:
  - I/O attention interrupt for a 2741, which is checked in the line end appendage.
  - Two separate interrupts for the 3270; (1) a keyboard-invoked I/O attention interrupt, followed by (2) an I/O complete interrupt for the read issued by TCAM in response to the first interrupt.
  - A user character string for a simulated attention, which is checked by the SIMATTN routine.

Simulated Attention (2):

The message control program (MCP) reads input from the terminal the same as it does for normal operation. It then passes the message to the message handler.

Message Handler (MH) (3):

- Checks for the following conditions and calls TIOC if any exist:
  - Terminal input (character string)
  - PA1 function key
  - Terminal output lines

TIOC Attention Handler (4):

- Ensures TSO is active.
- Gets the user's TSB.
- Checks if the attention was caused by a deleted line.
- Invokes QTIP (TIOC/TSO interface).

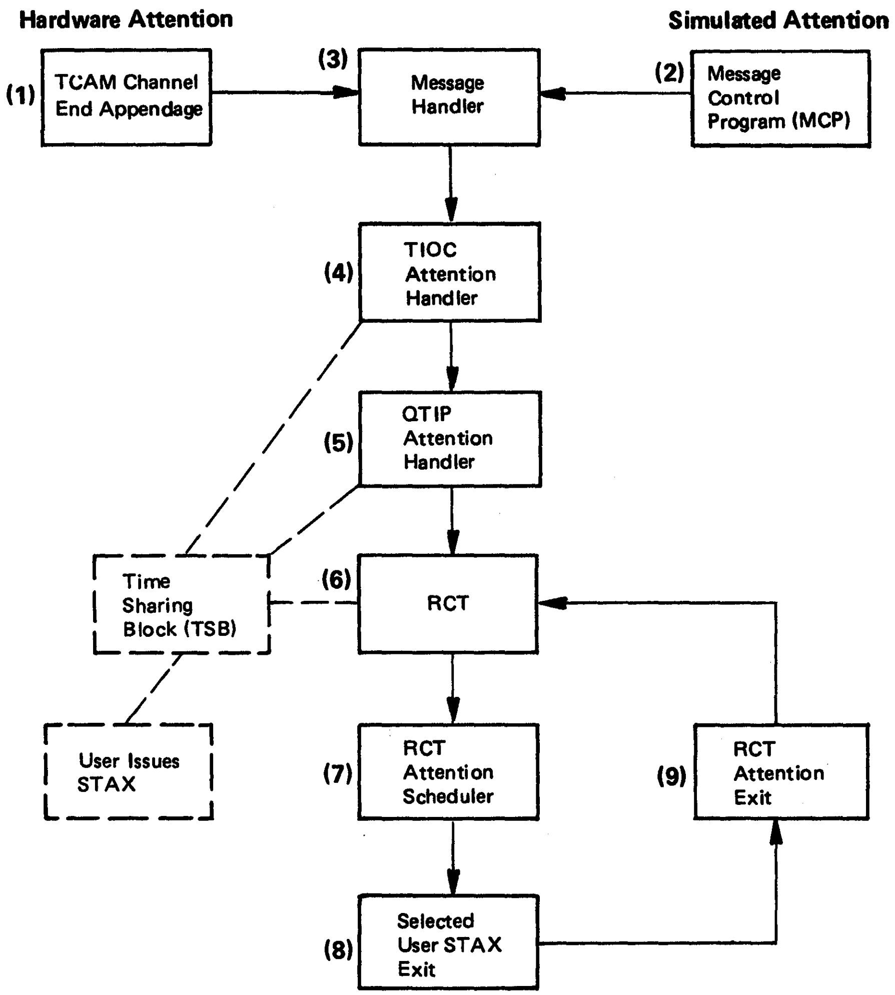

Hardware Attention
Simulated Attention
(1) TCAM Channel End Appendage
(3) Message Handler
(2) Message Control Program (MCP)
(4) TIOC Attention Handler
(5) QTIP Attention Handler
Time Sharing Block (TSB)
(6) RCT
User Issues STAX
(7) RCT Attention Scheduler
(9) RCT Attention Exit
(8) Selected User STAX Exit

Figure A-13. TSO Attention Flow

QTIP Attention Handler (5):

- Checks if the user has issued any STAX macros.
- Ensures the number of unprocessed attentions does not exceed the number of active STAXs (causes '!I' = TOO MANY ATTENTIONS or '!' = ATTENTION ACCEPTED to be printed at the terminal).
- Posts the RCT to schedule the user's attention exit.
- Purges input and output message queues to/from user except ASID type messages.

RCT (Region Task Control) (6):

- Waits for:
  - Termination
  - QUIESCE/RESTORE
  - Attention

RCT Attention Scheduler (7):

- Cancels previously-scheduled attentions that have not been executed.
- Determines the current attention level requested.
- Disables any affected tasks.
- If OBUF and/or IBUF was specified on the STAX macro, issues TPUT and/or TGET.

User STAX Exit (8):

- User defined.

RCT Attention Exit (9):

- Enables any affected tasks.
- Checks for another attention pending.
- RCT enters wait.

TSO APAR Documentation

TSO APAR documentation should include:

- Terminal input and output.
- SYSUDUMP or stand-alone dump, as appropriate.
- Information about how the system differs from PID release in the TSO area:
  - PTF list.
  - Information about non-IBM commands that appear in terminal output.
  - Description of any TMP modifications.
  - Description of applicable installation exits (LOGON, SUBMIT, etc.).
- Listing of the logon procedure, with a list of membernames in STEPLIBs, if any.

Appendix B. Stand-Alone Dump Analysis

This appendix contains a procedure that has been used successfully in stand-alone dump analysis. It is part of the course material in Field Engineering classes that teach MVS problem determination. This procedure does not attempt to cover all situations but it can be used as a guide through major status areas until you become thoroughly familiar with the system.

Overview

Stand-alone dumps are generally taken by the operator when:

- The system has stopped in a solid wait state with a wait state code.
- There appears to be a system loop.
- The system is not running or is running slowly.

Usually the 'Title From Dump' reflects what the operator thought happened.

Before becoming too involved in the problem itself, it is a good practice to get some feel for the status of the system at the time the dump was taken. Some valuable system status indicators can be obtained from the formatted section of the dump. Indicators can be obtained from the formatted portion of the dump under "System Summary" (produced by the SUMMARY control statement) and CSD, PSA, LCCA, and PCCA (produced by the CPUDATA control statement). Although it is seldom that any one indicator definitely points out the problem, when all indicators are noted and analyzed, a pattern might emerge that points the problem solver to the proper area for further investigation.

The enabled wait generally occurs as a result of the lack of some critical system resource. If the PRINT statement of PRDMP is used, PRDMP identifies the current task. If the current task is the wait task, the message "Current Task = Wait Task" might appear.

If it appears you have an enabled wait condition, read the chapter on "Waits" in Section 4 of this book before proceeding with your analysis.

The system can appear or actually prove to be bottlenecked because the operator cannot communicate with MVS. This is the sign of a problem almost anywhere in MVS, but an error in the communication task or its associated processing might be the direct cause. The communication task is identified by a X'FD' in the TCBTID field (TCB+X'EE'). By inspecting the RB structure associated with this task, you can determine the current status. It is not unusual to find one RB with a resume PSW address in the LPA and an RB wait count of one. If more than one RB is chained from the TCB and you could not enter commands, analyze the RB structure as this is not a normal condition.

Remember that communications task processing is very dependent on the rest of the operating system. Probably some external service or process has caused the communications task to back-up, and this possibility should be investigated.

For the system to continue execution, the major components must be operational. If any critical system components such as master scheduler, ASM, JES2, and TCAM for TSO, terminate abnormally and fail to recover, the system cannot continue normal operation. Usually this can be determined from the records in SYS1.LOGREC. However, check the TCB summary in the formatted section for completion codes.

The presence of a TCB completion code does not positively identify the associated task as being inoperative. It is possible that the completion code is residual and the task has recovered. The presence of a completion code makes the task suspect, however, and should be investigated.

Unless the operator STORE STATUS command was issued before taking the dump or the "Title from Dump" reflects a WSC (wait state code), it can be difficult to determine if a WSC exists and what it is if it does.

If however, the WSC PSW is dispatched by NIP during IPL, it is generally located in one of two places:

- In the MCH new PSW if a program check occurred prior to RTM initialization.
- In the nucleus vector table (NVT+X'E0') in the case of a system-detected error during the NIP process.

The other WSCs (they are few in number) issued by the system are dispatched by the master scheduler communications task and ASM. The current address space should identify who loaded the WSC PSW; WSC PSWs are issued when the system determines that it cannot continue. They are usually preceded by other error indicators that should be investigated along with the WSC.

Note: A valid WSC looks like: X'000A0000 00nnnxxx'

where: nnn is the reason code
xxx is the wait state code.

A disabled wait normally has a wait state code associated with it. If so, ,he messages and codes should contain a problem description.

If there is no wait state code, the trace table should indicate the last sequence of events leading to the wait state condition. Probably a bad PSW (wait bit on) has been loaded.

If no valid WSC exists and if the PSW reflects the wait bit, is disabled, and the STORE STATUS registers are not equal to zero, suspect: a user or Field Engineering trap or a SLIP trap (with a wait state code of X'01B'), a bad branch, or system damage. Examine the trace table and attempt to define the events that led up to the wait condition. Was the last entry an SRB dispatch or an SVC I/O interrupt? Using the PSW address, determine the entry point of the routine if possible.

The PSA is a system area whose status indicators are dynamically changing. The status indicators reflect the condition of the system the instant the dump was taken. Taken out of context, they can be misleading.

Therefore, find out if the operator entered a STORE STATUS command and keep in mind that the status could have been stored any time and not necessarily just before the dump.

Note: If you IPL the stand-alone dump program from the system control (SC) frame on the 3033, it is not necessary to perform the STORE STATUS operation. Status is automatically stored when stand-alone dump is invoked from the SC frame.

Note: The best evidence that the operator issued STORE STATUS is the content of 'Current Registers and PSW at Time of Dump.' This is because the stored status is put in the PSA +X'100' and the registers are put at PSA +X'160- 1FF', and SADMP program reads this area as the current PSW and registers and writes them to the dump data set. On a UP, the formatted current data will be the same as in the PSA. On an MP system, however, the SADMP program issues SIGP to another processor to store status. The STORE STATUS command always stores in the unprefixed PSA at location zero. This means that the unprefixed PSA will contain the registers and PSW from another processor. If the SADMP program did not save the STORE STATUS data before issuing the SIGP instruction to the other processor, the data from the operator's STORE STATUS command would be overlaid and the contents lost.

Also note that on an MP system there are three PSAs and the AMDPRDMP program formats all of them for you. The unprefixed PSA is used only during NIP (and SADMP). Always be sure you are looking at the right PSA when you are analyzing the PSA contents.

If the PSW+X'01' = xE or x2, the PSW is a wait PSW. If PSW+X'00' = X'04', the system was disabled. If PSW+X'00' = X'07', the system was enabled. Determine whether the PSW contains a WSC or an address. Then determine what key the PSW reflects. PSW+X'01' = X'xC' or X'xE' where the x = key, as follows:

0 =	supervisor
1 =	scheduler/JES2/JES3
5 =	IOS, data management, actual block processor, O/C/EOV
6 =	TCAM/VTAM
7 =	IMS
8 =	virtual problem program
9-F =	V=R problem program

In a SADMP on a UP, locations X'00' through X'18' are always overlaid by the IPL CCWs and PSW from the IPL of SADMP itself. These locations never contain valid data.

Loops can be either disabled or enabled. The best way of determining which has occurred is by examining the PSW, the loop recording option table, and the trace table.

Recorded addresses that fall within the SRM code are usually not indicative of a loop because this code is entered periodically as a result of a timer interrupt. This signifies, however, that the system does enable for interrupts and you can treat the error as an enabled loop.

Note: If the only addresses the operator furnished or the loop recording option recorded are in the timer or SRM code, check that it is not really an enabled wait condition.

The typical disabled loop is quite short, whereas the enabled loop covers a wide range of addresses. Be careful that the recorded addresses that may reflect a short loop are not a 'loop within a loop.' Scan the trace table and try to determine if a pattern of activity exists. Look for SIOs to the same device, SVCs from the same address, program checks occurring frequently for other than page faults, or any repetitive activity. If no pattern exists, try to correlate the last trace entry with what you already know about the loop (for example, I/O interrupts, a loop in an IOS or SRB dispatch, and a loop in the nucleus in some routine which is entered via an SRB).

The enabled loop usually reflects a wide range of addresses and can even span address spaces between a user and the system address spaces. An examination of the trace table usually shows some pattern of activity that is recognizable as a loop.

Be especially suspicious of a SVC 0D or SVC 0A for the same size area, SVC 33, SVC 4C, and SIOs to the same device with the same IOSB address in register 1.

Trace table entries with SVC 0D and/or SVC 33 in a stand-alone dump usually mean that some task is abending and the system is attempting to recover and purge the task from the system.

If any address within the loop points to the lock manager (module IEAVELK), the problem is probably caused by someone requesting an unavailable spin lock. On a UP, this is an invalid condition and always signifies an overlaid lockword. On an MP system, this signifies that another processor is holding the lock and failing to release it. There is a strong possibility that this indicates an overlaid lockword also. If not, the problem is on another processor. In either case, register 11 can point to the lockword requested and register 14 is the address of the requestor. Check the value in the lockword. Valid values are a fullword of zeros or three bytes of zeros and the CPUID in the fourth byte. Any other bit configuration causes the system to spin in a disabled loop and signifies an overlaid lockword. Register 12 always contains the bit mask to check the locks-held-table in the PSA.

If the lockword is overlaid, you must identify who overlaid it. It is possible that the lockword was overlaid in conjunction with some other problem.

This procedure is designed to aid the problem solver and to supplement the diagnostic procedures he has developed over the years. Its main purpose is to call attention to the new serviceability features within MVS and provide an index into the correct component analysis procedures in Section 5 of this manual. Once again, the component analysis procedures are there as hints and helps rather than to provide a structured approach to all problems.

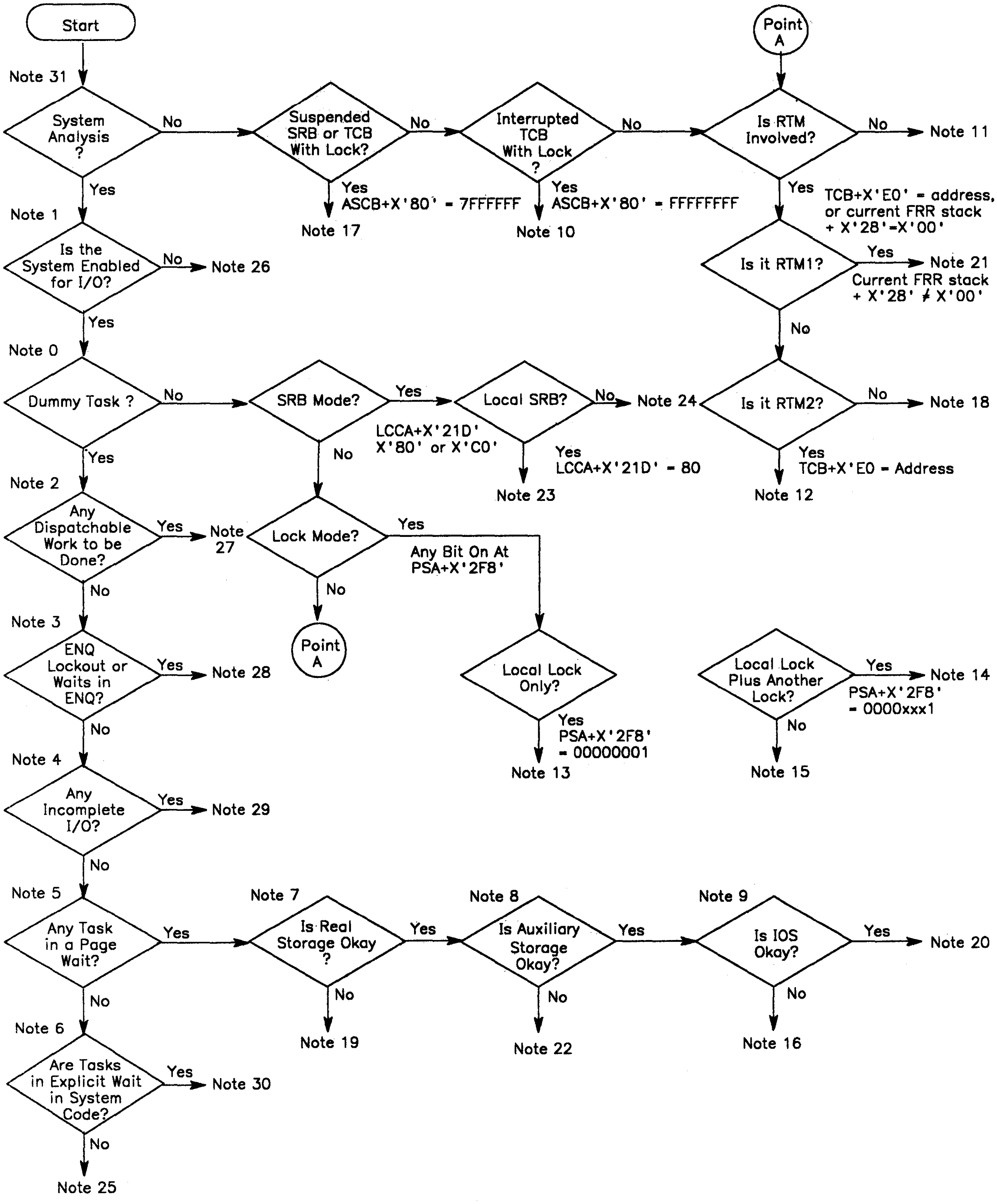


Figure B-1. Stand-alone Dump Analysis Flowchart

Analysis Procedure

The following explanations correlate to the “Notes” in Figure B-1.

Note 0 - Dummy Task?

The dummy wait has been dispatched on the processor if the following fields contain the values shown:

STORE STATUS PSW = 070E0000 00000000
STORE STATUS GPRs = 0
PSATNEW = PSATOLD = 0
PSAANEW points to ASID 0
PSAAOLD points to ASID 1
PSASUP1 = 04 (dispatcher)
PSAMODE = 08 (wait)

Note 1 - System Enabled for I/O?

Is bit 6 on in the current PSW?

Is control register 2 correctly loaded?

The current status of the system is in the PSA if a STORE STATUS command was entered before the dump was taken.

Note 2 - Dispatchable Work to be Done?

1. One of the first places to check for system dispatchability is the common system data area (CSD). For example, CSD+X‘C’=X‘40’ indicates that most of the system is non-dispatchable. This bit can be set by SDUMP. Is any address space abending and in the process of taking an SDUMP? Check the TCB summary for completion codes.

2. Dispatchable work within an address space is indicated by:

   ASCB+X‘80’ = X‘FFFFFFFF’ or X‘4FFFFFFF’
   ASCB+X‘80’ = X‘00000000’ (indicates the LOCAL lock is available)
   ASCB+X‘D0’ = SRB on the address space service management queue (ASCBLSMQ)
   ASCB+X‘D4’ = SRB on the address space service priority list (ASCBLSPL)
   ASCB+X‘D8’ ” 0 indicates ready TCBs not requiring the LOCAL lock.
   ASCB+X‘DC’ ” 0 indicates ready TCBs requiring the LOCAL lock.

3. The JES2/JES3 address space can contain work that should be passed to a waiting initiator or interface that has an address space for SYSIN or SYSOUT data.

4. Dispatchable work at the system level is indicated by SRBs queued to the global service manager queue (GSMQ) and the global service priority list (GSPL).

For (1), you must determine who set the bit on, who should have reset it and why the bit was set. It might be necessary to trap on the setting of this bit.

For (2), a X'7FFFFFFF' indicates that the holder of the local lock is suspended (for example, a page fault or CMS lock request). A CPUID value (such as X'00000040') indicates that the unit of work holding the local lock is currently running on that processor. Any nonzero value indicates that the lock is held and that TCBs requiring the LOCAL lock cannot be dispatched.

For (3), check the JES control blocks more closely.

For (4), determine why the dispatcher is not functioning. See the "Dispatcher" chapter in Section 5 of this manual.

Note 3 - Enqueue Lockout?

As in other systems, an exclusive enqueue prevents other tasks from using the same resource. However, in MVS, locks are now used frequently instead of an enqueue.

1. Use the QCB format function (QCBTRACE, Q, or GRSTRACE option of print dump) to print the QCBs and check for exclusive enqueues.

2. CVT+X'1B0' points to the GVT.
   GVT+X'10' points to the GVTX.
   GVTX+X'A4' points to the global queue hash table.
   GVTX+X'A8' points to the local queue hash table.
   GVTX+X'AC' points to the SYSID/ASID hash table.
   ASCB+X'110' points to the global QEL queue.
   ASCB+X'114' points to the local QEL queue.

   Note: The GVTX, the hash tables, and the QEL queues reside in the GRS address space.

3. Any QEL reflecting exclusive control prevents any other task from using that resource. Any QEL reflecting shared status prevents any task requesting exclusive control from using that resource.

4. The *Debugging Handbook* defines some of the major and minor ENQ names.

Note 4 - Incomplete I/O?

Label IECVSHDR in IEANUC01 points to a pool of cells used by IOS to build the IOQ (I/O queue element). The IOQs are found in two places:

1. An IOQ chained to the UCB-4 indicates an I/O operation is in progress or has completed on that device. The flag bytes at UCB+6 determine the current state of the device. The device is available when the flag byte is zero.

   No request for this device should be chained to the LCH during an enabled wait.

2. The IOQs are chained to the logical channel queues (LCH) if the I/O operation has been requested but not started.

The LCH is pointed to by the CVT+X'8C'. The entry for each logical channel is 20 bytes long. At X'00' into each entry is a pointer to the first IOQ queued for that logical channel. The presence of IOQs on any logical channel is immediately suspect when examining an enabled wait state dump. An empty queue (no requests) is indicated by a word of FFFFFFFF in the LCH at X'00'.

Note 5 - Is Any Task in a Page Wait?

Check the TCB RBs for a wait count not equal to zero.

RB+X'1C' = wait count
RB-8 " 40 (FLAG1)
TCBFLGS4 " '04'. This indicates a page-fault wait or a synchronous page fix wait.

Note 6 - Explicit Wait in System Code?

Does the address in the PSW fall within the nucleus or LPA code? Compare the address with a NUCMAP or LPA map.

Check the load list and CDEs for system modules that have been loaded into the private area.

Note 7 - Real Storage Okay?

If a task remains in a page wait, it could indicate a shortage of page frames or a real storage failure.

The control blocks that contain status about the use of real storage are:

1. Page vector table (PVT)

   PVT+X'04' = available frame count
   PVT+X'24' = free PCB count
   PVT+X'754' = deferred for lack of free page frames

2. Page frame table (PFT)

   Shows use of each frame of real storage available for paging.

Note 8 - Is Auxiliary Storage Okay?

If tasks are in a page wait and real storage is not a problem, the trouble could be within the auxiliary storage manager (ASM).

ASM status indicators are:

1. ASMVT+X'28' = the number of paging I/O requests received

2. ASMVT+X'2C' = the number of paging I/O requests completed

3. ASMVT+X'30' = the number of swap I/O requests received

4. ASMVT+X‘34’ = the number of swap I/O requests completed

5. ASMVT+X‘58’ = the SRB address used to redrive work through ASM

6. SRB+X‘1C’ = the address of an 8-byte parameter list that contains the addresses of the first and last I/O requests to be redriven through ASM

7. IORB+X‘03’ = indicates whether the I/O requests on the IORB have been passed to IOS.

If the number of I/O requests completed is equal to the number of I/O requests received, ASM has no outstanding work. If I/O requests have been started but not completed, determine what has happened to the I/O. If ASM's redrive SRB parameter list is nonzero, the SRB has been scheduled. Determine what the dispatcher has done with the SRB.

Note 9 - Is IOS Okay?

If all IORBs are idle (IORB+X‘03’, bit 0 is zero), then IOS has completely processed all the I/O that ASM has started.

Note 10 - Interrupted TCB?

The condition that caused the TCB holding the local lock or local lock and at least one cross memory services lock to be suspended has been resolved. The save area to be restored upon dispatching is the IHSA.

A TCB holding the local lock, or local lock and at least one cross memory services lock has been interrupted by a higher priority task. The save area used for redispatching is the IHSA. See the chapter “Dispatcher” in Section 5 or the chapter “System Execution Modes and Status Saving” in Section 2 of this manual.

Note 11 - Not RTM?

Without the detection of a failure by MVS, which would have caused entry into RTM, check the following. If the trace table in the stand-alone dump reflects the same task in most of its entries, this could be normal operation or the task could be in a loop. Check the following for status information:

LCCA
PSA
PCCA
Trace table
TCB
RB/SVRB

If no failure information is found (the system appears to be running normally), the problem might be that another task or address space should be running and is unable to. Check the following for status information:

1. Check each address space that is expected to be running to find out why it is not running. The information about each address space and task within that address space can be found in: ASCB, ASXB, TCB, and RB/SVRB.

2. Or, check the total system to find out why other work is not being run. Check the status of the system resources:

   ENQ lockout of data sets
   I/O failures
   RSM or ASM failure
   Waits in system code for other system resources (such as buffers)

If you are checking other than the current task, the TCBs could be dispatchable, but not yet dispatched. If the task is non-dispatchable (non-dispatchability bits on in the TCB), this can indicate an error situation. Or the task could be simply waiting (indicated by a wait count in the current RB). Check the dispatchability flags in the following control blocks for status of this task or select another address space or task and continue at Point A.

Status information can be found in: ASCB, ASXB, TCB, and RB/SVRB.

Note 12 - RTM2, Yes.

The most important place to find information about abend codes is *Message Library: System Codes.*

The RTM2 work area address is stored by RTM2 in TCB+X'E0'. Every system dump (SYSABEND/SYSMDUMP/SYSUDUMP) should have at least one TCB with an RTM2WA address at TCB+X'E0'. The error indicators contained in the RTM2WA are described in the *Debugging Handbook.*

If an ESTAE routine is in control when an error occurs, RTM builds an SDWA (described in the *Debugging Handbook)* and places its address at the RTM2WA+X'D4'.

Additional information about the failure may be found in the LOGREC buffer. RTM2WA+X'38' points to RTCT; RTCT+X'20' points to the LOGREC buffer.

If recursion occurs during RTM processing, other RTM2WAs may exist. If other work areas were obtained, the last one is pointed to by the TCB+X'E0'. The last RTM2WA points to the previous work area (RTM2WA+X'168,16C,170').

The RTM2WA is obtained from LSQA. It is therefore unique to each address space. If you are looking at a stand-alone dump, be sure that the area you are looking at belongs to the failing address space.

If the abending task is one of several abending tasks it is important to decide which task to look at first. There could be several failures or one failure causing all the others. Any failure in the system address spaces (JES2, master scheduler) are important because they might have caused the user address spaces to terminate.

For the system to continue execution, the major components must be operational. If any of the critical system components (master scheduler, ASM, JES2, TCAM for TSO, etc.) abend and fail to recover, the system cannot continue normal operation. Usually this can be determined from the records in LOGREC. However, check the TCB summary in the format section for completion codes.

The presence of a TCB completion code does not positively identify the associated task as being inoperational. It is possible that the completion code is residual and the task has recovered. The presence of a completion code makes the task suspect however, and should be investigated.

Simplify your choice of address spaces by using:

- SYS1.LOGREC external and internal entries
- Console sheets
- Trace table or GTF (check for SVC D or program check entries)

Once you have selected an address space and TCB, continue at Point A. (Check Section 5 for the component analysis of the involved component.)

In addition to the RTM2WA, status indicators related to the problem can be found in:

- Trace table
- ESTAE control block (SCB)
- RB/SVRB
- TCB

Note 13 - Local Lock Only?

The current ASCB+X'80' contains the CPU ID. The current TCB+X'F0' also contains the CPU ID. The loop is within this task. Status is saved (if a STORE STATUS was done) in:

- PSA
- LCCA
- Current stack
- Local SDWA (ASXB+X'6C') - if the task abended while holding the lock
- Trace table
- In-storage LOGREC buffer

Is this task looping in the lock manager's code? Check the PSA+X'228' and LCCA+X'20C'. If the task is looping and this is an MP system, another processor could be causing the loop by not freeing a spin lock that it is currently holding. The failure to free or obtain a lock can be caused by the lockword being overlaid on either an MP or UP.

If all processors of an MP are looping in lock manager code, then the failure could be in that code. If only one processor is in lock manager code, then the failure is likely to be in the processor currently holding the lock.

Where is the task looping? Why doesn't it free the locks? Is RTM involved with this task? If it is, continue at Point A.

See the chapters on "Locking" and "Effects of Multi-Processing on Problem Analysis" in Section 2 of this manual.

Note 14 - Local Lock Plus Another Lock.

The current ASCB+X'80' contains the CPU ID. The current TCB+X'F0' contains the CPU ID. The loop could be a spin loop waiting for the other processor to release a global lock. In this case, determine why the lock has not been released.

Status indicators can be found in the following areas (if a STORE STATUS was done):

- PSA
- LCCA
- Current stack
- Local SDWA (ASXB+X'6C') - if the task abended while holding the local lock and at least one cross memory services lock
- Trace table
- In-storage LOGREC buffer

See Note 13 for additional information. Also see the chapters "Locking" and "Effects of Multiprocessing on Problem Analysis" in Section 2 of this manual.

Note 15 - Global Lock Held.

A global lock loop in an MP system could be normal. The spin loop continues until the global lock is released by the other processor. Determine why the other processor has not released the lock.

Error status indicators can be found in the following areas if a STORE STATUS was done:

- PSA (current PSW)
- LCCA
- Current stack
- Global SDWA (if there was an abended failure while the global lock was held)

The global SDWA for the super stacks is located at the respective super stack+X'410'. For the normal stack, the global SDWA immediately follows the RESTART super stack SDWA+X'3F0'.

Now continue at Point A in the procedure. See Note 13 for additional information. Also see the chapters "Locking" and "Effects of Multiprocessing on Problem Analysis" in Section 2 of this manual.

Note 16 - IOS Not Okay.

Check the requests sent to IOS from auxiliary storage manager (ASM). Control blocks containing information are:

1. PART (paging activity reference table) - One entry per page data set. Each PART entry contains a pointer to an IORB (I/O request block) at X'1C' and a pointer to a UCB at X'2C'.

2. IORB contains the following I/O related data:

   IORB+X'1' = number of IORBs for this page data set
   IORB+X'3' = indicates whether IORB is in use
   IORB+X'4' = pointer to next IORB for this page data set
   IORB+X'8' = pointer to the first PCCW
   IORB+X'C' = pointer to the IOSB
   IORB+X'2C' = pointer to the last CCW in the channel program.

Refer to the Component Analysis section for additional IOS status indicators.

Note 17 - Suspended SRB or TCB With Lock Held.

An SRB can be suspended because of a page fault, a synchronous page fix, a request for a cross memory services lock when it is being held by another address space, or SMF suspension. The save area for the suspended SRB is the SSRB. If interrupted by a page fault, the SSRB is pointed to by the corresponding PCB+X'1C'.

For a general cross memory services lock request, the SSRB is on the requested CMS lock suspend queue, which can be located in the system lock area. (See the topic on locking to locate the system lock area.)

For an ENQ/DEQ cross memory services lock request, the SSRB is on the CMSEQDQ lock suspend queue (CMSEDSQH).

A locked TCB can be suspended for the same reasons as an SRB. The save area is the IHSA of the locally locked address space (described in the *Debugging Handbook*). The IHSA is valid during a page fault if the corresponding PCB+X'08' flag is on. The IHSA is valid for a cross memory services (CMS) lock suspension if the ASCB is on the CMS lock suspend queue (SQH) for that cross memory service lock requested (either the general CMS, the CMSEQDQ, or the CMSSMF).

Note 18 - Not RTM2.

The presence of a TCB completion code does not positively identify the associated task as being inoperational. It is possible that the completion code is residual and the task has recovered. The presence of a completion code makes the task suspect however, and it should be investigated.

The save areas have been released. The status of the error has been written to SYS1.LOGREC. Continue at Point A with other TCBs in the dump. Another abending task is likely. If this is a stand-alone dump, it very likely has the needed

LOGREC entry in the in-storage buffer. CVT+'23C' points to RTCT; RTCT+X'20' points to the LOGREC buffer.

Note 19 - Real Storage Not Okay.

If page waits seem to be caused by the lack of real frames, check their usage. The PFT contains information about each frame currently being used. Important items to check are:

Which ASID holds the most real storage?
What are the frames being used for?
Is it valid that they be held or is there a problem with the freeing of the frames?

Status information might be found in the PVT, PFT, and RSMHD and ASCB (X'98') for the ASID that is holding all the frames.

See the "RSM" chapter in Section 5 of this manual for more information about RSM.

Note 20 - IOS Okay.

Either something was missed along the way or the failure might be in one of the following areas:

- The IOS interrupt handler has failed to schedule the SRB/IOSB to the address space.

- The dispatcher has not handled the SRB correctly.

- POST has not functioned properly.

Information on these errors might be found in the trace table or the in-storage LOGREC buffers.

Note 21 - RTM1 Involved.

If there is an address at TCB+X'104' there might be two problems to resolve:

- The failure that caused the system to enter RTM initially.

- A loop between RTM1 and RTM2 since the pointer at TCB+X'104' normally lasts for only a short time.

  The pointer at TCB+X'104' is the EED (described in the *Debugging Handbook*). This data area is used to pass information from RTM1 to RTM2. Once RTM2 receives control the information is moved to the RTM2 work area and the EED is deleted. Therefore, because of its short duration, the presence of an EED is unusual.

A SLIP trap may be required to solve the RTM loop. This loop is of course the most important problem.

If the loop is in the current task, check these status indicators:

- LCCA
- PSA
- Current stack
- RTM1WA
- RTM2WA
- SDWA pointed to by RTM1WA
- EEDs
- LOGREC buffer
- Trace table

If the loop is not in the current task, all the indicators above except the LCCA, PSA, and current stack are valid. The current FRR stack is also a valid status indicator. Remember that all disabled or locally locked code runs under the protection of an FRR routine.

Check the current stack pointer at PSA+X'380'. If the current stack pointer points to a super FRR it is almost certain that system damage has occurred.

The normal stack at X'C00' contains a record of FRR activity for the current address space. Location X'C0C' is the pointer to the current entry on the normal FRR stack. An address at X'C0C' equal to the address at X'C00' indicates an empty stack. Any address at X'C0C' greater than the address at X'C00' indicates that the system is currently under FRR protection and the first word in each FRR entry is a pointer to the FRR routine. Because the FRR routine is usually embedded within the routine it protects, identifying the FRR routine identifies the "looper."

The second word in each entry contains an indicator in the first byte. A X'80' indicates that this routine is in control. A X'40' indicates that this nested recovery routine is in control. In a SADUMP, if an entry on the stack points to RTM or SVCDUMP's FRR, it is almost certain that system damage has occurred in a SADMP. This is normal in an SVC dump.

If the byte at X'C28' is not 0 and there is an address at either X'C5C' or X'C60', there has been an entry into RTM1 and an SDWA has been obtained. The loop could be occurring in the FRR routine itself. The first word in the FRR stack entry points to the FRR routine. The SDWA (pointed to by X'C5C' or X'C60') is the input passed to the FRR. Examine the code for the FRR and the module and consider the input passed to it in the SDWA to gain some insight into the cause of the loop.

Note 22 - Auxiliary Storage Not Okay.

If the count of I/O requests received (ASMVT+X'28' for paging requests, and ASMVT+X'30' for swapping requests) differs from the count of I/O requests completed (ASMVT+X'2C' for paging requests, and ASMVT+X'34' for swapping requests), and the IORBS are all idle (IORB+X'03', bit 0 is zero), locate those paging I/O requests (represented by an AIA) that ASM has received but not completed. Control blocks containing information are:

1. PARTE + X'3E' = count of outstanding I/O requests on a page data set

2. AIA-X'28' = part of PCB which contains RSM-related data

3. PART+X'24-28' = queue of AIAs waiting for PCCWs

4. ASMHD+X'0C' = swapout AIAs waiting to be processed

5. ASMHD+X'10' = queue of completed swap requests waiting to be returned to RSM

6. The following fields in the ASMVT point to SRBs to be processed:

   ASMVT+X'58' = the SRB whose parameter field (SRB+X'1C') points to a parameter list that points to I/O requests to be redriven through ASM

   ASMVT+X'5C' = the SRB whose parameter field (SRB+X'1C') has swap requests to be started

   ASMVT+X'60' = the SRB whose parameter field (SRB+X'1C') has error requests to be handled.

7. Each active IORB (PART entry+X'1C') contains a chain of PCCWs (IORB+X'8'). Each of these PCCWs that is active points to an AIA (PCCW+X'10').

8. If the AIA cannot be found by the above means (that is, it was lost by ASM), PCB/AIA may be found on the common I/O queue (PVT+X'75C-760') or one of the local I/O queues (RSMHD+X'1C-20').

For further information, see ASM's "General Debugging Approach" in Section 5.

Note 23 - Local SRB Mode.

This indicates a loop (or enabled wait) within a single address space.

The SRB code cannot be pre-empted. If a loop occurs in the SRB routine, no higher priority task can be dispatched.

For an MP system there is a second possibility. Determine if the loop is in the lock manager code. If so, see notes 13, 14, and 15 for additional information. Continue at Point A.

Status Indicators

- Trace table.

- PSA (current PSW).

- LCCA.

- Current stack.

- RTM1WA (SDWA) - if abend occurred during SRB processing.

- ASCB.

- RTM1WA+X‘38’ points to an SDWA obtained via GETMAIN (if RTM1WA+X‘40’ = 10).

- RTM1WA+X‘34’ points to a local SDWA if the GETMAIN for SDWA failed.

Note: If the system is an MP and the loop is in the lock manager code, then another processor might be at fault. See notes 13, 14, and 15 for additional information. Continue at Point A.

Status Indicators

- PSA (current PSW).

- LCCA.

- Current stack.

- RTM1WA (SDWA) - if failure occurred during SRB processing.

- Trace table.

- RTM1WA+X‘38’ points to an SDWA obtained via GETMAIN (if RTM1WA+X‘40’ = 10).

- RTM1WA+X‘34’ points to a local SDWA if the GETMAIN failed. See the chapter “Dispatcher” in Section 5. Also see the chapters “Locking,” “System Execution Modes and Status Saving,” and “Effects of MP on Problem Analysis” in Section 2 of this manual.

Note 24 - Global SRB Mode.

This indicates an enabled loop (or enabled wait) within a single address space.

The SRB code cannot be pre-empted. If a loop occurs in the SRB routine, no higher priority task can be dispatched.

For an MP system there is a second possibility. Determine if the loop is in the lock manager code. If so, see notes 13, 14, and 15 for additional information. Continue at Point A.

Status Indicators

- Trace table.

- PSA (current PSW).

- LCCA.

- Current stack.

- RTM1WA (SDWA) - if ABEND occurred during SRB processing.

- ASCB.

- RTM1WA+X'38' points to an SDWA obtained via GETMAIN (if RTM1WA+X'40' = 10).

- RTM1WA+X'34' points to a local SDWA if the GETMAIN failed.

Note: If this is an MP system and the loop is in the lock manager code, then another processor might be at fault. See notes 13, 14, and 15 for additional information. Continue at Point A.

Status Indicators

- PSA (current PSW).

- LCCA.

- Current stack.

- RTM1WA (SDWA) - if failure occurred during SRB processing.

- Trace table.

- RTM1WA+X'38' points to an SDWA obtained via GETMAIN (if RTM1WA+X'40' = 10).

- RTM1WA+X'34' points to a local SDWA if the GETMAIN failed. See the chapter "Dispatcher" in Section 5. Also see the chapters "Locking," "Systen Execution Modes and Status Saving," and "Effect of MP on Problem Analysis" in Section 2 of this manual.

Note 25 - Wait in User Code.

This could be normal operation for an explicit wait (SVC 1) issued by a user routine. Determine if the event waited upon has completed. Check the TCB non-dispatchability flags to determine the reason. The flags normally indicate th area of the problem. For example, if TCBFLGS4 = X'04', this indicates a VARY or QUIESCE command is in process on an MP system; TCBFLGS5 = X'80' means the task was terminated.

Note 26 - Non-enabled System.

A disabled wait normally has a wait state code associated with it. If so, the messages and codes should contain a problem description.

If there is no wait state code, the trace table should indicate the last sequence of events leading to the wait state condition. Probably a bad PSW (wait bit on) has been loaded.

Status Indicators

- LCCA
- PSA
- Current stack
- Trace table
- In-storage LOGREC buffer

If no valid WSC exists, if the PSW reflects the wait bit and is disabled, and if the STORE STATUS registers are not equal to zero, suspect a user/FE trap, bad branch, or system damage. Examine the trace table and attempt to define events that lead up to the wait condition. Was the last entry an SRB dispatch or an SVC or I/O interrupt? Using the PSW address, determine the entry point of the routine if possible and go to the chapter "MVS Trace Analysis" in Section 2 of this manual.

If the wait state occurs during system initialization, see the NIP vector table for error information. If the system is in a disable loop, determine what code is in control and why it is not returning to the enabled state.

A disabled loop in the lock manager on an MP system could be okay. Read notes 13, 14, and 15. A disabled loop in the SIGP processor on an MP system could be okay. (Another processor should turn off its PCCA's parallel/serial bit.)

If the system is looping (no wait bit), follow the SRB mode path. Check if RTM is involved and if it is, go to Point A.

*Note 27 - **Dispatchable Work Available.***

If the system is dispatchable and an address space has dispatchable work, the following are possible causes:

- The dispatcher is not functioning.

- CPU affinity may have been requested.

- JES2 might not be sending work to the initiators. In this case take a closer look at JES2.

See the chapter "Dispatcher" in Section 5 of this manual to determine why the dispatcher is not functioning properly.

*Note 28 - **Enqueue Lockout.***

Determine why the top task of a series of exclusive enqueues is not running or has not dequeued from the resource.

Note: It is valid for the top task to be swapped out. If it does not get swapped back in, then the failure might be in the system resource manager (SRM).

Note 29 - Incomplete I/O.

This is a probable hardware error. See the "IOS" chapter in Section 5 to determine the status of I/O.

Note 30 - Explicit Wait in System Code.

Check in the program listings (on microfiche) for the reason of the wait. Then determine which resource is being waited upon.

Once the resource is identified, determine if the wait should have been satisfied. If the wait appears to be a normal operation, continue at Point A for this TCB.

If the last thing done before the wait was an SVC 23 (WTO), related information can be found in the UCM base, prefix UCM, UCM extension and the chain of used WQEs.

Note 31 - System Analysis.

If the failing task or component is not known, continue on the "yes" path of the flowchart.

To determine status about a TCB without doing a total system analysis, continue on the "no" path of the flowchart.

For a complete system analysis, start with low storage. Check the PSA for a low storage overlay. Critical fields are the CVT pointer at X'10', the PSW new locations at location X'58-78' and at location X'00', and the trace table pointer at location X'54'. Be especially critical of the interrupt handler new PSWs. Any change to any new PSW will cause the next interrupt handler for that event to be dispatched in the wrong mode or key or to the wrong address. Subsequent results can be very unpredictable.

Keep in mind that the CVT pointer at location X'10' is constantly refreshed and the old PSWs are constantly updated by the hardware. They could have been overlaid at one time and still look okay in the dump from an MP system.

In a SADMP on a UP, locations X'00' through X'18' are always overlaid by the IPL CCWs and PSW from the IPL of SADMP itself. They will never contain valid data.

Other important fields in the PSA are as follows.

The interrupt code for the various classes of interrupts are located at:

- X'84' external interrupt
- X'88' SVC interrupt
- X'8C' program interrupt

These fields indicate the last type of interrupt associated with each interrupt class for each processor.

PSA+X'210' - address of the LCCA (1 per processor). The LCCA contains many of the status-saving areas that were located in low storage in previous systems. It is used for software environment saving and indicators. The registers associated with each of the interrupts you have discovered in the PSA are saved in this area. In addition, the system mode indicators for each processor are maintained in the LCCA.

The ASCB and TCB NEW/OLD pointers in the PSA (locations X'218-227') indicate the currently dispatched task. *Note:* PSATOLD can equal zero if an SRB is dispatched.

PSA+X'228' - PSASUPER. This is a field of bits that represent various supervisory functions in the system. If a loop is suspected, check these fields to isolate the looping process. Note that the dispatcher's super bit (X'04') will be left on when the wait task is dispatched.

PSA+X'2F8' - PSACLHS. This field indicates the current locks held on each processor. Knowing which locks are held may help isolate the problem, especially in a loop situation. By determining the lock holders you can isolate the current process.

PSA+X'380' - PSACSTK. This is the address of the active recovery stack that contains the addresses of the recovery routines to which control will be routed in case of an error. If the address is other than X'C00' (normal stack), determining the type of stack (for example, program check FLIH, restart FLIH) should aid in debugging the loop situation. Note that the external FLIH runs under the super stack (current stack = SSTK).

Another thing to consider in systems analysis is the possibility of a storage overlay of some critical system code such as IOS or GETMAIN.

Because of the recovery aspects of MVS (percolation and retry), evidence of storage overlays can often be found in the LOGREC recording buffers.

To find the LOGREC recording buffers:

- CVT+X'23C' = pointer to the recovery termination control table. (RTCT).
- The RTCT+X'20' = pointer to the recording buffers.
- The recording buffer (LRB)+0 = pointer to the start of the area.
- The recording buffer (LRB)+4 = pointer to the end of the area.
- The recording buffer (LRB)+8 = pointer to the next available buffer.

Each buffer entry for a software record begins with X'408x' or '428x' where x = the release number. Each software entry is approximately X'200' bytes long. The first X'20' bytes is header information and contains the CPUID and serial, the time and date, and the JOBNAME if entry is made from an ESTAE routine. This is followed by an SDWA as defined in the *Debugging Handbook*.

Identify the last entry. Are there entries following it? If so, the buffer might have been wrapped and it no longer contains the earliest entry. It is a good idea to have the SYS1.LOGREC records for the time leading up to the dump. Scan the trace table for SVC 4C. This represents a call to the LOGREC recording task and identifies a record being written to SYS1.LOGREC. If SVC 4Cs appear in the trace, it is certain that there are SYS1.LOGREC records that may more closely define the problem. (See the discussion of LOGREC records in the chapter "Use of Recovery Work Areas for Problem Analysis" in Section 2 of this manual.)

As a general approach, follow the flow of FRR activity from the last entry backwards until a pattern is recognizable or the first entry is found.

If the abend codes relate to a particular component, refer to that component's analysis procedure in Section 5 of this manual.

If you can define a function that is consistently failing (IOS, a program check, etc.), examine the trace table for evidence of successful completion of this function. If the function completed successfully, the search for the function that caused the overlay is narrowed to those functions appearing in the trace between the last successful completion and the first evidence of error. This should at least narrow the search to the address space and task level.

Analyze the contents of the overlaid storage. If it appears to contain registers, determine what data areas or modules the registers are pointing at. This helps to identify the failing code.

If there is no evidence of a storage overlay, return to your system analysis at the beginning of Note 31.

If a storage overlay exists, further examination of the reported problem is usually non-productive until the cause of the system damage is explained.

It might be necessary to build a trap to identify the cause of the overlay. The chapter "Additional Data Gathering Techniques" in Section 2 of this manual helps in building such a trap.

Appendix C. SVC DUMP Title Directory

This directory lists the titles of SVC dumps produced by MVS components via the SDUMP macro instruction. It also provides a module name to dump title cross-reference.

The directory has the following topics:

- **System-Defined SVC Dump Titles** - lists, in alphameric order, the titles of SVC dumps and provides diagnostic information for the modules that initiate the SVC dump via the SDUMP macro.
- **Operator- and Caller-Defined SVC Dump Titles** - provides diagnostic information for the modules that initiate SVC dumps via the SDUMP macro but where the dump title is defined by the system operator or the caller of SVC dump.
- **SVC Dumps Without Titles** - provides diagnostic information for the modules that initiate SVC dumps but where no titles are supplied on the SDUMP macro.
- **Module to SVC Dump Title Cross-Reference** - lists, in alphameric order, the MVS modules that issue the SDUMP macro and provides the titles of the SVC dumps specified by the modules on the SDUMP macro.

System-Defined SVC Dump Titles

This topic lists, in alphameric order, the titles of SVC dumps and provides diagnostic information for the modules that initiate the SVC dump via the SDUMP macro.

ABDUMP ERROR

Component: RTM - ABDUMP (5752-SC1CM)

Issuing Module: IEAVTABD

PLM: *OS/VS2 System Logic Library*

Explanation: An error has occurred during RTM processing of a SYSABEND, SYSMDUMP, or SYSUDUMP dump. The error occurred when (1) ABDUMP attempted to set up dump processing, or (2) SNAP or SDUMP processing encountered an error while taking the dump. The areas dumped are RGN, LPA, and LSQA.

Problem Determination: A software record is written to SYS1.LOGREC. The failing CSECT name and the error condition can be determined from RTM2WA and SDWA (in the LOGREC record).

ABEND IN IEAVTGLB

Component: RTM - PER Activation/Deactivation (5752-SC1CM)

Issuing Module: IEAVTGLB

PLM: *OS/VS2 System Logic Library*

Explanation: An error has occurred then the SLIP processor attempted to activate or deactivate PER in the system. Message IEA415I is also issued. The areas dumped are PSA and SQA. The summary part of the dump requested by IEAVTGLB contains information relevant to the error.

Problem Determination: A software record is written to SYS1.LOGREC. Also refer to message IEA415I in *VS2 System Messages.*

ABEND IN IEAVTJBN

Component: RTM - PER Activation/Deactivation (5752-SC1CM)

Issuing Module: IEAVTJBN

PLM: *OS/VS2 System Logic Library*

Explanation: An error has occurred when the SLIP processor attempted to determine if PER should be active for a new address space, started task, logon, mount, or job. Message IEA422I is also issued. The areas dumped are PSA and SQA. The summary part of the dump requested by IEAVTJBN contains information relevant to the error.

Problem Determination: A software record is written to SYS1.LOGREC. Also refer to message IEA422I in *System Messages.*

ABEND IN IEAVTLCL

Component: RTM - PER Activation/Deactivation (5752-SC1CM)

Issuing Module: IEAVTLCL

PLM: *OS/VS2 System Logic Library*

Explanation: An error has occurred when the SLIP processor was attempting to activate or deactivate PER in an address space. Message IEA415I is also issued. The areas dumped are PSA, SQA, and LSQA. The summary part of the dump requested by IEAVTLCL contains information relevant to the error.

Problem Determination: A software record is written to SYS1.LOGREC. Also refer to message IEA415I in *System Messages.*

ABEND IN SMF INTERVAL PROCESSING - ROUTINE IEEMB836
JOBNAME = xxxxxxxx

Component: SMF (5752-SC102)

Issuing Module: IEEMB836 - FRR

PLM: *OS/VS2 System Logic Library*

Explanation: An abend has occurred during SMF interval processing. xxxxxxxx indicates the name of the affected job.

Problem Determination: A software record is written to SYS1.LOGREC. The SDWACSCT field in the SDWA contains the name of the module in control at the time of the error.

ABEND code AT hhhhhhhh (nnnnnn) + X'nnnn' cc- - -cc

Component: JES2 (5752-SC1BH)

Issuing Module: HASPTERM or HASPRAS

PLM: *JES2 Logic.*

Explanation: An abend has occurred during JES2 processing. Fields in the dump title are:

code	abend code
hhhhhhhh	failing module name
nnnnnn	entry point address
X'nnnn'	offset to the failing instruction
cc- - -cc	brief description of the abend code and the JES2 or JES2 NJE release level

Abend codes that start with S are system codes, and those that start with $ are JES2 codes. The areas dumped are PSA, NUC, RGN, TRT, SQA, CSA, LPA, and SWA.

Problem Determination: For information on system abend codes, refer to *System Codes*; for JES2 abend codes, refer to message $HASP095 in *JES2 Messages.*

ABEND=aaa,COMPON=ALLOC,COMPID=SC1B4,ERRMOD=xxxxxxxx, ERRCSECT=yyyyyyyy,LVL=zzzzzzzz,ISSUER=IEFAB4E6

Component: Allocation (5752-SC1B4)

Issuing Module: IEFAB4E6 - Recovery routine

PLM: *OS/VS2 System Logic Library*

Explanation: An error has occurred during allocation processing. The areas dumped are NUC, LPA, ALLPSA, SQA, TRT, and RGN. If the error occurred during processing related to the allocation address space (ALLOCAS), message IEF100I is issued, the allocation address space might be terminated, and allocation processing continues. For other errors, all units allocated to the failing address space are unallocated and the job is abnormally terminated. The fields in the dump title are:

aaa	system completion code
xxxxxxxx	name of the failing load module
yyyyyyyy	name of the failing CSECT
zzzzzzz	PTF or product level number of the failing CSECT (such as JBB1226)

Problem Determination: A software record is written to SYS1.LOGREC. If the recovery routine was entered due to system completion code 05C, register 0 contains a reason code. See *System Codes* for an explanation of system code 05C and reason codes. If the recovery routine was entered due to an error related to allocation address space processing, message IEF100I is also issued. See *System Messages* for an explanation of message IEF100I.

ABEND=aaa,COMPON=CONVERTER, COMPID=SC1B9,ISSUER=IEFNB9CR

Component: Converter (5752-SC1B9)

Issuing Module: IEFNB9CR - Converter Recovery Routine

PLM: None (refer to microfiche)

Explanation: IEFNB9CR was entered due to an expected error (0B0 abend or program check) during converter processing. The areas dumped are LSQA, RGN, LPA, and SWA.

Problem Determination: A software record is written to SYS1.LOGREC.

ABEND=aaa,COMPON=INTERPRETER, COMPID=SC1B9,ISSUER=IEFNB9CR

Component: Interpreter (5752-SC1B9)

Issuing Module: IEFNB9IR - Interpreter Recovery Routine

PLM: None (refer to microfiche)

Explanation: IEFNB9IR was entered due to an expected error (0B0 abend or program check) during interpreter processing. The areas dumped are LSQA, RGN, LPA, and SWA.

Problem Determination: A software record is written to SYS1.LOGREC.

ABEND=aaa,COMPON=PC/AUTH-PCLINK UNSTACK,COMPID=SCXMS, ISSUER=IEAVXSTK

Component: PC/AUTH Services (5752-SCXMS)

Issuing Module: IEAVXSTK

PLM: *OS/VS2 System Logic Library*

Explanation: An error has occurred while the PCLINK service routine (IEAVXSTK) was processing an UNSTACK request. The areas dumped are NUC, LSQA, SQA, and SUM.

Problem Determination: A software record is written to SYS1.LOGREC. The SDWAVRA includes the following key-length data formatted entries:

- The 24-byte FRR parameter area
- The caller's return address
- The current TCB address (if not task mode, 0)
- The PCLINK work areas in the LCCA (56 bytes beginning at LCCASREG)
- The PSASTKE and PSAKPSW fields

ABEND=aaa,REASON=xxyy,GPRrr=zzzzzzzz,COMPON=PC/AUTH-macro, COMPID=SCXMS,ISSUER=IEAVXPCR

Component Program Call/Authorization (PC/AUTH) Services

Issuing Module IEAVXPCR - PC/AUTH Services FRR

PLM: *OS/VS2 System Logic Library*

Explanation: An error has occurred while a PC/AUTH service routine was processing. IEAVXPCR issues the SDUMP macro. The areas dumped are ALLPSA, NUC, SQA, SUMDUMP, RGN, and LSQA. Also, subpools 255

(LSQA) and 229 (pageable private area) of the PC/AUTH address space are synchronously dumped via SUMLSTA. Fields in the title are:

aaa	system completion code
xxyy	low-order two bytes of register 15 at the time of the error
rr	general purpose register
zzzzzzzz	contents of general purpose register rr
macro	name of the macro that invoked the PC/AUTH service

Notes:

1. *If the system completion code is 053, refer to VS2 System Codes for a description of the reason code xxyy and the diagnostic register value. For other system completion codes, the contents of register 14 at the time of the error are shown as GPR14=zzzzzzzz.*

2. *If register contents are not available at the time of error, then the title text contains N/A or UNAVAIL.*

3. *If the FRR is entered with an invalid service-in-control code value in PCRASERV, the title will contain 'PCRAEERC=nnnn' where nnnn is the PCRAEERC field at entry to the FRR.*

Problem Determination: A software record is written to SYS1.LOGREC and includes key-length-data formatted entries in SDWAVRA containing:

- The program call recovery area (PCRA) at entry to the FRR.
- Diagnostic information, if any, from the control block validation and/or queue verification routines.
- The service routine recovery area (SRRA). (As much as possible is included in the SDWAVRA.)

It is suggested that you initially print the dump using the PRDMP options LOGDATA, SUMMARY, SUMDUMP, and PRINT CURRENT to obtain sufficient diagnostic data.

ABP:IDA121A2 - ABEND FROM ABP FRR

Component: Block Processor (5752-SC1DA)

Issuing Module: IDA121A2 - FRR

PLM: *OS/VS2 VSAM Logic*

Explanation: An abnormal termination has occurred during VSAM block processing. The FRR routine in IDA121A2 issues the SDUMP macro. The areas dumped are PSA, NUC, RGN, TRT, CSA, and SQA.

Problem Determination: A VSAM request was being processed in the actual block processor (ABP), initiating I/O, when the error occurred. Register 3 points to the IOMB for the request. For information on how the IOMB relates to the VSAM control block structure, refer to *OS/VS2 VSAM Logic*.

ABP:IDA121A3 - ABEND FROM NORMAL END FRR

Component: Block Processor (5752-SC1DA)

Issuing Module: IDA121A3 - FRR

PLM: *OS/VS2 VSAM Logic*

Explanation: An abnormal termination has occurred while IDA121A3 was processing a VSAM request. RTM passes control to the FRR in IDA121A3 (at entry point IDA121F3), which issues the SDUMP macro. The areas dumped are PSA, NUC, RGN, TRT, CSA, and SQA.

Problem Determination: I/O for a VSAM request had completed normally when the error occurred. Register 3 points to the IOMB for the request. For information on how the IOMB relates to the VSAM control block structure, refer to the *OS/VS2 VSAM Logic*.

ABP:IDA121A4 - ABEND FROM ABNORMAL END FRR

Component: Block Processor (5752-SC1DA)

Issuing Module: IDA121A4 - FRR

PLM: *OS/VS2 VSAM Logic*

Explanation: An abnormal termination has occurred while IDA121A4 was processing a VSAM request. RTM passes control to the FRR in IDA121A4 (at entry point IDA121F4), which issues the SDUMP macro. The areas dumped are PSA, NUC, RGN, TRT, CSA, and SQA.

Problem Determination: I/O for a VSAM request had completed abnormally when the error occurred. Register 3 points to the IOMB for the request. For information on how the IOMB relates to the VSAM control block structure, refer to *OS/VS2 VSAM Logic*.

ABP:IGC121 - ABEND FROM SIOD FRR

Component: Block Processor (5752-SC1DA)

Issuing Module: IGC121 - FRR

PLM: *OS/VS2 VSAM Logic*

Explanation: An abnormal termination has occurred while IGC121 was processing a VSAM request. RTM passes control to the FRR in IDA121 (at entry point IDA121F1), which issues the SDUMP macro. The areas dumped are PSA, NUC, RGN, TRT, CSA, and SQA.

Problem Determination: The I/O manager was processing a VSAM request when the error occurred. Register 3 points to the IOMB for the request. For information on how the IOMB relates to the VSAM control block structure, refer to *OS/VS2 VSAM Logic*.

AHL007I GTF TERMINATING ON ERROR CONDITION

Component: GTF (5752-SC111)

Issuing Module: AHLGTFI

PLM: *OS/VS2 MVS Service Aids Logic*

Explanation: An error has occurred during GTF initialization. The ESTAE routine AHLIESTA in AHLGTFI requests a retry action which issues the SDUMP macro, writes message AHL016I, and frees storage and other resources that were allocated to GTF. GTF terminates itself. The areas dumped are RGN, SQA, and MCHEAD control block.

Problem Determination: A software record is written to SYS1.LOGREC. All control blocks allocated to GTF are dumped.

COMMON AUTHORIZATION CHECK ROUTINE ERROR,
ABEND=xxx,COMPON=SCHR-CMF,COMPID=BB131,
ISSUER=IEFCMAUT

Component: Scheduler (5752-SC1B6)

Issuing Module: IEFCMAUT

PLM: *System Logic Library*

Explanation: An abend has occurred during authorization checking. ESTAE routine SETESTAE in IEFCMAUT sets up the recovery environment. If no previous abend has occurred, recovery routine RECOVERY in IEFCMAUT requests a retry. If there was a previous abend, the recovery routine issues a SETRP to indicate that RTM should percolate the error to the next level of recovery.

Problem Determination: If an SDWA was obtained, a software record is written to SYS1.LOGREC which includes:

SDWAMODN	- IEFCMAUT (load module in error)
SDWACSCT	- IEFCMAUT (CSECT in error)
SDWAREXN	- IEFCMAUT (CSECT containing recovery routine)
SDWARRL	- RECOVERY (recovery routine)
SDWACID	- SC1B6 (component identifier)
SDWARCDE	- return code
SDWAMLVL	- product level

COMPID=SC1B8, xxx ABEND IN MASTER TR modname

Component: Master SCheduler Commands (5752-SC1B8)

Issuing Module: IEEMB816

PLM: *OS/VS2 System Logic Library*

Explanation: An abend has occurred during (1) processing of the TRACE command, or (2) execution of the IEETRACE macro, xxx indicates the abend code. Modname indicates the module in control at the time of the error and is one of the following:

IEEMB808 - the error occurred while adding an entry to the master trace function during system initialization or in response to a TRACE command.

IEEMB809 - the error occurred while activating or deactivating the master trace function during system initialization or in response to a TRACE command.

IEEMB816 - the abend occurred while processing some other error in the master trace facility.

UNKNOWN - the recovery routine could not determine the module that was in control at the time of the error.

The areas dumped are SUMDUMP, TRT, FRR work area, FRR parameter area, UCM extension, master trace caller's parameter list, and load module IEEMB808 with its dynamically acquired storage. Message IEE480I or IEE581I is also issued.

Problem Determination: A software record is written to SYS1.LOGREC. Refer to message IEE480I or IEE481I in *VS2 System Messages,* and to the appropriate code (indicated by xxx in the title) in *VS2 System Codes.*

COMPON=COMMTASK,COMPID=SC1CK,ISSUER=IEAVMFRR=FRR or ESTAE, COMMUNICATIONS TASK DUMP

Component: Communications Task (5752-SC1CK)

Issuing Module: IEAVMFRR

PLM: *OS/VS2 System Logic Library*

Explanation: IEAVMFRR is entered when an error has occurred during processing of a communications task function. The areas dumped are SQA, NUC, LSQA, LPA, SWA, CSA, ALLPSA, RGN, SUM, and TRT. The SUMLSTA contains the UCM, RDCMs, SACBs, and TDCMs.

Problem Determination: A software record is written to SYS1.LOGREC.

COMPON=COMMTASK,COMPID=SC1CK,ISSUER=IEAVSTAA, FAILURE IN COMMUNICATIONS TASK

Component: Communication Task (5752-SC1CK)

Issuing Module: IEAVSTAA

PLM: *OS/VS2 System Logic Library*

Explanation: IEAVSTAA is entered when both (1) an error has occurred during communications task processing, and (2) after unsuccessful recovery processing by ESTAE or FRR routines in the communications task. The areas dumped are NUC, LSQA, RGN, LPA, CSA, SWA, ALLPSA, SQA, and TRT.

Problem Determination: A software record is written to SYS1.LOGREC.

COMPON = COMMTASK,COMPID = SC1CK,ISSUER = IEAVN700,FAILURE IN COMM TASK ADDRESS SPACE CREATE ROUTINE

Component: Communications Task (5752-SC1CK)

Issuing Module: IEAVN700

PLM: *OS/VS2 System Initialization Logic*

Explanation: An error has occurred while IEAVN700 was creating the communications task address space. The areas dumped are ALLPSA, RGN, LSQA, SQA, and SUMDUMP. SUMDUMP contains the trace table, registers, and storage near the register values at the time of the error.

Problem Determination: A software record is written to SYS1.LOGREC.

COMPON = COMMTASK,COMPID = SC1CK,ISSUER = IEAVN701, FAILURE IN COMM TASK ADDRESS SPACE INITIALIZATION

Component Communications Task (5752-SC1CK)

Issuing Module: IEAVN701

PLM: *OS/VS2 System Initialization Logic*

Explanation: An error has occurred while IEAVN701 was initializing the communications task address space. The areas dumped are ALLPSA, NUC, RGN, LSQA, SQA, CSA, TRT, and SUMDUMP. SUMDUMP contains the trace table, registers, and storage near the register values at the time of the error.

Problem Determination: A software record is written to SYS1.LOGREC.

COMPON = GRS-COMMANDS,COMPID = SCSDS,ISSUER = ISGCRET0, POST OF GVTCECB FAILED

Component: Global Resource Serialization (5752-SCSDS)

Issuing Module: ISGCRET0

PLM: *OS/VS2 MVS Global Resource Serialization Logic*

Explanation: An error has occurred while one of the following modules was attempting to cross-memory post the command ECB being used by ISGCMDR.

ISGBSR	- RSA send/receive routine
ISGCMDR	- Command router
ISGGFRR0	- FRR for ENQ/DEQ;RESERVE
ISGGTRM1	- ENQ/DEQ/RESERVE termination resource manager

ISGCMDR was waiting for a command request or a message request. The areas dumped are PSA, SQA, and LSQA of the global resoruce serialization address space, and the GVT.

Problem Determination: Either the ECB address provided on the cross-memory post is in error, or the RB address in the ECB is in error.

COMPON=GRS-COMMANDS,COMPID=SCSDS,ISSUER=ISGCRET1, POST OF ECB OF COMMAND REQUESTOR FAILED

Component: Global Resource Serialization (5752-SCSDS)

Issuing Module: ISGCRET1

PLM: *OS/VS2 MVS Global Resource Serialization Logic*

Explanation: An error has occurred while ISGCMDR (command router) was attempting to cross-memory post the ECB that was being used by a command requestor to wait for a command request to be processed by ISGCMDR. The areas dumped are PSA, SQA, and LSQA of the command requestor's address space, and the commmand requestor's ECB.

Problem Determination: Either the ECB address provided on the cross-memory post is in error, or the RB address in the ECB is in error.

COMPON=GRS-CTC-DRIVER,COMPID=SCSDS,ISSUER=ISGJRCV

Component: Global Resource Serialization (5752-SCSDS)

Issuing Module: ISGJRCV

PLM: *OS/VS2 MVS Global Resource Serialization Logic*

Explanation: An error has occurred while ISGJDI (CTC driver DIE) was processing. The FRR ISGJRCV (for ISGJDI) uses the branch entry to SVC dump. If the GCL is valid, a summary dump is requested that contains the GCV.

Problem Determination: A software record is written to SYS1.LOGREC. It includes the failing CSECT name plus the following in the variable recording area (SDWAVRA) of the SDWA:

- If the IOSB is valid:
  - UCBNAME field of the UCB.
  - IOSB identifier.
  - IOSB address.
  - IOSB Fields: IOSFLA, IOSFLB, IOSFLC, IOSCOD, IOSUCB, IOSCSW, IOSUSE, IOSSNS.
- If the GCQ is valid:
  - GCQ address.
  - GCQDISEC field.

- If the GCL is valid:

  - GCL address.
  - GCL (without IOSBs)

- If the IOSB, GCQ, and/or GCL is in error, then the IOSB, GCQ, and/or GCL address is recorded.

COMPON=GRS-CTC DRIVER ENF EXITS,COMPID=SCSDS,ISSUER=ISGJENF0

Component: Global Resource Serialization (5752-SCSDS)

Issuing Module: ISGJENF0 - ESTAE

PLM: *OS/VS2 MVS Global Resource Serialization Logic*

Explanation: An error has occurred while ISGJENF0 (event notification facility exits routine) was processing. The ESTAE routine ISGJENFR (in ISGJENF0) issues the SDUMP macro.

Problem Determination: A software record is written to SYS1.LOGREC. It includes the ESTAE parameter list and the VARY parameter list in the variable recording area (SDWAVRA) of the SDWA.

COMPON=GRS-QUEUE SCANNING SERVICES,COMPID=SCSDS, ISSUER=ISGQSCNR

Component: Global Resource Serialization (5752-SCSDS)

Issuing Module: ISGQSCNR - FRR

PLM: *OS/VS2 MVS Global Resource Serialization Logic*

Explanation: An error has occurred while ISGQSCAN (queue scanning services) was processing. The FRR routine ISGQSCNR issues the SDUMP macro.

Problem Determination: A software record is written to SYS1.LOGREC.

COMPON=GRS-RING-PROCESSING,COMPID=SCSDS, ISSUER=ISGBERCV

Component: Global Resource Serialization (5752-SCSDS)

Issuing Module: ISGBERCV - ESTAE

PLM: *OS/VS2 MVS Global Resource Serialization Logic*

Explanation: An error has occurred while ISGBTC (ring processing task mode routine) or ISGBCI (ring processing command interface routine) was processing. ESTAE routine ISGBERCV issues the SDUMP macro. If the basic control blocks are valid, a summary dump is requested that includes the GVT, SQA (obtained by ISGBTC), and the private area (obtained by

(ISGBTC) for ring processing. An asynchronous dump of the current address space is always included in the dump request.

Problem Determination: A software record is written to SYS1.LOGREC. It includes the failing CSECT name plus the following in the variable recording area (SDWAVRA) of the SDWA:

- Address of ISGREPL (input parameter list to ISGBERCV).
- The ISGREPL.
- Address of ISGRSC (input parameter list to ISGBCI).
- The ISGRSC if ISGBTC failed.

COMPON=GRS-RING-PROC,COMPID=SCSDS,ISSUER=ISGBFRCV

Component: Global Resource Serialization (5752-SCSDS)

Issuing Module: ISGBFRCV - FRR

PLM: *OS/VS2 MVS Global Resource Serialization Logic*

Explanation: An error has occurred while ISGBSR (RSA send/receive routine) was processing. The FRR ISGBFRCV uses the branch entry to SVC dump. If the basic control blocks are valid, a summary dump is requested that includes the GVT, SQA (obtained by ISGBTC), and the private area (obtained by ISGBTC) for ring processing. An asynchronous dump of the current address space is always included in the dump request.

Problem Determination: A software record is written to SYS1.LOGREC. It includes the failing CSECT name plus the following in the variable recording area SDWAVRA) of the SDWA:

- RVRPARM address.

- The RVRPARM (input parameter area to ISGBFRCV).

- RSV ID and address.

- The following fields in the RSV control block:

  RSVCRSAT
  RSVRSLR
  RSVRSLS
  RSVRSLRF
  RSVRSLSF
  RSVTRSL
  RSVERR
  RSVWLOCK
  RSVRSASC
  RSVCPHNO
  RSVIBFOR (input buffer)
  RSAMRPFX (input buffer)
  RSVOBFOR (output buffer)
  RSAMRPFX (output buffer)

- RSL ID and address.

- The following fields in the RSL control block (if the error occurred in ISGBSRRI):

  RSLWLOCK
  RSLLNKI
  RSLNMSC
  RSLLKSF
  RSLLKIF
  RSLBFCT
  RSLERR

COMPON = GRS,COMPID = SCSDS,ISSUER = ISGDSNRV

Component: Global Resource Serialization (5752-SCSDS)

Issuing Module: ISGDSNAP

PLM: *OS/VS2 MVS Global Resource Serialization Logic*

Explanation: An error has occurred while ISGDSNAP (snap dump exit) was processing. ESTAE routine ISGDSNRV (in ISGDSNAP) issues the SDUMP macro.

Problem Determination: A software record is written to SYS1.LOGREC.

COMPON = GRS,COMPID = SCSDS,ISSUER = ISGGFRR0

Component: Global Resource Serialization (5752-SCSDS)

Issuing Module: ISGGFRR0 - FRR

PLM: *OS/VS2 MVS Global Resource Serialization*

Explanation: An error has occurred while one of the following modules was processing.

ISGGDEQP
ISGGNQDQ
ISGGQWBC
ISGGQWBI
ISGGREX0
ISGGRP00
ISGGTRM0
ISGGTRM1
ISGLNQDQ
ISGSALC
ISGSDAL
ISGSHASH

The FRR ISGGFRR0 uses the branch entry to SVC dump. A summary dump is requested that includes the GVT and GVTX control blocks. An asynchronous dump of the current address space is also included in the dump request.

Problem Determination: A software record is written to SYS1.LOGREC.

COMPON = GRS,COMPID = SCSDS,ISSUER = ISGSMIFR

Component: Global Resource Serialization (5752-SCSDS)

Issuing Module: ISGSMI

PLM: *OS/VS2 MVS Global Resource Serialization*

Explanation: Either a program check has occurred while ISGSMI, ISGSALC, or ISGSDAL was processing, or an abend has occurred while ISGSALC was processing. The FRR routine ISGSMIFR (in ISGSMI) uses the branch entry to SVC dump. The areas dumped are PSA, SQA, and GRSQ. The dump also contains a summary dump.

Problem Determination: A software record is written to SYS1.LOGREC.

COMPON=IOS,COMPID=SC1C3,ISSUER=IECVBRSV

Component: IOS (5752-SC1C3)

Issuing Module: IECVBRSV

PLM: *OS/VS2 I/O Supervisor Logic*

Explanation: An error has occurred during IECVBRSV (build reserve table) processing. The FRR routine in IECVBRSV issues the SDUMP macro. The areas dumped are SQA, NUC, SUM, and ALLPSA.

Problem Determination: A software record is written to SYS1.LOGREC. The SDWAVRA contains the FRR work area.

COMPON=IOS,COMPID=SC1C3,ISSUER=IECVDPTH

Component: IOS (5752-SC1C3)

Issuing Module: IECVDPTH

PLM: *OS/VS2 I/O Supervisor Logic*

Explanation: An error has occurred during IECVDPTH (dynamic pathing) processing. The FRR routine (DPTHFRR) in IECVDPTH issues the SDUMP macro. The areas dumped are SQA, NUC, ALLPSA, and SUM.

Problem Determination: A software record is written to SYS1.LOGREC. The SDWAVRA is formatted in the key-length-data format.

COMPON=IOS,COMPID=SC1C3,ISSUER=IECVFCHN,FCHNFRR

Component: IOS (5752-SC1C3)

Issuing Module: IECVFCHN

PLM: *OS/VS2 I/O Supervisor Logic*

Explanation: An error has occurred during IECVFCHN (force channel offline) processing. The FRR routine (FCHNFRR) in IECVFCHN issues the SDUMP macro.

Problem Determination: A software record is written to SYS1.LOGREC and includes:

SDWAMODN	- IECVFCHN
SDWACSCT	- IECVFCHN
SDWAREXN	- IECVFCHN
SDWACID	- SC1C3
SDWARRL	- FCHNFRR
SDWAVRA	- The six-word FRR work area, and the first reserve table segment if any devices were entered in the reserve table segment.

COMPON=IOS,COMPID=SC1C3,ISSUER=IECVFDEV,FDEVFRR, FAILURE IN FORCE DEVICE ROUTINE

Component: IOS (5752-SC1C3)

Issuing Module: IECVFDEV

PLM: *OS/VS2 I/O Supervisor Logic*

Explanation: An error has occurred during IECVFDEV (force device) processing. The FRR routine FDEVFRR in IECVFDEV issues the SDUMP macro.

Problem Determination: A software record is written to SYS1.LOGREC and includes:

SDWAMODN	- IECVFDEV
SDWACSCT	- IECVFDEV
SDWAREXN	- IECVFDEV
SDWACID	- SC1C3
SDWARRL	- FDEVFRR
SDWAVRA	- The 24-byte FRR parameter list.

COMPON=IOS,COMPID=SC1C3,ISSUER=IECVGENA

Component: IOS (5752-SC1C3)

Issuing Module: IECVGENA - FRR

PLM: None - refer to the microfiche

Explanation: A program check has occurred during IECVGENA (IOSGEN resident subroutine) processing. The areas dumped are RGN, LPA, and TRT.

Problem Determination: A software record is written to SYS1.LOGREC. The SDUMP buffer contains:

- Parameters indicating the function performed
- Register save area
- SDWA
- The 6 word FRR parameter area

COMPON=IOS,COMPID=SC1C3,ISSUER=IECVHREC

Component: IOS (5752-SC1C3)

Issuing Module: IECVHREC

PLM: *OS/VS2 I/O Supervisor Logic*

Explanation: An error has occurred during IECVHREC (hot I/O recovery) processing. The FRR routine in IECVHREC issues the SDUMP macro. The areas dumped are SQA, NUC, SUM, and ALLPSA.

Problem Determination: A software record is written to SYS1.LOGREC. The SDWAVRA contains the FRR work area and a copy of the SCD.

COMPON=IOS,COMPID=SC1C3,ISSUER=IECVIOSI

Component: IOS (5752-SC1C3)

Issuing Module: IECVIOSI

PLM: *OS/VS2 I/O Supervisor Logic*

Explanation: An error has occurred during IECVIOSI (dynamic pathing initialization) processing. The ESTAE routine (IOSIRECV) in IECVIOSI issues the SDUMP macro. The areas dumped are SQA, RGN, TRT, LSQA, SWA, SUM, and automatic storage area.

Problem Determination: A software record is written to SYS1.LOGREC. The SDWAVRA is formatted in the key-length-data format.

COMPON=IOS,COMPID=SC1C3,ISSUER=IECVRRSV

Component: IOS (5752-SC1C3)

Issuing Module: IECVRRSV

PLM: *OS/VS2 I/O Supervisor Logic*

Explanation: An error has occurred during IECVRRSV (re-reserve device) processing. The FRR routine in IECVRRSV issues the SDUMP macro. The areas dumped are SQA, NUC, SUM, and ALLPSA.

Problem Determination: A software record is written to SYS1.LOGREC. The SDWAVRA contains the FRR work area.

COMPON=IOS,COMPID=SC1C3,ISSUER=IOSVRSUM - RESUME SERVICE ROUTINE

Component: IOS (5752-SC1C3)

Issuing Module: IOSVRSUM

PLM: *OS/VS2 I/O Supervisor Logic*

Explanation: An error has occurred during IOSVRSUM (resume I/O service) processing. The areas dumped are SQA, PSA, NUC, and TRT.

Problem Determination: A software record is written to SYS1.LOGREC. The SDWAVRA contains the FRR work area. It also contains the UCB and IOSB if their addresses are not zero. If the UCB or IOSB address is zero, then the SDWAVRA contains 'UCB ZERO' or 'IOSB ZERO'.

COMPON=JES3 SUBSYS COMMUNIC,COMPID=SC1BA,
ISSUER=IATSSRE(SSREFRR)

Component: JES3 (5752-SC1BA)

Issuing Module: IATSSRE

PLM: *OS/VS2 MVS JES3 Logic*

Explanation: An error has occurred during read end processing of subsystem communication. Recovery routine SSREFRR issues the SDUMP macro.

Problem Determination: A software record is written to SYS1.LOGREC.

COMPON=JES3 SUBSYS COMMUNIC,COMPID=SC1BA,
ISSUER=IATSSXM(SXMFRR)

Component: JES3 (5752-SC1BA)

Issuing Module: IATSSXM

PLM: *OS/VS2 MVS JES3 Logic*

Explanation: An error has occurred during cross memory processing of subsystem communication. Recovery routine SXMFRR issues the SDUMP macro.

Problem Determination: A software record is written to SYS1.LOGREC.

COMPON=MSTR-WAIT,COMPID=SC1B8,ISSUER=IEEVWAIT,reason

Component: Master Scheduler Commands (5752-SC1B8)

Issuing Module: IEEVWAIT

PLM: *OS/VS2 System Logic Library*

Explanation: An error has occurred during command processing. The 'reason' field is one of the following:

BAD ESTAE RETURN CODE
ERROR IN MASTER ADDR SPACE
ERROR IN CONSOLE ADDR SPACE
IEEVWAIT RESTART FAILED IN CONSOLE ADDR SPACE

IEEVWAIT issues SDUMP for all but percolation and machine check entries. The areas dumped are PSA, NUC, LSQA, RGN, LPA, TRT, CSA, GRSQ, and SQA.

Problem Determination: A software record is written to SYS1.LOGREC.

COMPON=M S CMNDS,COMPID=SC1B8,ISSUER=IEECB862, FAILURE IN VARY ONLINE/OFFLINE/CONSOLE PROCESSOR

Component: Master Scheduler Commands (5752-SC1B8)

Issuing Module: IEECB862

PLM: *System Logic Library*

Explanation: An error has occurred in the VARY device command. The areas dumped are SQA, ALLPSA, LSQA, LPA, TRT, and GRSQ. In addition, the UCM and UCMEs are dumped using a storage list.

Problem Determination: A software record is written to SYSl.LOGREC. The SDWAVRA contains the following:

- Pointer to the vary service routine interface list (VSRI)
- The vary footprints
- Pointer to the XSA
- Pointer to the CSCB
- The command operand from CHBUF
- The command verb code
- The caller's token

COMPON=M S CMNDS,COMPID=SC1B8,ISSUER=IEEMB881, FAILURE IN SYSTEM ADDR SPACE CREATE ROUTINE

Component: Master Scheduler Commands (5752-SC1B8)

Issuing Module: IEEMB881 - System address space create routine

PLM: *OS/VS2 System Initialization Logic*

Explanation: An error occurred, after master scheduler initialization, while IEEMB881 was attempting to start a system address space. Routine EAESTAE issues the SDUMP macro. The areas dumped are SQA, PSA, LSQA, LPA, TRT, GRSQ, and the master scheduler's ASCB.

Problem Determination: A software record is written to SYS1.LOGREC. The SDWAVRA contains the following:

- Return and reason codes
- Footprints
- Input attribute list
- Name of the initialization routine specified by the caller
- Start parameters specified by the caller
- Code and data registers
- Pointers to the CSCB, ASCB, JSCB, TCB, and BASEA

COMPON=M S CMDS,COMPID=SC1B8,ISSUER=IEEMB883, FAILURE IN SYSTEM ADDR SPACE INIT WAIT/POST ROUTINE

Component Master Scheduler Commands (5752-SC1B8)

Issuing Module: IEEMB883 - System address space initialization WAIT/POST routine

PLM: *OS/VS2 System Initialization Logic*

Explanation: An error occurred, after master scheduler initialization, during WAIT/POST processing. Routine WPESTAE issues the SDUMP macro. The areas dumped are SQA, PSA, LSQA, LPA, and TRT.

Problem Determination: A software record is written to SYS1.LOGREC. The SDWAVRA contains the following:

- Return and reason codes
- Input event code
- Footprints
- Code and data registers
- Pointer to TCB in error
- Pointers to the CSCB, ASCB, JSCB, and BASEA

COMPON=MS CMNDS,COMPID=SC1B8,ISSUER=IEEMB887,
GENERALIZED PARSER,ABEND=xxx,
RSN=xxxxxxxx|UNKNOWN
or
COMPON=MS CMNDS,COMPID=SC1B8,ISSUER=IEEMB887,
GENERALIZED PARSER-EXIT ABENDED,ABEND=xxx,
RSN=xxxxxxxx|UNKNOWN

Component: Master Scheduler (5752-SC1B8)

Issuing Module: IEEMB887 - Generalized Parser

PLM: *System Logic Library*

Explanation: An error has occurred: (1) in module IEEMB887, or (2) in an exit routine that was called by IEEMB887.

Recovery routine PRSESTAE issued a summary SVC dump with the following areas included:

- IEEMB887
- IEEMB887's data area
- SCL (IEEMB887's parameter list)
- First parse description
- Current parse description
- Input being processed

Problem Determination: A software record is written to SYS1.LOGREC. The SDWAVRA contains:

- 'ENABLING DAE'
- If 'ROUT' exit routine abended, exit routine address with the address of the keyword used to call the routine
- If I/O exit abended, exit routine address
- Footprints
- Base registers
- Data register
- Address of SCL
- Address of current parse description
- Current value of input record pointer

COMPON=PROGRAM-MANAGER-VIRTUAL-FETCH,COMPID=SC1CJ, ISSUER=CSVVFCES-CSVVFCRE

Component: Virtual Fetch (5752-SC1CJ)

Issuing Module: CSVVFCRE - issued by ESTAE CSVVFCES

PLM: *OS/VS2 System Logic Library*

Explanation: An abend (other than code 222, 322, or 522) occurred during virtual fetch initialization processing in CSVVFCRE. ESTAE routine CSVVFCES requests an SVC dump. Up to nine storage ranges are dumped and include:

- The VFCB pointed to by CVTVFCB.
- The VFCB obtained by this task and pointed to by VFCBPTR.
- The old hash table, if a new hash table was obtained.
- The new hash table.
- The temporary table of PDS directory entries (INFOTAB).
- The ACA area.
- The VCBs area.
- The area used as a VIO window for reformatted modules.
- The area used as an input buffer for load module TXT records.

Problem Determination: A software record is written to SYS1.LOGREC. Register 1 contains the address of the SDWA. The following fields in the SDWA are filled in: SDWAMODN, SDWACSCT, SDWAREXN, SDWASC, SDWALVL, SDWARRL, and SDWACID.

Field CVTVFCB points to the VFCB. Field VFCBASCB points to the ASCB of the address space that owns the VFCB. CSVVFCRE sets VFCBASCB early in its processing. The VFCB also indicates whether

CSVVFCRE was doing an initial build (VFCBECB=0) or a refresh (VFCBECB"0).

In CSECT CSVVFCR1, local variable IOERTEXT contains a text description of the most recent I/O error detected by BSAM while reading modules. It is filled in by BSAM whenever the SYNAD exit is entered for the DCB named DCBLIBS.

In CSECT CSVVFCRE, field USERPRMS contains processing footprints that indicate the resources owned and the stage of processing at the time of the dump. USERPRMS is mapped by the structure FOOTPRTS in CSVVFCRE. Field RETRY in CSVVFCRE indicates whether CSVVFCES was retrying (RETRY=ON) or percolating the error. The intended retry address is in field RETRYADR in CSVVFCRE.

In the case when the ESTAE routine is percolating the error, no clean up has been performed and the footprints will describe the stage of processing at the time of the error.

COMPON=PROGRAM-MANAGER-VIRTUAL-FETCH,COMPID=SC1CJ,
ISSUER=CSVVFCFR-CSVVFCRE

Component: Virtual Fetch (5752-SC1CJ)

Issuing Module CSVVFCRE - issued by FRR CSVVFCFR

PLM: *OS/VS2 System Logic Library*

Explanation: During refresh processing of the virtual fetch service address space, an abend occurred in CSVVFCES (CSVVFCRE's ESTAE routine) while it was attempting to release the logical group number (LGN) of the new VIO data set obtained by CSVVFCRE. Refresh processing failed after the new LGN was obtained, and before the VFCB had been updated for the new generation. CSVVFCFR requests an SVC dump only when flag SDWACLUP is on, or when the FRR was reentered after attempting a retry.

Up to nine storage areas are dumped and include:

- The VFCB pointed to by CVTVFCB.
- The VFCB obtained by this task and pointed to by VFCBPTR.
- The old hash table.
- The new hash table.
- The temporary table of PDS directory entries (INFOTAB).
- The ACA area.
- The VCBs area.
- The area used as a VIO window for reformatted modules.
- The area used as an input buffer for load module TXT records.

Problem Determination: A software record is written to SYS1.LOGREC. Register 1 contains the address of the SDWA. The following fields in the SDWA are filled in: SDWAMODN, SDWACSCT, SDWAREXN, SDWASC, SDWALVL, SDWARRL, and SDWACID.

In CSECT CSVVFCR1, local variable IOERTEXT contains a text description of the most recent I/O error detected by BSAM while reading modules. It is filled in by BSAM whenever the SYNAD exit is entered for the DCB named DCBLIBS.

The VRA area in the SDWA contains a copy of variable USERPRMS as it was on entry to CSVVFCFR. Variable USERPRMS (in CSVVFCRE) contains processing footprints that indicate the resources owned and the stage of processing at the time of the dump. USERPRMS is mapped by the structure FOOTPRTS in CSVVFCRE. If this is a recursion, flag FPFRRCUR in USERPRMS is set on.

COMPON=RMF-ENQ EVENT HANDLER,COMPID=XY400,ISSUER=ERBMFEEQ

Component: RMF (5752-XY400)

Issuing Module: ERBMFEEQ

PLM: *OS/VS2 MVS Resource Measurement Facility (RMF) Version 2 Program Logic Manual*

Explanation: An abend occurred while the RMF Monitor 1 ENQ event handler (ERBMFEEQ) was processing. EREMFEEQ receives control when an increase or decrease in enqueue contention occurs. Recovery routine ERBMFFRQ issues the SDUMP macro. The areas dumped are SUMDUMP and TRT. The SUMLIST option specifies the ERBMFEEQ module work area and the ENQ data collection area (ERBEQEDT and ERBEQRES).

Problem Determination: A software record is written to SYS1.LOGREC. The SDWAVRA contains module trace information and pointers to the module work area and the ERBEQEDT area.

COMPON=SUPV CNTL,COMPID=SC1C5,FUNCTION=RESUME, MODULE=IEAVETCL,ISSUER=IEAVETCL(IEAVETCR)

Component: Supervisor Control (5752-SC1C5)

Issuing Module: IEAVETCL

PLM: *OS/VS2 System Logic Library*

Explanation: An error has occurred during RESUME or TCTL processing. Recovery routine IEAVETCR issues the SDUMP macro.

Problem Determination: A software record is written to SYS1.LOGREC. The SDWAVRA contains diagnostic information indicating the function in control at the time of the error, and debugging data related to that function.

DAVV ERROR

Component: IOS - DASD Volume Verification (5752-SC1C3)

Issuing Module: IECVDAVV - FRR/ESTAE

PLM: *OS/VS2 I/O Supervisor Logic*

Explanation: An error has occurred while IECVDAVV was in control. The areas dumped are PSA, TRT, dump buffer, and storage containing the module.

Problem Determination: The dump buffer contains:

X‘0’	- SRB/IOSB
X‘A6’	- EWA (work area)
X‘146’	- UCB

The SDWAVRA also contains a copy of the IOSB. A software record is written to SYS1.LOGREC.

D U,,ALLOC ABEND

Component: DIDOCS (5752-SC1C4)

Issuing Module: IEE24110 - ESTAE

PLM: *OS/VS2 System Logic Library*

Explanation: An error has occurred during D U,ALLOC (DISPLAY) processing. Any storage areas obtained are freed. The following areas are dumped for both the master and the allocation address space: SQA, PSA, NUC, LSQA, RGN, LPA, TRT, and CSA. The ESTAE routine percolates to IEECB860.

Problem Determination: A software record is written to SYS1.LOGREC.

DUMP BY/(OF) MODULE xxxxxxxx

Component: GTF (5752-SC111)

Issuing Module: AHLWTO

PLM: *OS/VS2 Service Aids Logic*

Explanation: Entry point AHLDMPMD in AHLWTO provides a dumping service for the GTF FGBRs (filter, gather, and build routines). xxxxxxxx indicates the FGBR affected: AHLTSLIP, AHLTSYSM, AHLTUSR, AHLTSIO, AHLTSVC, AHLTPID, AHLTSYFL, AHLTEXT, AHLTFOR, or AHLTXSYS. The GTF control blocks dumped are MCHEAD, MCRWSA, MCAWSA, MCCE, MCQE, and GTFPCT. The SQA, SDWA, and the failing FGBR module are also dumped.

Problem Determination: The error is probably a page fault that occurred when the FGBR referenced a data area that should be fixed but was not.

Message AHL118I is issued. For additional information, refer to message AHL118I in *VS2 System Messages.* For most errors, the GTF FGBR also writes a software record to SYS1.LOGREC.

DUMP OF AHLREADR

Component: GTF (5752-SC111)

Issuing Module: AHLREADR

PLM: *OS/VS2 Service Aids Logic*

Explanation: An error has occurred while AHLREADR was attempting to pass GTF buffers to SDUMP or SNAP for inclusion in an outstanding dump request. The dump taken by AHLREADR includes a dump of itself plus a dump of the failing address space. The AHLREAD macro request is cleaned up, which includes posting the original requestor, releasing locks, dequeueing on the MC (monitor call) control blocks, and releasing allocated storage.

Problem Determination: A software record is written to SYS1.LOGREC.

DUMP OF GTF MODULE AHLSBLOK

Component: GTF (5752-SC111)

Issuing Module: AHLSBUF

PLM: *OS/VS2 Service Aids Logic*

Explanation: An error has occurred while moving the GTF global trace buffer to a page in the GTF address space. The failing address space is dumped. The error is percolated to the FRR for the active data gathering routine. The FRR in the router routine (AHLMCER) disables and terminates GTF.

Problem Determination: A software record is written to SYS1.LOGREC.

DUMP OF GTF MODULE AHLWTASK

Component: GTF (5752-SC111)

Issuing Module: AHLWTASK

PLM: *OS/VS2 Service Aids Logic*

Explanation: An error has occurred when either (1) entry point AHLWPOST in AHLWTASK was attempting to post AHLWTASK in order to schedule message AHL118I, or (2) entry point AHLWTASK was attempting to schedule an SRB for a WTO of message AHL118I. The areas dumped are the SDUMP buffer, failing module, and failing address space.

Problem Determination: Message AHL119I is issued. The SDUMP buffer contains message AHL118I (which would have been issued if the error had not occurred), the SRB that did not complete, and the SDWA.

DUMP OF JES2 CHECKPOINT DATA. SYSTEM=id, $ERROR CODE=code

Component: JES2 or JES2 NJE (5752-SC1BH)

Issuing Module: HASPCKPT

PLM: *OS/VS2 MVS JES2 Logic* or *Network Job Entry Facility for JES2 Logic*

Explanation: JES2 has detected a major error during I/O processing to the checkpoint data set. Fields in the dump title are:

id - system ID on which the error was detected
code - JES2 abend code

The JES2 actual checkpoint master record, job queue, and JOT storage are dumped.

Problem Determination: For additional information on JES2 error codes, refer to message HASP095 in *VS2 System Codes.* For additional information on the action to be taken, refer to *SPL: JES2* or *SPL: NJE for JES2.*

DYNAMIC DEVICE RECOVERY ERROR DUMP

Component: RMS (5752-SC1CE)

Issuing Module: IGFDE1 - DDR ESTAE exit

PLM: *OS/VS2 System Logic Library*

Explanation: An error has occurred during DDR (dynamic device reconfiguration) processing. The areas dumped are PSA, TRT, SQA, NUC, module storage, and DDRCOM (the DDR general work area.

Problem Determination: A software record is written to SYS1.LOGREC. Also, the ASXBDDR field in the ASXB of the master address space points to the current DDRCOM. The DDRCOM might contain useful debugging information.

ENF ABEND ERRORMOD=IEFENFFX

Component: Scheduler Services (5752-BB131)

Issuing Module: IEFENFFX

PLM: *OS/VS2 System Logic Library*

Explanation: An abend has occurred while IEFENFFX (ENF request router routine) was processing an event notification request. The areas dumped are NUC and SQA.

Problem Determination: A software record is written to SYS1.LOGREC. The SDWAVRA contains the ESTAE or FRR parameter list and footprint bits that indicate the execution path of IEFENFFX.

ENF ABEND ERRORMOD=IEFENFNM

Component: Scheduler Services (5752-BB131)

Issuing Module: IEFENFNM

PLM: *OS/VS2 System Logic Library*

Explanation: An abend has occurred while IEFENFNM (ENF mainline routine) was processing an event notification request. The areas dumped are NUC, RGN, CSA, and SQA.

Problem Determination: A software record is written to SYS1.LOGREC. The SDWAVRA contains the ESTAE or FRR parameter list and footprint bits that indicate the execution path of IEFENFNM.

ENQUEUE FAILURE IN ABDUMP

Component: RTM - ABDUMP Processing (5752-SC1CM)

Issuing Module: IEAVTABD

PLM: *OS/VS2 System Logic Library*

Explanation: An error has occurred when ABDUMP attempted to enqueue (ENQ) on the dump data set in order to process a dump request specified on a user's SYSABEND, SYSMDUMP, or SYSUDUMP DD statement. The areas dumped are RGN, LPA, LSQA, and SQA.

Problem Determination: Because ABDUMP was unable to take the dump requested by the user, use the SVC dump information to identify the user's problem.

ERROR DURING RESTART PROCESSING - IEFXB609

Component: Scheduler Restart (5752-SC1B3)

Issuing Module: IEFXB609 - Data Set Descriptor Record Processor

PLM: *OS/VS2 System Logic Library*

Explanation: An error has occurred during restart processing. The areas dumped are NUC, LSQA, RGN, LPA, TRT, SWA, and SQA.

Problem Determination: A software record is written to SYS1.LOGREC.

ERROR DURING SNAP

Component: RTM - Dump Services (5752-SC1CM)

Issuing Module: IEAVAD01 - ESTAE

PLM: *OS/VS2 System Logic Library*

Explanation: An error has occurred during SNAP dump processing when SNAP was attempting to take a dump for the user. An I/O error or invalid control block field can cause this error. The areas dumped are RGN, LPA, and SQA.

Problem Determination: A software record is written to SYS1.LOGREC. It includes the failing CSECT name that identifies the formatter in control at the time of the error.

ERROR IN AHLSETEV

Component: GTF (5752-SC111)

Issuing Module: AHLSETEV

PLM: *OS/VS2 Service Aids Logic*

Explanation: A program check has occurred when referencing the MC (monitor call) tables that are built during GTF initialization by the SETEVENT macro. GTF applications are terminated and acquired resources are freed. Message AHL132I is issued. The area dumped is SQA, which contains the MC tables.

Problem Determination: Validate the MC tables that are located in the SQA For additional information, refer to message AHL132I in *VS2 System Messages*.

ERROR IN GETMAIN/FREEMAIN

Component: VSM (5752-SC1CH)

Issuing Module: IEAVGFRR - FRR

PLM: *OS/VS2 System Logic Library*

Explanation: One of the following errors has occurred during IEAVGM00 (GETMAIN/FREEMAIN service routine) processing:

- The SALLOC lock-release return code was not zero.
- The page release return code was not zero or eight.
- The create segment return code was not zero.
- The find page return code was not zero or four.
- GFRECORE failed during SQA or LSQA expansion.
- The SALLOC lock obtain return code was not zero or four.
- Subpool freemain for subpool 253 found an AQE whose area was not in an LSQA DQE.

The areas dumped are PSA, LSQA, and SQA.

Problem Determination: A software record is written to SYS1.LOGREC. For additional debugging information, refer to the topics "Virtual Storage Allocation (IEAVGM00 - GETMAIN/FREEMAIN" and "GETMAIN's Functional Recovery Routine - IEAVGFRR)" under VSM in Section 5 of this publication.

ERROR IN IATSIDMO FOR SYSOUT DATA SET

Component: JES3 (5752-SC1BA)

Issuing Module: IATDMFR - FRR

PLM: *OS/VS2 MVS JES3 Logic*

Explanation: An error has occurred while module IATSIDM (USAM subsystem interface routine) was attempting to open a SYSOUT data set. The FRR routine IATDMFR issues the SDUMP macro. IATDMFR returns to IATSIDM via the retry address (RETADDR parameter) on the SETRP macro. IATSIDM terminates the job with a 1FB system abend code. The areas dumped are SQA, CSA, and LPA.

Problem Determination: A software record is written to SYS1.LOGREC. For a description of the 1FB abend code, refer to *VS2 System Codes.*

ERROR IN IEAVTSLP

Component: RTM - SLIP Processor (5752-SC1CM)

Issuing Module: IEAVTSLR

PLM: *OS/VS2 System Logic Library*

Explanation: An error has occurred during SLIP processing. The FRR in IEAVTSLR issues SDUMP. The areas dumped are ALLPSA, NUC, LSQA, and SQA. The summary part of the dump requested by IEAVTSLR contains information relevant to the error.

Problem Determination: A software record is written to SYS1.LOGREC. The FRR parameter list is put in the SDWAVRA.

ERROR IN MASTER SUBSYSTEM BROADCAST FUNCTION, ABEND=aaa,COMPON=INIT-SSI,COMPID=SC1B6,ISSUER=IEFJRASP

Component: Initiator - Subsystem Interface (5752-SC1B6)

Issuing Module: IEFJRASP

PLM: *OS/VS2 System Logic Library*

Explanation: An abend has occurred while IEFJRASP was routing a subsystem interface request to all active subsystems, via the subsystem interface. The areas dumped are NUC, CSA, LPA, TRT, and LSQA.

Problem Determination: A software record is written to SYS1.LOGREC. The SDWAVRA contains footprint bits that indicate the execution path of IEFJRASP.

ERROR IN MODULE AHLMCER

Component: GTF (5752-SC111)

Issuing Module: AHLMCER

PLM: *OS/VS2 MVS Service Aids Logic*

Explanation: An error has occurred during GTF processing when AHLMCER attempted to route the MC (monitor call) interruption to its affiliated FGBR (filter, gather, and build routine). The FRR routine (AHLMCFRR) requests the dump prior to attempting retry. The MCRWSA and SDWA are moved into the SDUMP buffer. AHLMCER is included in the dump as part of the storage dumped. GTF is terminated. The areas dumped are SQA, SDUMP buffer, failing module, and failing address space.

Problem Determination: Message AHL007I is issued. A software record is written to SYS1.LOGREC. This error is usually an inability to pass control to an FGBR because of changes to the FGBR in SYS1.LPALIB. Field

MCREID in the MCRWSA contains the event identifier of the HOOK that GTF was processing.

ERROR IN QMNGRIO PROCESSING

Component: RTM - Dump Services (5752-SC1CM)

Issuing Module: IEAVAD01 - ESTAE

PLM: *OS/VS2 System Logic Library*

Explanation: An error has occurred during SNAP dump processing when the QMNGRIO macro attempted to read the JFCB in order to obtain an output line and the page capacity. The areas dumped are RGN, LPA, SWA, and SQA.

Problem Determination: The JFCB might be in error.

ERROR IN REAL STORAGE MANAGER

Component: RSM (5752-SC1CR)

Issuing Module: IEAVRCV - FRR

PLM: *OS/VS2 System Logic Library*

Explanation: An error has occurred during RSM processing. The areas dumped are SQA, PVT, and PFT. If IEAVRCV was running in the failing address space, LSQA is also dumped.

Problem Determination: A software record is written to SYS1.LOGREC and includes:

SDWAMODN	- Name of the module in error.
SDWACSCT	- CSECT name of the routine in control at the time of the error.
SDWAREXN	- IEAVRCV (recovery routine)
SDWAVRA	- The 24-byte FRR parameter list (RCA).

The RCA contains:

Byte 9	- CSECT ID of the module that issued the SETFRR to put IEAVRCV on the recovery stack. The IDs are defined in mapping macro IHARCA.
Bytes 8, 12-17	- Footprint bits that indicate which RSM module was in control at the time of the error. A description of the bits is in mapping macro IHARCA.
Byte 11	- Reason code for COD abends. An explanation of RSM abend reason codes is in the "Real Storage Manager (RSM)" topic.

Refer to mapping macro IHARCA for an explanation of all bytes in the RCA.

ERROR IN REAL STORAGE MANAGER FRR

Component: RSM (5752-SC1CR)

Issuing Module: IEAVRCV - FRR

PLM: *OS/VS2 System Logic Library*

Explanation: An error has occurred while the FRR routine IEAVRCV was processing. Routine IEAVRCV3 issues the SDUMP macro. The areas dumped are SQA, PVT, and PFT.

Problem Determination: A software record is written to SYS1.LOGREC and includes:

SDWAMODN	- IEAVRCV (module in error)
SDWACSCT	- IEAVRCV (CSECT in error)
SDWAREXN	- IEAVRCV3 (recovery routine)

ERROR IN SUBSYSTEM SERVICE RTN, COMPON=INIT-SSI,
COMPID=SC1B6,ISSUER=IEFJSBLD,ABEND=aaa

Component: Initiator - Subsystem Interface (5752-SC1B6)

Issuing Module: IEFJSBLD

PLM: *OS/VS2 System Initialization Logic*

Explanation: An abend (aaa) has occurred while IEFJSBLD was either building an SSCVT, SSVT, SHAS, or SAST, or was preparing to link to a subsystem's initialization routine. The areas dumped are ALLPSA, LSQA, RGN, CSA, and TRT.

Problem Determination: A software record is written to SYS1.LOGREC and includes:

SDWAMODN	- IEEVIPL (module in error)
SDWACSCT	- IEFJSBLD (CSECT in error)
SDWAREXN	- IEFJSBLD (recovery routine)

The SDWAVRA contains the input parameter list and footprint bits that indicate the execution path of IEFJSBLD.

ERROR IN SUBSYSTEM INITIALIZATION,COMPON=INIT-SSI,
COMPID=SC1B6,ISSUER=IEFJSIN2,ABEND=aaa

Component: Initiator - Subsystem Interface (5752-SC1B6)

Issuing Module: IEFJSIN2

PLM: *OS/VS2 System Initialization Logic*

Explanation: An abend (aaa) occurred during initialization processing of the subsystems. The error occurred in IEFJSIN2 or in service routines IEEMB878 or IEEMB882. The areas dumped are ALLPSA, LSQA, RGN, and TRT.

Problem Determination: A software record is written to SYS1.LOGREC and includes:

SDWAMODN	- IEEMB860 (module in error)
SDWACSCT	- IEFJSIN2 (CSECT in error)
SDWAREXN	- IEFJSIN2 (recovery routine

The SDWAVRA contains footprint bits to indicate the execution path of IEFJSIN2.

EVENT NOTIFICATION FACILITY ERROR,ABEND=xxx,
COMPON=SCHR-ENF,COMPID=BB131,ISSUER=IEFENFWT

Component: Scheduler Services (5752-BB131)

Issuing Module: IEFENFWT

PLM: *OS/VS2 System Logic Library*

Explanation: An abend has occurred while IEFENFWT (ENF wait routine) was processing. The areas dumped are NUC, CSA, SQA, and RGN.

Problem Determination: A software record is written to SYS1.LOGREC.

FAILURE DURING SNAP RECOVERY

Component: RTM - Dump Services (5752-SC1CM)

Issuing Module: IEAVAD01 - ESTAE

PLM: *OS/VS2 System Logic Library*

Explanation: An error has occurred while the SNAP dump ESTAE routine was attempting to cleanup after an error had occurred during SNAP mainline processing. No further cleanup is attempted. The areas dumped are RGN, LPA, and SQA.

Problem Determination: The SNAP storage buffers are probably incorrect. Use the previous RTM2WA to identify the error that occurred during SNAP mainline processing. The SNAP mainline error might have affected this error.

FIOD:IDA019S2 - ABEND FROM FIOD FRR

Component: VSAM - Record Management (5752-SC1DE)

Issuing Module: IDA019S2 - FRR

PLM: *OS/VS2 VSAM Logic*

Explanation: An abnormal termination has occurred during VSAM record management processing. The FRR routine IDA019S2 (at entry point IDAF19S2), issues the SDUMP macro. The areas dumped are PSA, NUC, RGN, TRT, CSA, and SQA.

Problem Determination: A VSAM ICIP (improved control interval processing) request was executing in supervisor state or SRB mode and encountered a program check while the I/O manager was processing the request. Register 3 points to the IOMB for the request. For information on how the IOMB relates to the VSAM control block structure, refer to *OS/VS2 VSAM Logic.*

GTF TERMINATING ON ERROR CONDITION

Component: GTF (5752-SC111)

Issuing Module: AHLTMON

PLM: *OS/VS2 MVS Service Aids Logic*

Explanation: An error has occurred during GTF initialization before the initialization was successfully completed. The retry routine AHLTERM2 (in AHLTMON) issues the SDUMP macro. GTF is terminated. The areas dumped are RGN, LPA, SQA, and MCHEAD control block.

Problem Determination: A software record is written to SYS1.LOGREC.

HASPDUMP SUBSYS=ssss vvvvvvvv MODULE=mmmmmmmm CODE=cccc

Component: JES2 (5752-SC1BH)

Issuing Module: HASPTERM or HASPRAS

PLM: *OS/VS2 MVS JES2 Logic*

Explanation: An error has occurred during JES2 processing. ssss is the subsystem identification, normally JES2 (ssss is obtained from the TIOT); vvvvvvvv is the JES2 version identification; mmmmmmmm is the name of the primary JES2 load module, normally HASJES20; cccc is either the system completion code, Shhh (such as S0C1) or JES2 catastrophic error code, $ccc (such as $K01).

Problem Determination: See message $HASP095 in *System Messages* for an explanation of JES2 error codes, and *System Codes* for an explanation of system codes.

A software record is written to SYS1.LOGREC. Refer to the JES2 LGRR mapping macro in module HASPDOC for a description of SDWAVRA information.

IATSIJS JSESEXIT

Component: JES3 (5752-SC1BA)

Issuing Module: IATSIJS

PLM: *OS/VS2 MVS JES3 Logic*

Explanation: An abend has occurred during IATSIJS (job processing subsystem interface) processing. The ESTAE routine established by IATSIJS is given control to examine the function control table (FCT) active at the time of termination to determine which function or DSP has failed. The areas dumped are PSA, NUC, SQA, RGN, LPA, TRT, and CSA.

Problem Determination: A software record is written to SYS1.LOGREC.

IATSNLS - ESTAE EXIT

Component: JES3 (5752-SC1BA)

Issuing Module: IATSNLS

PLM: *OS/VS2 MVS JES3 Logic*

Explanation: A subtask was terminated because an abend has occurred in one of the following: (1) OPNDST processing, (2) CLSDST exit, (3) CLSDST error exit, (4) SETLOGON exit, (5) SIMLOGON exit, (6) LOGON IRB, (7) TPEND processing, (8) LOSTERM exit, (9) RESPONSE IRB exit, (10) DFSAY exit or (11) OPEN or CLOSE processing (in which case, no retry is attempted). IATSNLS issues the SDUMP macro. The areas dumped are SQA, ALLPSA, NUC, LSQA, RGN, LPA, TRT, and CSA.

Problem Determination: A software record is written to SYS1.LOGREC.

IATSSCM READ-END FAILURE

Component: JES3 (5752-SC1BA)

Issuing Module: IATSSCM

PLM: *OS/VS2 MVS JES3 Logic*

Explanation: An error has occurred during IATSSCM (subsystem communication scheduler) read-end processing. The areas dumped are PSA, NUC, RGN, LPA, TRT, CSA, and SQA.

Problem Determination: A software record is written to SYS1.LOGREC.

IAT1081 ERROR IN IATDMDKT - IATYISR POSSIBLY LOST

Component: JES3 (5752-SC1BA)

Issuing Module: IATDMFR - FRR

PLM: *OS/VS2 MVS JES3 Logic*

Explanation: A software or hardware error has occurred and caused the JES3 channel end termination routine (IATDMDKT) to abnormally terminate. The FRR routine IATDMFR was not able to recover from the error. Either the input/output service block (IOSB) or service request block (SRB) in IATYISR might be invalid. The areas dumped are SQA, LPA, and CSA.

Problem Determination: A software record is written to SYS1.LOGREC. Message IAT1801 is issued. For a description of message IAT1801, refer to *JES3 Messages.*

IAT3702 dspname (ddd) ABENDED/FAILED ABEND code/DMxxx - JES3 FAILURE NO.nnn

Component: JES3 (5752-SC1BA)

Issuing Module: IATABN0

PLM: *OS/VS2 MVS JES3 Logic*

Explanation: A DSP has abended or failed. The text in the SVC dump title identifies the failing DSP (dspname), and the unit address (ddd), if available. The system abend code (code) or the DM type (xxx) is given along with the unique JES3 failsoft identifier (nnn). Message IAT3702 is issued. IATABN0 (online format driver) issues the SDUMP macro. The areas dumped are PSA, NUC, SQA, LSQA, RGN, LPA, TRT, and CSA.

Problem Determination: For additional information: refer to abend codes in *VS2 System Codes*, DM codes in *JES3 Debugging Guide*, and message IAT3702 in *JES3 Messages.*

IAT4830 IATIISB MASTERTASK ABEND

Component: JES3 (5752-SC1BA)

Issuing Module: IATIISB

PLM: *OS/VS2 MVS JES3 Logic*

Explanation: An abend has occurred during IATIISB (interpreter master subtask) processing. The areas dumped are NUC, PSA, RGN, LPA, TRT, and CSA.

Problem Determination: A software record is written to SYS1.LOGREC. Also check the SYSMSG data set for error indications.

IAT4831 IATIISB SUBTASK ABEND

Component: JES3 (5752-SC1BA)

Issuing Module: IATIISB (IATYICT work area)

PLM: *OS/VS2 MVS JES3 Logic*

Explanation: An abend has occurred while an interpreter subtask (R/I or C/I) was processing. Message IAT4211 is issued. IATIISB issues the SDUMP macro. The areas dumped are SQA, PSA, NUC, RGN, LPA, TRT, and CSA.

Problem Determination: A software record is written to SYS1.LOGREC.

ICBRECRD RECORDING FAILED

Component: MSS - MSSC (5752-SC1DP)

Issuing Module: ICBRECRD - ESTAE

PLM: *OS/VS2 MVS Mass Storage System Communicator Logic*

Explanation: An abend has occurred in the 3850 message journaling function of MSS processing. The RESTAEXT routine in module ICBRECRD issues the SDUMP macro. The areas dumped are CSA, LSQA, and TRT.

Problem Determination: A software record is written to SYS1.LOGREC. Message ICB328E is sent to the system console indicating that 3850 message journaling is disabled due to a system failure. The MJX (message journaling exit control block), located in the CSA, indicates the function that ICBRECRD was performing at the time of error (such as, enabling the message journaling function, disabling the message journaling function, or writing a record to the message journaling data set). Also, the MJX contains the queue header for the messages to be written to the message journaling data set.

ICB425I ABEND IN PROCESS - MSVC TASK (nnnnnnnn)

Component: MSS - MSVC (5752-SC1DR)

Issuing Module: ICBVPR00 - ESTAE

PLM: *OS/VS2 MVS Mass Storage System Communicator Logic*

Explanation: An abend has occurred in the MSVC (mass storage volume control) function of MSS processing. The VPRSDUMP routine in module ICBVPR00 issues the SDUMP macro. The areas dumped are PSA, TRT, and CSA. Address ranges are included to dump the following control blocks: SDWA, MSVC control block (IEZVVICB), and the MSSC control block (IEZSSC).

Problem Determination: In the dump title, nnnnnnnn indicates the module in error, ICBVPR00 is a generalized recovery module, therefore, nnnnnnnn can be any module that is part of the MSVC function. Refer to *OS/VS2 MVS Mass Storage System Communicator Logic* for a complete list of MSVC modules. If 'UNKNOWN' appears in the title, ICBVPR00 was unable to determine the module in error. Message ICB425I is sent to the system operator. This message indicates the failing module and contains the MSS order that was being performed at the time of the error. When 'UNKNOWN' appears in the message and SDUMP title, the MSS order can be used to determine which MSVC module was involved in the error. Refer to *OS/VS2 MVS Mass Storage System Communicator Logic* for a description of the MSVC modules and their relationship to MSS orders.

ICHRST00 - RACF SVCS, ABEND CODE=xxx, SVC=svcname, USER=user, GROUP=group, EXIT=installation exit name

Component: RACF - SVC Processing (5752-XXH00)

Issuing Module: ICHRST00 - ESTAE

PLM: *OS/VS2 MVS Resource Access Control Facility (RACF): Program Logic Manual*

Explanation: An abend has occurred during processing of one of the RACF SVCs (ICHRIN00, ICHRCK00, or ICHRDF00). The executing task is terminated. The areas dumped are PSA, RGN, LPA, TRT, CSA, and SQA.

Problem Determination: A software record is written to SYS1.LOGREC and includes:

SDWAMODN	- main CSECT name of SVC
SDWACSCT	- blanks or installation exit name
SDWAREXN	- ICHRST00 (recovery routine)
SDWAGR15	- reason code if the abend is a RACF abend
SDWACRC	- completion code
SDWACID	- XXH00
SDWAEAS	- 1 if SDUMP is taken by ICHRST00
SDWAREQ	- 0 if SDUMP is taken by ICHRST00

ICTMCS01, CRYPTOGRAPHY INITIALIZATION

Component: Programmed Cryptographic Facility (5752-XY500)

Issuing Module: ICTMCS01 - ESTAE

PLM: *OS/VS2 MVS Programmed Cryptographic Facility: Program Logic Manual*

Explanation: An abend has occurred during initialization of the Programmed Cryptographic Facility. The areas dumped are PSA, NUC, LSQA, RGN, LPA, TRT, CSA, SWA, and SQA.

Problem Determination: A software record is written to SYS1.LOGREC and includes:

SDWAMODN	- ICTMCS01 (module in error)
SDWACSCT	- ICTMCS01 (CSECT in error)
SDWAREXN	- ESTAEXIT (recovery routine)

ICTMKG00, KEY GENERATOR PROGRAM

Component: Programmed Cryptographic Facility (5752-XY500)

Issuing Module: ICTMKG00 - ESTAE

PLM: *OS/VS2 MVS Programmed Cryptographic Facility: Program Logic Manual*

Explanation: An abend has occurred during key generator program processing in ICTMKG00. The areas dumped are PSA, NUC, LSQA, RGN, TRT, CSA, and SQA.

Problem Determination: A software record is written to SYS1.LOGREC and includes:

SDWAMODN	- ICTMKG00 (module in error)
SDWACSCT	- ICTMKG00 (CSECT in error)
SDWAREXN	- ESTAEXIT (recovery routine)

ICTMKG01 HANDLE SYSIN MODULE

Component: Programmed Cryptographic Facility (5752-XY500)

Issuing Module: ICTMKG01 - ESTAE

PLM: *OS/VS2 MVS Programmed Cryptographic Facility: Program Logic Manual*

Explanation: An abend has occurred during key generator control statement processing in ICTMKG01. The areas dumped are PSA, NUC, LSQA, RGN, TRT, CSA, and SQA.

Problem Determination: A software record is written to SYS1.LOGREC and includes:

SDWAMODN	- ICTMKG00 (module in error)
SDWACSCT	- ICTMKG01 (CSECT in error)
SDWAREXN	- RECVRTN (recovery routine)

ICTMKM01, START CRYPTOGRAPHY COMMAND

Component: Programmed Cryptographic Facility (5752-XY500)

Issuing Module: ICTMKM01 - ESTAE

PLM: *OS/VS2 MVS Programmed Cryptographic Facility: Program Logic Manual*

Explanation: An abend has occurred during start cryptography command processing in ICTMKM01. The areas dumped are PSA, NUC, LSQA, RGN, LPA, TRT, CSA, SWA, and SQA.

Problem Determination: A software record is written to SYS1.LOGREC and includes:

SDWAMODN	- ICTMKM01 (module in error)
SDWACSCT	- ICTMKM01 (CSECT in error)
SDWAREXN	- ESTAEXIT (recovery routine)

ICTMKM04 - KEY MANAGER

Component: Programmed Cryptographic Facility (5752-XY500)

Issuing Module: ICTMKM04 - FESTAE

PLM: *OS/VS2 MVS Programmed Cryptographic Facility: Program Logic Manual*

Explanation: An abend has occurred during GENKEY or RETKEY macro processing in ICTMKM04. The areas dumped are PSA, NUC, LSQA, RGN, LPA, TRT, CSA, SWA, and SQA.

Problem Determination: A software record is written to SYS1.LOGREC and includes:

SDWAMODN	- ICTMKM04 (module in error)
SDWACSCT	- ICTMKM04 (CSECT in error)
SDWAREXN	- ICTMKM04 (recovery routine)

Message ICT922I is issued to the master console and identifies the requested function and abend code.

ICTMSM07 - ICTMSM07 - CIPHER DUMP

Component: Programmed Cryptographic Facility (5752-XY500)

Issuing Module: ICTMSM07 - FESTAE or FRR

PLM: *OS/VS2 MVS Programmed Cryptographic Facility: Program Logic Manual*

Explanation: An abend has occurred during processing of a request to encipher or decipher data (CIPHER macro) in ICTMSM07. If the CIPHER macro was branch-entered, an FRR was established and a branch entry to SDUMP was used. The areas dumped are NUC, LSQA, RGN, LPA, TRT, CSA, SWA, ALLPSA, and SQA.

Problem Determination: A software record is written to SYS1.LOGREC and includes:

SDWAMODN	- ICTMSM07 (module in error)
SDWACSCT	- ICTMSM07 (CSECT in error)
SDWAREXN	- ERRFES or ERRFRR (recovery routine)

ICTMSM07 - ICTMSM08 TRNSKEY DUMP

Component: Programmed Cryptographic Facility (5752-XY500)

Issuing Module: ICTMSM07 - FESTAE

PLM: *OS/VS2 MVS Programmed Cryptographic Facility: Program Logic Manual*

Explanation: An abend has occurred during the processing of the translate key (TRNSKEY macro) function. The areas dumped are NUC, LSQA, RGN, LPA, TRT, CSA, SWA, ALLPSA, and SQA.

Problem Determination: A software record is written to SYS1.LOGREC and includes:

SDWAMODN	- ICTMSM07 (module in error)
SDWACSCT	- ICTMSM08 (CSECT in error)
SDWAREXN	- ERRFES (recovery routine)

ICTMSM07 - ICTMSM09 EMK DUMP

Component: Programmed Cryptographic Facility (5752-XY500)

Issuing Module: ICTMSM09 - FESTAE

PLM: *OS/VS2 MVS Programmed Cryptographic Facility: Program Logic Manual*

Explanation: An abend has occurred during the processing of the encipher under master key (EMK macro) function. The areas dumped are NUC, LSQA, RGN, LPA, TRT, CSA, SWA, ALLPSA, and SQA.

Problem Determination: A software record is written to SYS1.LOGREC and includes:

SDWAMODN	- ICTMSM07 (module in error)
SDWACSCT	- ICTMSM09 (CSECT in error)
SDWAREXN	- ERRFES (recovery routine)

IDA019SB:IDA121F7 - ABEND FROM BUILD IDACPA

Component: VSAM - Record Management (5752-SC1DE)

Issuing Module: IDA019SB - FRR

PLM: *OS/VS2 VSAM Logic*

Explanation: An abnormal termination has occurred during VSAM record management processing. The FRR in IDA019SB issues the SDUMP macro. This FRR allows termination to continue. The areas dumped are PSA, NUC, RGN, TRT, CSA, and SQA.

Problem Determination: A channel program was being constructed for a VSAM global shared resources (GSR) request. Register 3 points to the

IOMB for the request. For information on how the IOMB relates to the VSAM control block structure, refer to *OS/VS2 VSAM Logic*.

IEAVEAC0 IEAVEAC0 IEAVEAC3

Component: Task Manager - ASCB Dispatching (5752-SC1CL)

Issuing Module: IEAVEAC0 - FRR

PLM: *OS/VS2 System Logic Library*

Explanation: An error has occurred during mainline ASCB CHAP processing. The area dumped is TRT.

Problem Determination: A software record is written to SYS1.LOGREC. The SDWAVRA field contains the associated ASCB-queue-verification output.

IEAVEMCR - MEMORY CREATE ABNORMAL TERMINATION

Component: Supervisor Control (5752-SC1C5)

Issuing Module: IEAVEMCR - Memory Create

PLM: *OS/VS2 System Logic Library*

Explanation: An error has occurred during memory create processing in IEAVEMCR. The ESTAE routine in IEAVEMCR issues the SDUMP macro. The areas dumped are NUC, LPA, TRT, ALLPSA, and SQA.

Problem Determination: A software record is written to SYS1.LOGREC and includes:

SDWAMODN	- IEAVEMCR (module in error)
SDWACSCT	- IEAVEMCR (CSECT in error)
SDWAREXN	- MCRESTAE (recovery routine)

IEAVEMDL - MEMORY DELETE ABNORMAL TERMINATION

Component: Supervisor Control (5752-SC1C5)

Issuing Module: IEAVEMDL - Memory Delete

PLM: *OS/VS2 System Logic Library*

Explanation: An error has occurred during memory delete processing in IEAVEMDL. The ESTAE routine in IEAVEMDL issues the SDUMP macro. The areas dumped are NUC, LPA, TRT, ALLPSA, and SQA.

Problem Determination: A software record is written to SYS1.LOGREC and includes:

SDWAMODN	- IEAVEMDL (module in error)
SDWACSCT	- IEAVEMDL (CSECT in error)
SDWAREXN	- MDLESTAE (recovery routine)

IEAVEMRQ UNEXPECTED ABEND

Component: Supervisor Control (5752-SC1C5)

Issuing Module: IEAVEMRQ - Memory Request

PLM: *OS/VS2 System Logic Library*

Explanation: An error has occurred during memory request processing in IEAVEMRQ while the global dispatcher lock was not held. The ESTAE routine in IEAVEMRQ issues the SDUMP macro. The areas dumped are NUC, LPA, TRT, ALLPSA, and SQA.

Problem Determination: A software record is written to SYS1.LOGREC and includes:

SDWAMODN	- IEAVEMRQ (module in error)
SDWACSCT	- IEAVEMRQ (CSECT in error)
SDWAREXN	- MRQESTAE (recovery routine)

IEAVEMRQ - UNEXPECTED ABEND WITH DISPATCHER LOCK

Component: Supervisor Control (5752-SC1C5)

Issuing Module: IEAVEMRQ - Memory Request

PLM: *OS/VS2 System Logic Library*

Explanation: An error has occurred during memory create (ASID or ASCB) processing while the global dispatcher lock was held. The FRR in IEAVEMRQ issues the SDUMP macro. The areas dumped are NUC, LPA, TRT, ALLPSA, and SQA.

Problem Determination: A software record is written to SYS1.LOGREC. It contains information concerning the associated ASCB and ASCR.

IEAVSY50 IGC001 IGC002 XMPOST FAIL - NO ERRET

Component: Task Manager - POST Processing (5752-SC1CL)

Issuing Module: IEAVSY50 - FRR

PLM: *OS/VS2 System Logic Library*

Explanation: During error recovery, there was not an available error routine (ERRET) when the POST FRR retry routine (SRBRETRY) attempted to schedule the error routine's SRB. This error occurs when the originating address space has terminated. The areas dumped are LSQA, TRT, and SQA.

Problem Determination: A software record is written to SYS1.LOGREC. Register 0 (in the LOGREC record) contains the SRB address. The parameter list addressed via the SRB contains the originating ASCB address, ECB address, ERRET address, post completion code, and key (if specified).

IEAVTRT2 - UNRECOVERABLE ABEND FAILURE

Component: RTM - RTM2 Processing (5752-SC1CM)

Issuing Module: IEAVTRT2

PLM: *OS/VS2 System Logic Library*

Explanation: An unrecoverable error has occurred during RTM2 processing. On completion of IEAVTRT2 processing, the current task tree is set nondispatchable and the failing address space is terminated. The areas dumped are PSA, NUC, RGN, and SQA.

Problem Determination: The most recent RTM2WA addressed by the TCB contains the most pertinent information. However, if a RTM2WA does not exist, not enough storage was available in LSQA or SQA.

IECVESIO ERROR

Component: IOS (5752-SC1C3)

Issuing Module: IECVESIO

PLM: *OS/VS2 I/O Supervisor Logic*

Explanation: An error has occurred during IECVESIO (special I/O) processing. The FRR routine in IECVESIO (ESIOFRR) issues the SDUMP macro. The areas dumped are SQA, NUC, SUM, and ALLPSA.

Problem Determination: A software record is written to SYS1.LOGREC. The SDWAVRA contains the FRR work area.

IECVIOPM PROGRAM ERROR

Component: IOS (5752-SC1C3)

Issuing Module: IECVIOPM

PLM: *OS/VS2 I/O Supervisor Logic*

Explanation: An error has occurred during IECVIOPM (I/O path mask update) processing. The areas dumped are NUC, LPA, TRT, and SQA.

Problem Determination: A software record is written to SYS1.LOGREC. The SDWAVRA field contains a copy of the dynamic work area used by IECVIOPM.

IECVIRST ERROR

Component: IOS (5752-SC1C3)

Issuing Module: IECVIRST

PLM: *OS/VS2 I/O Supervisor Logic*

Explanation: An error has occurred during channel recovery processing in IECVIRST. The areas dumped are PSA, NUC, TRT, and SQA.

Problem Determination: The SDWAVRA field contains the FRR parameter area and the copy of the lost channel mask. The 4K dump buffer contains the IECVIRST general work area and the list of reserved devices on the failing channels.

IECVRDIO ERROR

Component: IOS (5752-SC1C3)

Issuing Module: IECVRDIO

PLM: *OS/VS2 I/O Supervisor Logic*

Explanation: An error has occurred during IECVRDIO (redrive I/O service) processing. The FRR routine in IECVRDIO issues the SDUMP macro. The areas dumped are SQA, NUC, SUM, and ALLPSA.

Problem Determination: A software record is written to SYS1.LOGREC. The SDWAVRA contains the FRR work area. The SDUMP buffer contains the IECVRDIO work area.

IEC251I, VSAM GSR FORCE DLVRP DUMP DATA

Component: VSAM - CLOSE Processing (5752-SC1DE)

Issuing Module: IDA0200T

PLM: *OS/VS2 Virtual Storage Access Method (VSAM) Logic*

Explanation: VSAM was closing the last data set opened against the resource pool, and the ASCB originating the pool had already terminated. A force delete of the pool was done to release resources and storages.

Problem Determination: This is an informational dump (with associated message IEC251I). It indicates that a FORCE DLVRP was done to free storage used by a GSR (global shared resources) pool, with an attempt to dump control blocks to the SYS1.DUMP data set. For additional information, refer to message IEC251I to *VS2 System Messages.*

IEC999I IFG0RR0A,IFG0RR0F,jobname,stepname, WORKAREA=address

Component: Open/Close/EOV (5752-SC1D1)

Issuing Module: IGF0RR0F - ESTAE

PLM: *OS/VS2 Open/Close/EOV Logic*

Explanation: An error has occurred during open, close, or EOV processing. If the TIOT is available, jobname and stepname indicate the name of the affected job. WORKAREA=address indicates the address of the task recovery routine (TRR) work area. The areas dumped are NUC and RGN.

Problem Determination: For additional information, refer to message IEC999I in *VS2 System Messages.*

IEC999I IFG0RR0A,error-module,jobname,stepname, WORKAREA=address

Component: Open/Close/EOV (5752-SC1D1)

Issuing Module: IFG0RR0A - ESTAE

PLM: *OS/VS2 Open/Close/EOV Logic*

Explanation: An error has occurred during open, close, EOV, or DADSM processing. error-module indicates the name of the module in which the error occurred. If the TIOT is available, jobname and stepname indicate the name of the affected job. WORKAREA=address indicates the address of the task recovery routine (TRR) work area. The area dumped is RGN.

Problem Determination: For additional information, refer to message IEC999I in *VS2 System Messages.*

IEC999I IFG0RR0A,error-module,jobname,stepname, WORKAREA=address

Component: Open/Close/EOV (5752-SC1D1)

Issuing Module: IFG0RR0E - ESTAE

PLM: *OS/VS2 Open/Close/EOV Logic*

Explanation: An error has occurred during open, close, EOV, or DADSM processing. error-module indicates the name of the module in which the error occurred. If the TIOT is available, jobname and stepname indicate the name of the affected job. WORKAREA=address indicates the address of the task recovery routine (TRR) work area. The areas dumped are NUC and RGN.

Problem Determination: For additional information, refer to message IEC999I in *VS2 System Messages.*

IEC999I IFG0TC0A,subroutine,jobname,stepname,DEB ADDR=address
IEC999I IFG0TC4A,subroutine,jobname,stepname,DEB ADDR=address
IEC999I IFG0TC5A,subroutine,jobname,stepname,DEB ADDR=address

Component: Open/Close/EOV (5752-SC1D1)

Issuing Module: IFG0TC0A (Task Close) or IFG0TC4A (ESTAE)

PLM: *OS/VS2 Open/Close/EOV Logic*

Explanation: An error has occurred during task close processing. If the abend occurs in one of the subroutines called by task close, the task close ESTAE routine IFG0TC4A issues SDUMP. If the error occurs during mainline task close processing, IFG0TC0A issues SDUMP. More than one SDUMP may be issued when errors are encountered in the called subroutines. The failing subroutine is indicated by subroutine in the title. If the TIOT is available, jobname and stepname indicate the name of the affected job. DEB ADDR=address identifies the associated DEB control block. The areas dumped are NUC, RGN, CSA, and SQA.

Problem Determination: If a program check has occurred within task close processing, a software record is written to SYS1.LOGREC that includes the error module/routine information in the format of message IEC999I. For additional information, refer to message IEC999I in *VS2 System Messages.*

IEECB800 - TRACK COMMAND

Component: Master Scheduler Commands (5752-SC1B8)

Issuing Module: IEECB800 - DISPLAY TRACK Common Processor

PLM: *OS/VS2 System Logic Library*

Explanation: An error has occurred during TRACK command processing. Routine STAEXIT in IEECB800 issues the SDUMP macro. The areas dumped are PSA, NUC, LSQA, RGN, LPA, TRT, CSA, ALLPSA, and SQA.

Problem Determination: A software record is written to SYS1.LOGREC.

IEECB861 - FAILURE IN COMMAND xxxx

Component: Master Scheduler Commands (5752-SC1B8)

Issuing Module: IEECB860

PLM: *OS/VS2 System Logic Library*

Explanation: An error has occurred in the command processor while processing command xxxx. The areas dumped are PSA, NUC, LSQA, RGN, LPA, TRT, CSA, and SQA.

Problem Determination: A software record is written to SYS1.LOGREC.

IEECB906 SLIP ESTAE DUMP

Component: RTM - SLIP Command (5752-SC1CM)

Issuing Module: IEECB906 - ESTAE

PLM: *OS/VS2 System Logic Library*

Explanation: An error has occurred during SLIP or DISPLAY SLIP command processing.

Problem Determination: The SDWAVRA field in the SDWA contains the ESTAE parameter list. A software record is written to SYS1.LOGREC.

IEECB914 SLIP TSO COMM RTN ESTAE DUMP

Component: RTM - SLIP TSO Communication (5752-SC1CM)

Issuing Module: IEECB914

PLM: *OS/VS2 System Logic Library*

Explanation: An error has occurred while a SLIP command was being entered from a TSO terminal. The area dumped is SQA.

Problem Determination: A software record is written to SYS1.LOGREC. The SDWAVRA contains a copy of the ESTAE parameter list and a copy of the SLIP TSO element (STE) associated with the SLIP command.

IEEMB860

Component: Master Scheduler Commands (5752-SC1B8)

Issuing Module: IEEMB860 - Master Scheduler Region Initialization

PLM: *OS/VS2 System Initialization Logic*

Explanation: Either ESTAE or recovery termination setup failed. The error occurs if the LOAD macro (SVC 8) was unsuccessful, or master scheduler initialization failed. The areas dumped are PSA, NUC, LSQA, RGN, LPA TRT, CSA, and SQA.

Problem Determination: A software record is written to SYS1.LOGREC.

IEEMPDM - DUMP OF MAIN WORKAREA

Component: Reconfiguration (5752-SC1CZ)

Issuing Module: IEEMPDM

PLM: *OS/VS2 System Logic Library*

Explanation: An abend has occurred during DISPLAY MATRIX processing. The main work area of the command processor is dumped.

Problem Determination: A software record is written to SYS1.LOGREC and includes:

SDWAMODN	- IEEMPDM (module in error)
SDWACSCT	- IEEMPDM (CSECT in error)
SDWAREXN	- ESTAERTN (recovery routine)

IEEMPS03 - DUMP OF MAIN WORKAREA

Component: Reconfiguration (5752-SC1CZ)

Issuing Module: IEEMPS03

PLM: *OS/VS2 System Logic Library*

Explanation: An abend has occurred during QUIESCE command processing. The main work area for IEEMPS03 is dumped.

Problem Determination: A software record is written to SYS1.LOGREC and includes:

SDWAMODN	- IEEMPS03 (module in error)
SDWACSCT	- IEEMPS03 (CSECT in error)
SDWAREXN	- ESTAERTN (recovery routine)

IEEMPVST - DUMP OF MAIN WORK AREA

Component: Reconfiguration (5752-SC1CZ)

Issuing Module: IEEMPVST

PLM: *OS/VS2 System Logic Library*

Explanation: An abend has occurred during VARY STORAGE command processing. The main work area for IEEMPVST is dumped.

Problem Determination: A software record is written to SYS1.LOGREC and includes:

SDWAMODN	- IEEMPVST (module in error)
SDWACSCT	- IEEMPVST (CSECT in error)
SDWAREXN	- ESTAERTN (recovery routine)

IEEVIPL - ERROR IN MASTER SCHEDULER INITIALIZATION

Component: Master Scheduler Commands (5752-SC1B8)

Issuing Module: IEEVIPL - Master Scheduler Base Initialization

PLM: *OS/VS2 System Logic Library*

Explanation: During error recovery processing, SDUMP is issued if (1) STAE processing was unsuccessful, (2) the time of day clock failed, (3) a program check occurred, (4) the system restart key was pressed, or (5) control was returned because system initialization terminated. The areas dumped are PSA, NUC, LSQA, RGN, LPA, TRT, CSA, and SQA.

Problem Determination: A software record is written to SYS1.LOGREC.

IEEVLDWT ERROR

Component: Reconfiguration (5752-SC1CZ)

Issuing Module: IEEVLDWT

PLM: *OS/VS2 System Logic Library*

Explanation: An error has occurred during IEEVLDWT (load-wait) processing. The FRR routine in IEEVLDWT issues the SDUMP macro.

Problem Determination: The SDWAVRA field in the SDWA contains the FRR parameter list.

IEEVPTH - MAIN WORK AREA DUMP

Component: Reconfiguration (5752-SC1CZ)

Issuing Module: IEEVPTH

PLM: *OS/VS2 System Logic Library*

Explanation: An abend has occurred during VARY PATH command processing. The main work area of IEEVPTH is dumped.

Problem Determination: A software record is written to SYS1.LOGREC and includes:

SDWAMODN	- IEEVPTH (module in error)
SDWACSCT	- IEEVPTH (CSECT in error)
SDWAREXN	- VPTHESTA (recovery routine)

IEE5103D - FAILURE IN SVC 34/COMMAND xxxx

Component: Master Scheduler Commands (5752-SC1B8)

Issuing Module: IEE5103D - STAE

PLM: *OS/VS2 System Logic Library*

Explanation: The SVC 34 STAE routine IEE5103D issues SDUMP when (1) a system error or program check has occurred, or (2) the system restart key was pressed. The areas dumped are PSA, NUC, LSQA, RGN, LPA, TRT, CSA, and SQA.

Problem Determination: A software record is written to SYS1.LOGREC.

IEFIB620

Component: Initiator (5752-SC1B6)

Issuing Module: IEFIB620 - ESTAE

PLM: *OS/VS2 System Logic Library*

Explanation: During initiator processing, the ESTAE exit routine IEFIB620 issues SDUMP when (1) a system error or program check has occurred, or (2) the system restart key is pressed. The areas dumped are PSA, NUC, RGN, LPA, TRT, CSA, and SQA.

Problem Determination: A software record is written to SYS1.LOGREC.

IGCT0018,jobname,stepname

Component: SAM (5752-SC1D0)

Issuing Module: IGCT0018 - ESTAE

PLM: *OS/VS2 SAM Logic*

Explanation: During SVC 18 (BLDL or FIND) processing, the ESTAE routine IGCT0018 issues SDUMP if either (1) an abend has occurred, (2) a previous error recovery routine has failed, or (3) a system error has occurred. For a system error, message IEC909I is issued. The areas dumped are PSA, NUC, SQA, and RGN.

Problem Determination: A software record is written to SYS1.LOGREC. Also refer to message IEC909I in *VS2 System Messages.*

IGCT002D,jobname,stepname

Component: SAM (5752-SC1D0)

Issuing Module: IGCT002D - ESTAE

PLM: *OS/VS2 SAM Logic*

Explanation: During SVC 24 (DEVTYPE) processing, the ESTAE routine IGCT002D issues SDUMP if either (1) an abend has occurred, (2) a previous error recovery routine has failed, or (3) a system error has occurred. For a system error, message IEC912I is issued. The areas dumped are PSA, NUC, RGN, and SQA.

Problem Determination: A software record is written to SYS1.LOGREC. Also refer to message IEC912I in *VS2 System Messages.*

IGCT002E,jobname,stepname

Component: SAM (5752-SC1D0)

Issuing Module: IGCT002E - ESTAE

PLM: *OS/VS2 SAM Logic*

Explanation: During SVC 25 (track balance/overflow) processing, the ESTAE routine IGCT002E issues SDUMP if either (1) an abend has occurred, (2) a previous error recovery routine has failed, or (3) a system error has occurred. For a system error, message IEC915I is issued. The areas dumped are PSA, NUC, RGN, and SQA.

Problem Determination: A software record is written to SYS1.LOGREC. Also refer to message IEC915I in *VS2 System Messages.*

IGCT0021,jobname,stepname

Component: SAM (5752-SC1D0)

Issuing Module: IGCT0021 - ESTAE

PLM: *OS/VS2 SAM Logic*

Explanation: During SVC 21 (STOW) processing, the ESTAE routine IGCT002I issues SDUMP if either (1) an abend has occurred, (2) a previous error recovery routine has failed, or (3) a system error has occurred. For a system error, message IEC911I is issued. The areas dumped are PSA, NUC, RGN, and SQA.

Problem Determination: A software record is written to SYS1.LOGREC. Also refer to message IEC911I in *VS2 System Messages.*

IGCT005C,jobname,stepname

Component: DAM (5752-SC1D7)

Issuing Module: IGCT005C - ESTAE

PLM: *OS/VS2 BDAM Logic*

Explanation: During SVC 53 (exclusive control) processing, the ESTAE routine IGCT005C issues SDUMP if either (1) a previous error recovery routine has failed, or (2) a system error has occurred. Message IEC903I is issued. The areas dumped are PSA, NUC, RGN, and SQA.

Problem Determination: A software record is written to SYS1.LOGREC. Also refer to message IEC903I in *VS2 System Messages.*

IGCT005G,jobname,stepname

Component: DAM (5752-SC1D7)

Issuing Module: IGCT005G - ESTAE

PLM: *OS/VS2 BDAM Logic*

Explanation: During SVC 57 (FREEDBUF) processing, the ESTAE routine IGCT005G issues SDUMP if either (1) an error other than a program check occurred in the cleanup routine, (2) a previous error recovery has failed, or (3) a system error has occurred. For a system error, message IEC905I is issued. The areas dumped are PSA, NUC, RGN, and SQA.

Problem Determination: A software record is written to SYS1.LOGREC. Also refer to message IEC905I in *VS2 System Messages.*

IGCT006H,jobname,stepname,procstepname,744

Component: SAM (5752-SC1D0)

Issuing Module: IGCT006H - ESTAE

PLM: *OS/VS2 SAM Logic*

Explanation: During SVC 68 (SYNADAF/SYNADRLS) processing, the ESTAE routine IGCT006H issues SDUMP if either (1) an abend has occurred, (2) a previous error recovery routine has failed, or (3) a system error has occurred. For a system error, message IEC906I is issued. The areas dumped are PSA, NUC, RGN, and SQA.

Problem Determination: A software record is written to SYS1.LOGREC. Also refer to message IEC906I in *VS2 System Messages.*

IGCT0069,jobname,stepname

Component: SAM (5752-SC1D0)

Issuing Module: IGCT0069 - ESTAE

PLM: *OS/VS2 SAM Logic*

Explanation: During SVC 69 (BSP) processing, the ESTAE routine IGCT0069 issues SDUMP if either (1) an abend has occurred, (2) a previous error recovery routine has failed, or (3) a system error has occurred. For a system error, message IEC917I is issued. The areas dumped are PSA, NUC, RGN, and SQA.

Problem Determination: A software record is written to SYS1.LOGREC. Also refer to message IEC917I in *VS2 System Messages.*

IGCT010E,jobname,stepname

Component: SAM (5752-SC1D0)

Issuing Module: IGCT010E - ESTAE

PLM: *OS/VS2 SAM Logic*

Explanation: During SVC 105 (IMGLIB) processing, the ESTAE routine IGCT010E issues SDUMP if either (1) an abend has occurred, or (2) a previous error recovery routine has failed, or (3) a system error has occurred. For a system error, message IEC920I is issued. The areas dumped are PSA, NUC, RGN, and SQA.

Problem Determination: A software record written to SYS1.LOGREC. Also refer to message IEC920I in *VS2 System Messages.*

IGCT105C jobname,stepname

Component: DAM (5752-SC1D7)

Issuing Module: IGCT105C - ESTAE

PLM: *OS/VS2 BDAM Logic*

Explanation: During SVC 53 (exclusive control) processing, the ESTAE routine IGCT105C issues SDUMP if either (1) an abend has occurred, or (2) an error other than a program check has occurred in the cleanup routine for the first-level ESTAE routine. Message IEC903I is issued. The areas dumped are PSA, NUC, RGN, and SQA.

Problem Determination: A software record is written to SYS1.LOGREC. Also refer to message IEC903I in *VS2 System Messages.*

IGCT1081,jobname,stepname

Component: SAM (5752-SC1D0)

Issuing Module: IGCT1081 - ESTAE

PLM: *OS/VS2 SAM Logic*

Explanation: During SVC 81 (SETPRT) processing, the ESTAE routine IGCT1081 issues SDUMP if either (1) the DEB is invalid, (2) the FCB image is invalid, or (3) a system error has occurred. If the ESTAE routine was not entered directly from RTM, then message IEC918 is issued. The areas dumped are PSA, NUC, RGN, and SQA.

Problem Determination: A software record is written to SYS1.LOGREC. Also refer to message IEC918I in *VS2 System Messages.*

IGC0002F CATLG CTLR 3

Component: Catalog Controller 3 - CVOL Processor (5752-SC1DH)

Issuing Module: IGC0002F - ESTAE

PLM: *OS/VS2 CVOL Processor Logic*

Explanation: During SVC 26 (CATALOG/INDEX/LOCATE) processing, the catalog controller ESTAE routine IGC0002F issues SDUMP if any OCx abend occurs. The ESTAE routine frees storage resources so they are not lost to the system. The areas dumped are PSA, LSQA, and RGN.

IGFTMCHK MIH PROGRAM ERROR

Component: RMS (5752-SC1CE)

Issuing Module: IGFTMCHK - ESTAE

PLM: *OS/VS2 System Logic Library*

Explanation: An error has occurred during MIH (missing interruption handler) initialization or while MIH attempted to issue missing interrupt messages. The areas dumped are NUC, SQA, and TRT. Message IGF993E is also issued.

Problem Determination: If an SDWA is provided by RTM, this ESTAE exit writes a record to SYS1.LOGREC. The SDWA variable recording area contains the current level of the module. Also refer to message IGF993E in *VS2 System Messages.*

IKJEFLGM REQUEST

Component: TSO Scheduler (5752-SC1T4)

Issuing Module: IKJEFLGM (LOGON Message Module)

PLM: *OS/VS2 System Logic Library*

Explanation: An error has occurred during LOGON processing. SDUMP is taken if one of the following messages is issued:

IKJ56451 - results from an installation-exit error.
IKJ56452 - results from a system error.
IKJ600I - results from an I/O, OBTAIN, or OPEN error.
IKJ603I - results from an installation-exit abend.
IKJ608I - results from a TSO service routine error.

The areas dumped are NUC, LSQA, SWA, SQA, and LPA if TSO dump is requested.

Problem Determination: Refer to message IKJ600I, IKJ603I, and IKJ608I in *VS2 System Messages.*

IKTLTERM - I/O ERROR

Component: TSO/VTAM (5752-SC1T9)

Issuing Module: IKTLTERM

PLM: *OS/VS2 MVS VTIOC and TCAS Logic*

Explanation: TSO/VTAM has issued an abend due to an unrecoverable I/O error. The installation had requested the SVC dump by specifying the RPL sense code for the I/O error via the RCFBDUMP keyword in the TSOKEYxx member of SYS1.PARMLIB.

Problem Determination: Excessive line or hardware errors might be occurring. A software record is written to SYS1.LOGREC.

IOS - IECIHIO ERROR

Component: IOS (5752-SC1C3)

Issuing Module: IECIHIO

PLM: *OS/VS2 I/O Supervisor Logic*

Explanation: An error has occurred during IECIHIO (resident HALT) processing. The FRR routine HIOFRR issues SDUMP. The areas dumped are TRT and SQA.

Problem Determination: A software record is written to SYS1.LOGREC. The dump buffer contains a copy of the UCB for which the HDV instruction had been requested. If the FRR was entered due to a program check, the most likely cause of the error is a bug in IECIHIO.

IOS - IECIOSCN ERROR

Component: IOS (5752-SC1C3)

Issuing Module: IECIOSCN - FRR

PLM: *OS/VS2 I/O Supervisor Logic*

Explanation: An error has occurred during IECIOSCN (basic IOS) processing. The FRR routine IECIOSCN issues SDUMP. The areas dumped are SQA, TRT, and PSA. The address space dumped is the address space associated with the I/O request being processed. This address space might not match the current ASID in the associated LOGREC entry.

Problem Determination: A software record is written to SYS1.LOGREC. The SDWAVRA contains the IOQ and UCB (when appropriate). The SDUMP buffer contains the IRT, UCB, IOQ, IOSB, and LCH (when appropriate).

IOS - IECVERPL ERROR

Component: IOS (5752-SC1C3)

Issuing Module: IECVERPL

PLM: *OS/VS2 I/O Supervisor Logic*

Explanation: The ESTAE routine of the ERP loader issues SDUMP if a program check occurs in either (1) IECVERPL, or (2) an ERP (error recovery procedure) or service routine that IECVERPL has loaded. The areas dumped are PSA, SQA, LPA, CSA, NUC, TRT, and the LSQA and SWA for the current address space.

Problem Determination: A software record is written to SYS1.LOGREC. The SRB and IOSB are copied to the SDUMP buffer. The SIRB chained off of the RCT TCB contains the name of the ERP module that is currently executing.

IOS - IECVSMGR ERROR

Component: IOS (5752-SC1C3)

Issuing Module: IECVSMGR - FRR

PLM: *OS/VS2 I/O Supervisor Logic*

Explanation: An error has occurred during IECVSMGR (storage manager) processing. The FRR routine IECVSMFR issues SDUMP. In most cases, the error is a C0D abend that results when a caller of IECVSMGR attempts to (1) get an IOS block that is already allocated, or (2) free an IOS block that is not in use. The areas dumped are SQA, NUC, and TRT.

Problem Determination: A software record is written to SYS1.LOGREC. Included in the SDWAVRA is the six-word FRR area, the storage pool headers and free queue headers. The registers identify the type of request and resulting error. The SDUMP buffer contains the storage pool headers, free queue headers, and registers.

IOS - POST STATUS ERROR

Component: IOS (5752-SC1C3)

Issuing Module: IECVPST

PLM: *OS/VS2 I/O Supervisor Logic*

Explanation: The IOS post status FRR has received control because of a program check. The error might have occurred in IECVPST or in an exit (channel end, abnormal end, PCI, attention, etc.). The areas dumped are PSA, SQA, NUC, and TRT. The SDUMP buffer contains the IOSB and the PSA highest lock held indicator. The SDWAVRA also contains the IOSB.

Problem Determination: Examine the IOSB for bad exit pointers IOSB+X'38' and IOSB+X'44'. Determine where in the module the error occurred. The IOSB and the UCB are valuable sources of information about exits via the device types. A software record is written to SYS1.LOGREC.

IOS - SMGR SQA EXHAUSTED

Component: IOS (5752-SC1C3)

Issuing Module: IECVSMGR - FRR

PLM: *OS/VS2 I/O Supervisor Logic*

Explanation: Module IECVSMGR was not able to obtain a 2K or 4K block of SQA for IOS control blocks. The areas dumped are SQA, NUC, and TRT.

Problem Determination: A software record is written to SYS1.LOGREC. Included in the SDWAVRA is the six-word FRR area, pool headers and free queue headers. The SDUMP buffer contains the storage pool headers, free queue headers, and registers.

ISAM INTRFC,OPEN,IDA0192I,IDAICIA1,**AUDIT NOT STARTED**
ISAM INTRFC,OPEN,IDA0192I,IDAICIA1,**IDA0192I IN CONTROL**
ISAM INTRFC,CLOSE,IDA0200S,IDAICIA1,**AUDIT UNAVAILABLE**
ISAM INTRFC,CLOSE,IDA0200S,IDAICIA1,**IDAIIPM1 IN CONTROL**
ISAM INTRFC,CLOSE,IDA0200S,IDAICIA1,**IDA0200S IN CONTROL**

Component: VSAM - ISAM-Interface (5752-SC1DE)

Issuing Module: IDAICIA1 - ESTAE

PLM: *OS/VS2 Virtual Storage Access Method (VSAM) Logic*

Explanation: An error has occurred during the opening or closing of a DCB via the ISAM interface. Module IDAICIA1 (ISAM-interface data-set management recovery routine) issues SDUMP. One of the five titles appears depending on the error and whether open or close was in control at the time of error.

Problem Determination: Depending on the error, some or all of the following areas are dumped: the dump list itself, user's DCB, protected copy of the user's DCB, OPEN/CLOSE work area, recovery work area, IICB, ACB, EXLST, buffers and message area.

ISTAPCES - ACF/VTAM PSS ESTAE ROUTINE

Component: ACF/VTAM (5665-28001)

Issuing Module: ISTAPCES - PSS ESTAE

PLM: *ACF/VTAM Diagnosis Reference*

Explanation: An abend has occurred while an ACF/VTAM task was processing and an ACF/VTAM IRB was active. The areas dumped are SQA, NUC, RGN, LPA, TRT, ALLPSA, and CSA.

Problem Determination: A software record is written to SYS1.LOGREC. For a description of the CRA fields recorded, refer to *ACF/VTAM Data Areas.*

ISTAPCFR - ACF/VTAM PSS FUNCTIONAL RECOVERY

Component: ACF/VTAM (5665-28001)

Issuing Module: ISTAPCFR - PSS FRR

PLM: *ACF/VTAM Diagnosis Reference*

Explanation: An abend has occurred while ACF/VTAM was processing and running under an SRB. The areas dumped are ALLPSA, CSA, NUC, SQA, TRT, LPA, and RGN.

Problem Determination: A software record is written to SYS1.LOGREC. For a description of the CRA fields recorded, refer to *ACF/VTAM Data Areas.*

ISTAPCMT - ACF/VTAM ABEND IN MEMORY TERMINATION

Component: ACF/VTAM (5665-28001)

Issuing Module: ISTAPCMT

PLM: *ACF/VTAM Diagnosis Reference*

Explanation: An abend has occurred while the ACF/VTAM memory termination resource manager was processing. ACF/VTAM attempts minimal cleanup so that ACF/VTAM can be restarted. However, CSA storage might not be usable until the next IPL. The areas dumped are SQA, NUC, RGN, LPA, LSQA, TRT, ALLPSA, and CSA.

Problem Determination: A software record is written to SYS1.LOGREC. For a description of the CRA fields recorded, refer to *ACF/VTAM Data Areas.*

ISTAPC61 - VTAM IRB ABEND

Component: VTAM or ACF/VTAM (5752-SC123)

Issuing Module: ISTAPC61 - VTAM ESTAE

PLM: *OS/VS2 MVS VTAM Logic* or *ACF/VTAM Logic*

Explanation: An abend has occurred while a VTAM task was processing and a VTAM IRB was active. The areas dumped are SQA, NUC, RGN, LPA, TRT, ALLPSA, and CSA.

Problem Determination: A software record is written to SYS1.LOGREC. For a description of the CRA fields recorded, refer to the *VTAM System Programmer's Guide* or *ACF/VTAM System Programmer's Guide.*

ISTAPC62 - VTAM SRB ABEND

Component: VTAM or ACF/VTAM (5752-SC123)

Issuing Module: ISTAPC62 - VTAM FRR

PLM: *OS/VS2 MVS VTAM Logic* or *ACF/VTAM Logic*

Explanation: An abend has occurred while VTAM was processing and running under an SRB. The areas dumped are ALLPSA, CSA, NUC, SQA, TRT, LPA, and RGN.

Problem Determination: A software record is written to SYS1.LOGREC. For a description of the CRA fields recorded, refer to the *VTAM System Programmer's Guide* or *ACF/VTAM System Programmer's Guide.*

ISTAPC66 - VTAM ABEND

Component: VTAM or ACF/VTAM (5752-SC123)

Issuing Module: ISTAPC66 - VTAM FRR

PLM: *OS/VS2 MVS VTAM Logic* or *ACF/VTAM Logic*

Explanation: An abend has occurred during ISTAPC25, ISTAPC55, ISTAPC59, or ISTAICIR processing. The FRR routine (ISTAPC66) issues the SDUMP macro and also indicates that an abend is in progress for each ACDEB and PST that are not associated with the VTAM jobstep TCB or a subtask of the VTAM jobstep. The areas dumped are SQA, NUC, RGN, LSQA, LPA, TRT, ALLPSA, CSA, and SWA.

Problem Determination: A software record is written to SYS1.LOGREC. For a description of the CRA fields recorded, refer to the *VTAM System Programmer's Guide* or *ACF/VTAM System Programmer's Guide.*

ISTATM00 - VTAM TERMINATION SUBTASK ESTAE

Component: VTAM or ACF/VTAM (5752-SC123)

Issuing Module: ISTATM00 - ESTAE

PLM: *OS/VS2 MVS VTAM Logic* or *ACF/VTAM Logic*

Explanation: An abend has occurred while the VTAM termination subtask (VTT) was processing. The ESTAE routine ISTATM00 issues the SDUMP macro for abends that occur during VTAM processing (but not for abends that occur during application processing). The areas dumped are SQA, LSQA, TRT, ALLPSA, CSA, and RGN.

Problem Determination: A software record is written to SYS1.LOGREC. For a description of the CRA fields recorded, refer to the *VTAM System Programmer's Guide* or *ACF/VTAM System Programmer's Guide.*

ISTINCST - VTAM STAE EXIT

Component: VTAM or ACF/VTAM (5752-SC123)

Issuing Module: ISTINCST - ESTAE

PLM: *OS/VS2 MVS VTAM Logic* or *ACF/VTAM Logic*

Explanation: An abend has occurred while the VTAM jobstep task was processing. The areas dumped are SQA, NUC, RGN, LPA, TRT, ALLPSA, and CSA.

Problem Determination: None.

ISTORMMG - ACF/VTAM FRR DUMP

Component: ACF/VTAM (5665-28001)

Issuing Module: ISTORMMG

PLM: *ACF/VTAM Diagnosis Reference*

Explanation: An abend has occurred while ISTORMMG was running in SRB mode. ISTORMMG frees CSA storage and recovery is attempted by zeroing the CSA to-be-freed queue (ATCORTBF). The areas dumped are SQA, NUC, RGN, LPA, ALLPSA, and CSA.

Problem Determination: A software record is written to SYS1.LOGREC. For a description of the CRA fields recorded, refer to *ACF/VTAM Data Areas.*

ISTRAMA2 DUMP FOR TRAMA6

Component: VTAM or ACF/VTAM (5752-SC123)

Issuing Module: ISTRAMA2

PLM: *OS/VS2 MVS VTAM Logic* or *ACF/VTAM Logic*

Explanation: An error has occurred in ISTRAMA5, which is an internal routine to ISTRAMA2 (address space termination resource manager). ISTRAMA5 receives control from ISTRAMA2 via an SRB for the purpose of scheduling TPEND exits. ISTRAMA5's FRR (ISTRAMA6) issues the SDUMP macro. The areas dumped are SQA, ALLPSA, NUC, RGN, LPA, TRT, and CSA.

Problem Determination: A software record is written to SYS1.LOGREC.

ISTRAMA2 DUMP FOR TRAMST

Component: VTAM or ACF/VTAM (5752-SC123)

Issuing Module: ISTRAMA2

PLM: *OS/VS2 MVS VTAM Logic* or *ACF/VTAM Logic*

Explanation: An abend has occurred in ISTRAMUE which is an internal routine to ISTRAMA2 (address space termination resource manager). ISTRAMUE receives control from ISTRAMA2 via an IRB for the purpose of scheduling user TPEND exits. ISTRAMUE's ESTAE routine (ISTRAMST) issues the SDUMP macro. The areas dumped are SQA, ALLPSA, NUC, RGN, LPA, TRT, and CSA.

Problem Determination: A software record is written to SYS1.LOGREC.

ISTRAMA3 - VTAM TASK TERM FAILS

Component: VTAM or ACF/VTAM (5752-SC123)

Issuing Module: ISTRAMA3 - ESTAE

PLM: *OS/VS2 MVS VTAM Logic* or *ACF/VTAM Logic*

Explanation: An abend has occurred while ISTRAMA1 (task termination resource manager) was cleaning up VTAM resources for a terminating user task. ISTRAMA1's ESTAE routine (ISTRAMA3) issues the SDUMP macro. The areas dumped are SQA, LSQA, TRT, ALLPSA, and CSA.

Problem Determination: A software record is written to SYS1.LOGREC. For a description of the CRA fields recorded, refer to the *VTAM System Programmer's Guide* or *ACF/VTAM System Programmer's Guide.*

ISTRAMA4 - VTAM MEMORY TERMINATION ESTAE

Component: VTAM or ACF/VTAM (5752-SC123)

Issuing Module: ISTRAMA4 - ESTAE

PLM: *OS/VS2 MVS VTAM Logic* or *ACF/VTAM Logic*

Explanation: An abend has occurred while ISTRAMA2 (address space termination resource manager) was processing. ISTRAMA2's ESTAE routine (ISTRAMA4) issues the SDUMP macro. The areas dumped are SQA, LSQA, TRT, ALLPSA, and CSA.

Problem Determination: A software record is written to SYS1.LOGREC. For a description of the CRA fields recorded, refer to the *VTAM System Programmer's Guide* or *ACF/VTAM System Programmer's Guide.*

JES2 FSI ERROR. CODE=cde RC=rc (text)

Component: JES2 (5752-SC1BH)

Issuing Module: HASPFSSM

PLM: *OS/VS2 MVS JES2 Logic*

Explanation: A catastrophic error has occurred in the JES2 functional subsystem interface (FSI) support routines (HASPFSSM) for which JES2 issued a $ERROR macro. HASPFSSM was operating in a functional subsystem (FSS) address space. JES2 terminates the FSS address space.

The HASPFSSM error routine FSMCATER issued the SDUMP macro. The areas dumped are ALLPSA, RGN, TRT, SQA, CSA, LPA, SWA, and LSQA

This dump is associated with JES2 message $HASP750 and system abend code 02C.

Problem Determination: See message $HASP750 in *System Messages* and abend code 02C in *System Codes* for information on this error.

JES3 LOCATE SUBTASK ABEND

Component: JES3 (5752-SC1BA)

Issuing Module: IATLVLC

PLM: *OS/VS2 MVS JES3 Logic*

Explanation: An abend has occurred during IATLVLC (locate subtask) processing. The ESTAE routine established by IATLVLC is given control to examine the function control table (FCT) active at the time of termination to determine which function or DSP has failed. The areas dumped are SQA, CSA, PSA, RGN, LPA, and TRT.

Problem Determination: A software record is written to SYS1.LOGREC.

JES3 SNA FRR IATSNDF

Component: JES3 (5752-SC1BA)

Issuing Module: IATSNDF - FRR

PLM: *OS/VS2 MVS JES3 Logic*

Explanation: The FRR routine (IATSNDF) handles abends that occur during SNA RJP processing under an SRB. Each time that the FRR is entered, an SVC dump is taken. Therefore, control of dumping is dependent on the FRR's recursion control to prevent more than two retry failures. (A dump is taken for every retry failure.) The areas dumped are SQA, ALLPSA, NUC, LSQA, RGN, TRT, CSA, and LPA.

Problem Determination: The SDWA is logged after it is updated with LCB data if available. A software record is written to SYS1.LOGREC.

JOB=jobname hh:mm:ss yy.ddd DUMP BY IGG0CLA9 - VSAM CATALOG MANAGEMENT

Component: VSAM - Catalog Management (5752-SC1DE)

Issuing Module: IGG0CLA9 - ESTAE

PLM: *OS/VS2 Catalog Management Logic*

Explanation: An abend has occurred during catalog management processing. The ESTAE routine IGG0CLA9 issues the SDUMP macro, frees storage resources, and backs-out partially defined catalog entries in the VSAM catalogs. Message IEC338I is also issued if a validity check failed on a user field parameter list (FPL) or a catalog parameter list (CPL).

Problem Determination: A software record is written to SYS1.LOGREC. The SDWAVRA includes the following:

Offset	Length	Meaning
0(0)	8	c'IGG0CLA9'
8(8)	3	Entry point address of IGG0CLA9.
11(B)	8	Name of the last routine called.
19(13)	3	Entry point address of the last routine called.
22(16)	8	Name of the calling routine.
30(1E)	3	Entry point address of the calling routine.
33(21)	4	c'CPL='
37(25)	28	User's CPL.

LOAD MOD-IEFW21SD EXIT RTN-xxxxxxxx

Component: Allocation (5752-SC1B4)

Issuing Module: IEFAB4ED - Allocation Common ESTAE Exit

PLM: *OS/VS2 System Logic Library*

Explanation: An error has occurred during allocation processing. The ESTAE routine IEFAB4ED performs general recovery processing and issues the SDUMP macro if no SDWA exists. If an SDWA exists, additional checks on the error are made. The SDUMP macro is then issued if the error was not a user error and is either (1) a program check, or (2) the restart key was pressed, or (3) an abend occurred and there was no percolation or if there was percolation, it was via FRR recovery. The areas dumped are: NUC, RGN, LPA, TRT, CSA, ALLPSA, and SQA.

In the title, xxxxxxxx indicates the name of the recovery routine that receives control after IEFAB4ED processing in order to release the resources for the function. The possible exit routine names and the functional areas of the allocation component that they represent are:

Exit	Functional Area
IEFAB4DD	Initiator/job or step unallocation interface
IEFAB4DE	Common unallocation
IEFAB4EA	JFCB housekeeping
IEFAB4E2	Volume mount and verify
IEFAB4E4	Initiator/job or step allocation interface
IEFAB4E7	Group lock/unlock
IEFAB4E8	Common allocation
IEFAB402	Dynamic allocation (SVC 99)

Problem Determination: A software record is written to SYS1.LOGREC.

PGM CHECK IN IEAVGCAS

Component: VSM - Memory Create (5752-SC1CH)

Issuing Module: IEAVCARR - FRR

PLM: *OS/VS2 System Logic Library*

Explanation: A program check has occurred in IEAVGCAS while recovering from a previous error in IEAVGCAS during address space create, free address space, or task termination. The areas dumped are PSA and LSQA.

Problem Determination: A software record is written to SYS1.LOGREC and includes:

SDWAMODN	- Module name of the routine in error.
SDWACSCT	- CSECT name of the routine in error.
SDWAREXN	- FRR module name.

PGM CHECK IN IEAVPRT0

Component: VSM - GETPART/FREEPART (5752-SC1CH)

Issuing Module: IEAVGPRR - FRR

PLM: *OS/VS2 System Logic Library*

Explanation: A program check has occurred in IEAVPRT0 while recovering from a previous error in IEAVPRT0 or in post processing by XMPOST. The areas dumped are PSA, LSQA, and NUC.

Problem Determination: A software record is written to SYS1.LOGREC and includes:

SDWAMODN	- Module name of the routine in error.
SDWACSCT	- CSECT name of the routine in error.
SDWAREXN	- FRR module name.

RACF INITIALIZATION FAILURE

Component: RACF (5752-XXH00)

Issuing Module: ICHSEC02 - ESTAE

PLM: *OS/VS2 MVS Resource Access Control Facility (RACF): Program Logic Manual*

Explanation: An abend has occurred during ICHSEC00 (RACF initialization) processing. The areas dumped are SQA, CSA, NUC, and RGN.

Problem Determination: A software record is written to SYS1.LOGREC and includes:

SDWAMODN	- ICHSEC00 (module in error)
SDWAREXN	- ICHSEC02 (recovery routine)
SDWACID	- XXH00
SDWACSCT	- ICHSEC00
SDWAEAS	- 1 if SDUMP is taken by ICHSEC00
SDWAREQ	- 0 if SDUMP is taken by ICHSEC00

RCT DUMPING LSQA

Component: Region Control Task (5752-SC1CU)

Issuing Module: IEAVAR00 - ESTAE

PLM: *OS/VS2 System Logic Library*

Explanation: The ESTAE routine in IEAVAR00 issues the SDUMP macro when a previous error recovery routine could not diagnose the error and either (1) RCT's RB was in control, (2) an error occurred in the previous recovery exit, (3) an RCT FRR routine requested the dump, or (4) retry recursion has occurred.

Problem Determination: A software record is written to SYS1.LOGREC. Pay particular attention to the cause of error flags and the RCT flags in the SDWAVRA. Additional footprints and data are available in the RCTD of the dumped storage.

RECORD PERMANENT ERROR

Component: RTM - RECORD Macro (5752-SC1CM)

Issuing Module: IEAVTRET - ESTAE

PLM: *OS/VS2 System Logic Library*

Explanation: An operation exception (0C1) has occurred while IEAVTRET (RECORD macro processing) was in control and processing a "temporary error" type but not while IEAVTRET was attempting to post the user's ECB. The recording function is turned off and message IEA896I is issued, stating that the recording function is not active. A return code of 20 is issued for following RECORD macro requests. The areas dumped are SQA, NUC, LPA, and RGN.

Problem Determination: A software record is written to SYS1.LOGREC. (Refer also to RECORD TEMPORARY ERROR.)

RECORD TEMPORARY ERROR

Component: RTM - RECORD Macro (5752-SC1CM)

Issuing Module: IEAVTRET

PLM: *OS/VS2 System Logic Library*

Explanation: An abend has occurred while IEAVTRET (RECORD macro processing) was in control and the RCB buffer was not being manipulated by the requesting routine. This abend is not a "permanent error" type. If the error was not an operation exception (0C1), an interface is established from the ESTAE parameter list and retry is initiated at the point where the task wakes up from the wait state. The areas dumped are SQA, NUC, LPA, and RGN.

Problem Determination: A software record is written to SYS1.LOGREC. (Refer also to RECORD PERMANENT ERROR.)

REQUESTOR = xxxxxxxx,ISSUER = ISGCRCV,COMPID = SCSDS, COMPON = GRS

Component: Global Resource Serialization

Issuing Module: ISGCRCV - ESTAE

PLM: *OS/VS2 MVS Global Resource Serialization Logic*

Explanation: An error has occurred while one of the following command processing modules was processing. The field xxxxxxxx in the dump title indicates the failing module.

ISGCMDI	ISGCQMRG	ISGNGRSP
ISGCMDR	ISGCQSC	
ISGCPRG	ISGCRST	

The ESTAE module ISGCRCV issues the SDUMP macro. If the error occurred in ISGCMDI, the master address space is dumped. For errors in the other modules, the global resource serialization address space is dumped.

Problem Determination: A software record is written to SYS1.LOGREC.

RESOURCE MANAGER

Component: Initiator (5752-SC1B6)

Issuing Module: IEFISEXR - ESTAE

PLM: *OS/VS2 System Logic Library*

Explanation: A program check or a restart interruption has occurred in the initiator or a subsystem interface resource manager. The ESTAE routine IEFISEXR issues the SDUMP macro. The areas dumped are SQA, PSA, LSQA, RGN, LPA, TRT, CSA, and NUC.

Problem Determination: A software record is written to SYS1.LOGREC.

RESTART INTERRUPT IN CONVERTER**IEFNB9CR**

Component: Converter (5752-SC1B9)

Issuing Module: IEFNB9CR - Converter Recovery Routine

PLM: None (refer to microfiche)

Explanation: A restart interruption has occurred during converter processing. The ESTAE routine IEFNB9CR issues the SDUMP macro. The areas dumped are SQA, LSQA, SWA, RGN, and LPA.

Problem Determination: A software record is written to SYS1.LOGREC.

RESTART INTERRUPT IN INTERPRETER**IEFNB9IR**

Component: Interpreter (5752-SC1B9)

Issuing Module: IEFNB9IR - Interpreter Recovery Routine

PLM: None (refer to microfiche)

Explanation: A restart interruption has occurred during interpreter processing. The recovery routine IEFBN9IR issues the SDUMP macro. The areas dumped are SQA, LSQA, SWA, RGN, and NUC.

Problem Determination: A software record is written to SYS1.LOGREC.

RMF DEA

Component: RMF (5752-XY400)

Issuing Module: ERBMFDEA - ESTAE

PLM: *OS/VS2 MVS Resource Measurement Facility (RMF) Version 2 Program Logic Manual*

Explanation: An error has occurred during RMF processing. The data control ESTAE routine ERBMFDEA issues the SDUMP macro. The areas dumped are SQA, CSA, RGN, LPA, ALLPSA, NUC, and LSQA.

Problem Determination: A software record is written to SYS1.LOGREC. The RMF control block STGST (pointed to by field CVTMFCTL in the CVT) and the trace table are helpful in determining the cause of the error.

RMF SDE

Component: RMF (5752-XY400)

Issuing Module: ERBMFSDE - ESTAE

PLM: *OS/VS2 MVS Resource Measurement Facility (RMF) Version 2 Program Logic Manual*

Explanation: An error has occurred during RMF processing. The MFSTART ESTAE routine ERBMFSDE issues the SDUMP macro. The areas dumped are SQA, CSA, RGN, LPA, ALLPSA, NUC, and LSQA.

Problem Determination: A software record is written to SYS1.LOGREC. The RMF control block STGST (pointed to by field CVTMFCTL in the CVT) and the trace table are helpful in determining the cause of the error.

SCHEDULER JCL FACILITY FAILURE,ABEND= ,COMPON=SCHR-SJF, COMPID=BB131,ISSUER=IEFSJCNL

Component: Scheduler Services (5752-BB131)

Issuing Module: IEFSJCNL

PLM: *System Logic Library*

Explanation: An abend has occurred while the scheduler JCL facility was processing. The areas dumped are SWA, CSA, LPA, and LSQA.

Problem Determination: A software record is written to SYSl.LOGREC. The SDWAVRA contains footprint bits that indicate the execution path of the scheduler JCL facility.

SDUMP - IGG0CLCA CVOL CATALOG MANAGEMENT

Component: Catalog Controller 3 - CVOL Processor (5752-SC1DH)

Issuing Module: IGG0CLCA - ESTAE

PLM: *OS/VS2 CVOL Processor Logic*

Explanation: An abend has occurred in the first CSECT of the CVOL processor mapper. The ESTAE routine IGG0CLCA issues the SDUMP macro, and dequeues the PCCB and DSNAME resources. The areas dumped are PSA, LSQA, LPA, and RGN.

SDUMP - IGG0CLCD - CVOL CATALOG MANAGEMENT

Component: Catalog Controller 3 - CVOL Processor (5752-SC1DH)

Issuing Module: IGG0CLCD - ESTAE

PLM: *OS/VS2 CVOL Processor Logic*

Explanation: An abend has occurred while IGG0CLCD was building catalog entries for CVOLs. The ESTAE routine IGG0CLCD issues the SDUMP macro and frees resources. The areas dumped are PSA, LSQA, LPA, and RGN.

SET SMF COMMAND - IEEMB835

Component: SMF (5752-SC100)

Issuing Module: IEEMB835 - ESTAE

PLM: *OS/VS2 System Logic Library*

Explanation: An abend has occurred during SET SMF processing. In addition to the SVC dump, message IEE965I is also issued.

Problem Determination: A software record is written to SYS1.LOGREC. For additional information, refer to message IEE965I in *VS2 System Messages.*

SLIP ID=xxxx

Component: RTM - SLIP Processor (5752-SC1CM)

Issuing Module: IEAVTSLP

PLM: *OS/VS2 System Logic Library*

Explanation: A SLIP trap has matched and the action specified on the trap definition is ACTION=SVCD. IEAVTSLP issues the SDUMP macro. The areas dumped are defaulted or specified on the SDATA, LIST, SUMLIST, and ASIDLST keywords of the SLIP command. ID=xxxx is the SLIP trap identifier.

Problem Determination: None - the system is functioning as requested.

SMF ABEND, ERRMOD=IEEMB829/IEFU29, RECVRMOD=IEEMB825

Component: SMF (5752-SC100)

Issuing Module: IEEMB825

PLM: *OS/VS2 System Logic Library*

Explanation: An abend has occurred during task mode SMF writer processing. If ERRMOD=IEFU29, the user exit was in control at the time of the error. The areas dumped are PSA, NUC, RGN, CSA, LPA, SQA, and SUMDUMP.

Problem Determination: A software record is written to SYS1.LOGREC.

SMF ABEND, ERRMOD=xxxxxxxx, RECVRMOD=IEEBM830

Component: SMF (5752-SC100)

Issuing Module: IEEMB830

PLM: *OS/VS2 System Logic Library*

Explanation: An abend has occurred during SMF record processing. If xxxxxxxx is IEFU83 or IEFU84, the error occurred during processing by the installation exit. Otherwise, xxxxxxxx is IEEMB830. The areas dumped are PSA, NUC, RGN, CSA, LPA, SQA, and SUMDUMP.

Problem Determination: A software record is written to SYS1.LOGREC. The SDWAVRA field contains the output of the queue verify routine IEAVEQV3, if it was invoked to verify the BQE chain.

SMF ABENDED, ERRMOD=IEEMB834, RECVMOD=IEEMB834

Component: SMF (5752-SC100)

Issuing Module: IEEMB834 - FRR

PLM: *OS/VS2 System Logic Library*

Explanation: An abend has occurred during the SRB mode processing which writes to the SMF recording data set. The areas dumped are PSA, NUC, RGN, CSA, LPA, SQA, and SUMDUMP.

Problem Determination: A software record is written to SYS1.LOGREC. The FRR parameter area contains footprints and is mapped by the structure FRRPARM.

SMF ERRMOD=IEASMFSP, RECVRMOD=IEASMFSP

Component: SMF (5752-SC100)

Issuing Module: IEASMFSP - FRR

PLM: *OS/VS2 System Logic Library*

Explanation: An abend occurred while SMF was attempting to suspend or reset a caller of SMF because of a buffer shortage. The areas dumped are PSA, NUC, RGN, CSA, LPA, SQA, and SUMDUMP.

Problem Determination: The FRR parameter area indicates the function being performed. It is mapped by the structure FRRPRM. A software record is written to SYS1.LOGREC.

SMF INITIALIZATION, RECVRMOD=IEEMB827

Component: SMF (5752-SC100)

Issuing Module: IEEMB827

PLM: *OS/VS2 System Initialization Logic*

Explanation: An abend has occurred during SMF writer initialization. If the completion code is X'253', the abend occurred during initialization of the writer task (as opposed to the initialization task). In this case, the initialization task abended itself due to the failure of the SMF writer task initialization. The areas dumped are PSA, NUC, RGN, CSA, LPA, SQA, and SUMDUMP.

Problem Determination: A software record is written to SYS1.LOGREC. For a description of code 253, refer to *VS2 System Codes*.

SMF TIMER - IEEMB839

Component: SMF (5752-SC102)

Issuing Module: IEEMB839 - FRR

PLM: *OS/VS2 System Logic Library*

Explanation: An error has occurred in the SMF timer module while the dispatcher lock was held.

Problem Determination: A software record is written to SYS1.LOGREC.

SRM - IRARMSRV 55F ABEND DURING XMPOST

Component: SRM (5752-SC1CX)

Issuing Module: IRARMSRV

PLM: *OS/VS2 System Logic Library*

Explanation: An error has occurred during the cross-address-space post function. The post was requested by module IRARMEVT to notify the issuer of a REQSWAP or TRANSWAP that the swap is complete or that the address space became non-swappable before the swap could be initiated. The address space being posted is terminated with a 55F completion code. The areas dumped are PSA, SQA, and TRT.

Problem Determination: The terminating address space's ASCB and OUCB are copied into the SDUMP buffer pointed to be CVTSDBF. The buffer fields are mapped by SDMPBUFF in module IRARMSRV.

SRM RECOVERY ENTERED,COMPON=SRM,COMPID=SC1CX, ISSUER=IRARMERR

Component: SRM (5752-SC1CX)

Issuing Module: IRARMERR - FRR

PLM: *OS/VS2 System Logic Library*

Explanation: An error has occurred during SRM processing. Depending on the error, retry of the failing function is attempted or the error is percolated. The current address space is dumped.

Problem Determination: A software record is written to SYS1.LOGREC. The variable recording area in the SDWA contains a message that gives an offset into the data module IRARMCNS. This offset is the location of the control block for the SRM routine in control when the error occurred.

SSICS ABEND 6FB

Component: JES3 (5752-SC1BA)

Issuing Module: IATSSCM

PLM: *OS/VS2 MVS JES3 Logic*

Explanation: A system error has occurred while IATSSCM (subsystem communication scheduler) was processing in an address space other than JES3's address space. Abend 6FB is issued. The areas dumped are PSA, RGN, LPA, TRT, CSA, NUC, and SQA.

Problem Determination: A software record is written to SYS1.LOGREC. For a description of code 6FB, refer to *VS2 System Codes.*

SSICS ESTAE-IATSSCM

Component: JES3 (5752-SC1BA)

Issuing Module: IATSSCM

PLM: *OS/VS2 MVS JES3 Logic*

Explanation: IATSSCM (subsystem communication scheduler) was not able to reduce the system impact caused by communication failures for the second time. JES3 is put in the IATSSCM quiesce condition. The areas dumped are PSA, RGN, LPA, TRT, CSA, NUC, and SQA.

Problem Determination: A software record is written to SYS1.LOGREC.

STARTED TASK CONTROL

Component: Master Scheduler Commands (5752-SC1B8)

Issuing Module: IEESB665

PLM: *OS/VS2 System Logic Library*

Explanation: The recovery exit routine IEESB665 schedules a retry for STC in the event of an error (if information is available for a retry). If an SDWA is provided, IEESB665 issues the SDUMP macro. The areas dumped are SQA, PSA, LSQA, RGN, LPA, TRT, CSA, and NUC.

Problem Determination: A software record is written to SYS1.LOGREC.

STARTED TASK CONTROL

Component: Master Scheduler Commands (5752-SC1B8)

Issuing Module: IEESB670

PLM: *OS/VS2 System Logic Library*

Explanation: The recovery exit routine IEESB670 schedules a retry of the job scheduling subroutine (IEESB605). If an SDWA is provided, IEESB670 issues the SDUMP macro. The areas dumped are SQA, PSA, LSQA, RGN, LPA, TRT, CSA, and NUC.

Problem Determination: A software record is written to SYS1.LOGREC.

STORAGE DUMP TAKEN AT ENTRY TO IEEMB812 ESTAE EXIT

Component: SRM (5752-SC1CX)

Issuing Module: IEEMB812 - SRM SET Processor

PLM: *OS/VS2 System Logic Library*

Explanation: An error has occurred during SRM processing of a SET command. The new tables are freed and the old controls remain in effect. The SET command is retried. If the error recurs, IEEMB812 percolates the error.

Problem Determination: A software record is written to SYS1.LOGREC.

STORAGE DUMP TAKEN AT ENTRY TO IRARMERR

Component: SRM (5752-SC1CX)

Issuing Module: IRARMERR - FRR

PLM: *OS/VS2 System Logic Library*

Explanation: An error has occurred during SRM processing. Depending on the error, retry of the failing function is attempted or the error is percolated. The current address space is dumped.

Problem Determination: A software record is written to SYS1.LOGREC. The variable recording area in the SDWA contains a message that gives an offset into the data module IRARMCNS. This offset is the location of the control block for the SRM routine in control when the error occurred.

SWA CREATE

Component: SWA Manager (5752-SC1B5)

Issuing Module: IEFIB645

PLM: *OS/VS2 System Logic Library*

Explanation: A program check or a restart interruption has occurred during interpreter, restart, warmstart, or SWA create processing. The recovery routine IEFIB645 issues the SDUMP macro. The areas dumped are SQA, PSA, LSQA, RGN, LPA, TRT, CSA, and NUC.

Problem Determination: A software record is written to SYS1.LOGREC.

SYSTEM LOG DUMP

Component: Master Scheduler Commands (5752-SC1B8)

Issuing Module: IEEMB803

PLM: *OS/VS2 System Logic Library*

Explanation: An error has occurred during IEEMB803 (system log initialization/writer) processing. The areas dumped are PSA, NUC, LSQA, RGN, and CSA.

Problem Determination: A software record is written to SYS1.LOGREC.

SYSTEM LOG DUMP

Component: Master Scheduler Commands (5752-SC1B8)

Issuing Module: IEEMB806 - ESTAE

PLM: *OS/VS2 System Logic Library*

Explanation: An abend has occurred during system log task processing. The areas dumped are PSA, NUC, LSQA, RGN, CSA, and SQA.

Problem Determination: A software record is written to SYS1.LOGREC.

SYSTEM LOG SVC DUMP

Component: Master Scheduler Commands (5752-SC1B8)

Issuing Module: IEEMB804

PLM: *OS/VS2 System Logic Library*

Explanation: The data of a non-authorized user does not exist in his own storage area. The user has failed the authorization check and FETCH/PROTECT validity test, or an abend occurred during processing.

Procedure DUMPRTN issues the SDUMP macro. The areas dumped are LSQA, RGN, and CSA.

Problem Determination: A software record is written to SYS1.LOGREC.

TCAS DUMP

Component: TSO/VTAM (5752-SC1T9)

Issuing Module: IKTCAS52

PLM: *OS/VS2 MVS VTIOC and TCAS Logic*

Explanation: TCAS (terminal control address space) is terminating because (1) the operator requested termination via the STOP command, or (2) a program check has occurred. The dump is taken as a result of the operator responding DUMP to message IKT012D.

TIMER FRR DUMP

Component: Timer Supervisor (5752-SC1CV)

Issuing Module: IEAVRTI1 - FRR

PLM: *OS/VS2 System Logic Library*

Explanation: An error has occurred during timer supervision processing. In module IEAVRTI1, subroutine IEAVRSWR (entry point IEAVRFRR) issues the SDUMP macro. The areas dumped are PSA, NUC, SQA, TRT, and LSQA for the current address space.

Problem Determination: A software record is written to SYS1.LOGREC. The data area TFRRPARM (mapped in IEAVRTI1) is saved in the SDWAVRA. TFRRPARM contains indicators that tell the type of processing taking place and the locks held at the time of the error, as well as the results of the TQE validation process. The SDWA also includes:

SDWAMODN	- Name of the module in control at the time of the error
SDWACSCT	- Name of the CSECT in control at the time of the error
SDWAREXN	- IEAVRFRR (recovery routine)

TSO OUTPUT CP ESTAE

Component: TSO Scheduler (5752-SC1T4)

Issuing Module: IKJCT460 - ESTAE

PLM: *OS/VS2 TSO Command Processor Logic, Volume IV*

Explanation: An abend error or a DETACH with STAE has occurred during TSO command processing. The ESTAE exit routine IKJCT460 receives control from the supervisor and issues the SDUMP macro for (1) x0A abends (except 80A), and (2) all other abends except for a DETACH with STAE, the abends B37, D37, E37, 913, 622, and 222. The areas dumped are LSQA, SWA, NUC, SQA, and LPA.

Problem Determination: For some errors, a software record is written to SYS1.LOGREC. If a workarea was obtained, SDWAREXN='IKJCT460' and the SDWA is based on register 1 for the SETRP macro.

TSO SDUMP FROM IKJEFT05 - THE TMP ESTAE ROUTINE

Component: TSO Scheduler (5665-28502)

Issuing Module: IKJEFT05

PLM: *TSO/E Terminal Monitor Program and Service Routines Logic*

Explanation: The TMP ESTAE exit routine, IKJEFT05, issues the SDUMP macro on the first occurrence of an error in a TMP module. The areas dumped are NUC, LSQA, RGN, TRT, and SQA.

Problem Determination: A software record is written to SYS1.LOGREC.

TSOLOGON ESTAE

Component: TSO Scheduler (5752-SC1T4)

Issuing Module: IKJEFLS - ESTAE

PLM: *OS/VS2 System Logic Library*

Explanation: A program check or PSW restart interruption has occurred during TSO logon initialization or scheduling. The ESTAE routine IKJEFLS issues the SDUMP macro. The areas dumped are LSQA, SWA, NUC, SQA, and LPA.

Problem Determination: A software record is written to SYS1.LOGREC. Register 1 points to the SDWA (if one exists) and includes:

SDWAMODN	- IKJEFLA - Name of the abending load module.
SDWAREXN	- IKJEFLS - Name of the ESTAE routine.

TSOLOGON ESTAI

Component: TSO Scheduler (5752-SC1T4)

Issuing Module: IKJEFLGB - Prompter's ESTAI

PLM: *OS/VS2 System Logic Library*

Explanation: During logon processing, the ESTAI routine IKJEFLGB issued the SDUMP macro for (1) a program check, (2) a PSW restart condition, or (3) an abend in IKJEFLD (logon pre-prompt exit). The areas dumped are LSQA, SWA, NUC, SQA, and LPA.

Problem Determination: A software record is written to SYS1.LOGREC. If a SDWA exists:

- Register 1 contains the address of the STAE work area.
- Register 14 contains the return address.

If a SDWA does not exist:

- Register 1 contains the abend code.
- Register 2 contains a pointer to the LWA.
- Register 14 contains the return address.

VIRTUAL FETCH JPQ FAILURE

Component: Virtual Fetch (5752-SC1CJ)

Issuing Modules: CSVVFIND - issued by ESTAE CSVVFFES or CSVVFGET - issued by FRR CSVVFRR

PLM: *OS/VS2 System Logic Library*

Explanation: One of the following occurred:

- During build or find processing, an abend occurred while CSVVFIND was attempting to remove a CDE from the job pack queue for a module that previously abended under the current TCB. ESTAE routine CSVVFFES in CSVVFIND requests an SVC dump. Register 8 points to the VFWK containing the CDE at the time of the error.

- During get processing, an abend occurred while CSVVFGET was attempting to remove a CDE from the job pack queue. FRR routine CSVVFRR in CSVVFGET requests an SVC dump. Field LWKVFWK of the local work area (LWK) points to the VFWK containing the CDE at the time of the error.

The areas dumped are LWK, VFPM, VFCB, and LSQA (which contains the ASXB, VFVT, VFWKs, and job pack queue).

Problem Determination: A software record is written to SYS1.LOGREC. The following fields in the SDWA are filled in: SDWAMODN, SDWACSCT, SDWAREXN, SDWAMDAT, SDWAMVRS, SDWACID, and SDWARRL.

VIRTUAL FETCH LOCAL CONTROL BLOCK FAILURE

Component: Virtual Fetch (5752-SC1CJ)

Issuing Modules: CSVVFIND - issued by ESTAE CSVVFFES or CSVVFGET - issued by ESTAE CSVVGFES or CSVVFGET - issued by FRR CSVVFRR

PLM: *OS/VS2 System Logic Library*

Explanation: One of the following occurred:

- An abend occurred while CSVVFIND (build or find processing) or CSVVFGET (get processing) was searching a queue of VFWKs. ESTAE routine CSVVFFES in CSVVFIND or CSVVFGES in CSVVFGET requests an SVC dump.

- An abend occurred while CSVVFGES (ESTAE for CSVVFGET) was attempting to free storage (by calling CSVVFRM) related to a VFWK. FRR routine CSVVFRR in CSVVFGET requests an SVC dump.

The areas dumped are LWK, VFPM, VFCB (if it exists), and LSQA (which contains the ASXB, VFVT, VFWKs, and job pack queue).

Problem Determination: A software record is written to SYS1.LOGREC. The following fields in the SDWA are filled in: SDWAMODN, SDWACSCT, SDWAREXN, SDWAMDAT, SDWAMVRS, SDWACID, and SDWARRL.

Field ASXBVFVT points to the VFVT. VFVTHASH is the start of a 31-way hash table, which contains pointers to the VFWK collision queues. The VFWKs are chained in single-threaded queues using field VFWKSYNP as the chain pointer.

If the error occurred while searching a queue of VFWKs, a probable cause might be that a chain pointer in the VFWK was overwritten with invalid data. In this case, flag VFVTVFUP is set off after requesting a SVC dump to prevent virtual fetch from being used again in this address space.

If the error occurred while freeing module and VCBs storage, the VFWK might have been overwritten causing CSVVFRM to pass invalid addresses to FREEMAN. In this case, flag VFWKBADV is set on after requesting an SVC dump to prevent the VFWK from being used again.

VIRTUAL FETCH RETRY INHIBITED

Component: Virtual Fetch (5752-SC1CJ)

Issuing Modules: CSVVFIND - issued by ESTAE CSVVFFES or CSVVFGET - issued by ESTAE CSVVFGES or CSVVFGET - issued by FRR CSVVFRR

PLM: *OS/VS2 System Logic Library*

Explanation: One of the following occurred:

- The FRR CSVVFRR (in CSVVFGET) was entered with flag SDWACLUP on and the abend was not a cancel abend (X'x22').

- One of the recovery routines (CSVVFFES, CSVVFGES, or CSVVFRR) attempted to retry, but was reentered before retry processing was established. Retry is not attempted again. CSVVFFES and CSVVFGES set flag LWKRECUR on in the LWK. CSVVFRR sets flag LWKFRRR on in the LWK.

The areas dumped are LWK, VFPM, VFCB (if it exists), and LSQA (which contains the ASXB, VFVT, VFWKs, and job pack queue).

Problem Determination: A software record is written to SYS1.LOGREC. The following fields are filled in: SDWAMODN, SDWACSCT, SDWAREXN, SDWAMDAT, SDWAMVRS, SDWACID, and SDWARRL.

VSAM CHECKPOINT (IDA0xxxx) or VSAM RESTART (IDA0xxxx) with one of the following:

MACHINE CHECK
PROGRAM CHECK LOCATION=xxxxxx
RESTART KEY DEPRESSED
PAGING ERROR
ABEND Sxxx, Uxxxx, REGISTER 15=xxxxxxxx

Component: VSAM - Checkpoint/Restart (5752-SC1DE)

Issuing Module: IDACKRA1 - ESTAE

PLM: *OS/VS2 Virtual Storage Access Method (VSAM) Logic*

Explanation: An error has occurred during VSAM checkpoint or restart processing. The ESTAE routine issues the SDUMP macro. The title on the dump depends on the type of error and whether checkpoint or restart was in control at the time of error. The areas dumped are SQA, LPA, and user's region.

Operator- and Caller-Defined SVC Dump Titles

This topic provides diagnostic information for the modules that initiate SVC dumps via the SDUMP macro but where the dump title is defined by the system operator or the caller of SVC dump.

variable title - supplied by the system operator

Component: Master Scheduler Commands (5752-SC1B8)

Issuing Module: IEECB866 - Console Dump

PLM: *OS/VS2 System Logic Library*

Explanation: The system operator has issued the DUMP command and specified the title of the SVC dump on the command.

Problem Determination: None.

variable title - supplied by the system operator

| **Component:** JES2 (5752-SC1BH)

| **Issuing Module:** HASPTERM or HASPRAS

| **PLM:** *JES2 Logic.*

Explanation: The system operator has entered an SVC dump title in response to the $HASP098 message. This title overrides the default dump title. The areas dumped are PSA, NUC, RGN, TRT, SQA, CSA, LPA, and SWA.

| **Problem Determination:** For information on the error, refer to messages $HASP098 and $HASP095 in *JES2 Messages.*

variable title - supplied by the caller

Component: Task Manager (5752-SC1CL)

Issuing Module: IEAVECH0 - CHAP Service Routine

PLM: *OS/VS2 System Logic Library*

Explanation: An error has been detected in the TCB dispatching queue for the current address space by the TCB queue verification routine. The FRR (IGC044R1) issues the SDUMP macro. The areas dumped are LSQA, SQA, and TRT.

Problem Determination: A software record is written to SYS1.LOGREC. The SDWAVRA contains the associated TCB-queue-verification output.

SVC Dumps Without Titles

This topic provides diagnostic information for the modules that initiate SVC dumps but where no titles are supplied on the SDUMP macro.

no title

Component: Catalog Controller 3 - CVOL Processor (5752-SC1DH)

Issuing Module: IGG0CLCB - ESTAE

PLM: *OS/VS2 CVOL Processor Logic*

Explanation: An abend has occurred during the processing of a GENERIC LOCATE request for a CVOL. All storage resources are freed and the CVOL processor SDUMP routine issues the SDUMP macro. The area dumped is LPA.

no title

Component: IOS (5752-SC1C3)

Issuing Module: IGC0001F

PLM: *OS/VS2 I/O Supervisor Logic*

Explanation: An error has occurred during IGC0001F (nonresident purge) processing. The areas dumped are PSA, SQA, NUC, and TRT.

Problem Determination: A software record is written to SYS1.LOGREC. The SDUMP buffer contains a copy of the purge work area and optionally:

- A copy of a UCB (if the IOSUCB lock is held)
- A copy of a LCH (if the IOSLCH lock is held)
- Copies of IOQEs on the LCH for which the lock is held.

The most likely cause of the error is an invalid IOSB address.

no title

Component: JES3 (5752-SC1BA)

Issuing Module: IATIIII (IATYIIW work area)

PLM: *OS/VS2 MVS JES3 Logic*

Explanation: An abend has occurred during interpreter/initiator (IATIIII) processing. The ESTAE routine established by IATIIII is given control to examine the function control table (FCT) active at the time of termination to determine which function or DSP failed. The areas dumped are PSA, RGN, LPA, TRT, and CSA.

Problem Determination: A software record is written to SYS1.LOGREC. Register 9 points to a work area containing formatted messages.

no title

Component: RTM (5752-SC1CM)

Issuing Module: IEAVTRTE

PLM: *OS/VS2 System Logic Library*

Explanation: The SDUMP macro is issued by the RTM2 exit processor (IEAVTRTE) as a result of an error during abnormal termination of a subtask of the jobstep task. The jobstep task is terminated with a D0D completion code. The current task is set non-dispatchable in order to wait for the jobstep task to detach it. If another failure occurs, the address space is terminated. The areas dumped are SQA, LSQA, LPA, TRT, CSA, and SWA.

Problem Determination: The RTM2WAs addressed by the current TCB contain the most pertinent information.

no title

Component: VTAM or ACF/VTAM (5752-SC123)

Issuing Module: ISTZRM01 - FRR

PLM: VTAM - refer to the microfiche; for ACF/VTAM refer to *ACF/VTAM Logic*

Explanation: An abend has occurred during VTAM processing while the global UCB spin lock was held. The areas dumped are ALLPSA, CSA, LSQA, SQA, TRT, LPA, NUC, and SWA.

Problem Determination: A software record is written to SYS1.LOGREC. The SQA includes the UCB and the ICNCB or LDNCB control block.

Module to SVC Dump Title Cross-Reference

This topic lists, in alphameric order, the MVS modules that issue the SDUMP macro and provides the titles of the SVC dumps specified by the modules on the SDUMP macro.

Issuing Module	**Dump Title**
AHLGTFI	AHL007I GTF TERMINATING ON ERROR CONDITION
AHLMCER	ERROR IN MODULE AHLMCER
AHLREADR	DUMP OF AHLREADR
AHLSBUF	DUMP OF GTF MODULE AHLSBLOK
AHLSETEV	ERROR IN AHLSETEV
AHLTMON	GTF TERMINATING ON ERROR CONDITION
AHLWTASK	DUMP OF GTF MODULE AHLWTASK
AHLWTO	DUMP BY/(OF) MODULE xxxxxxxx
CSVVFCRE	COMPON=PROGRAM-MANAGER...CSVVFCES...
CSVVFCRE	COMPON=PROGRAM-MANAGER...CSVVFCFR...
CSVVFGET	VIRTUAL FETCH JPQ FAILURE
	VIRTUAL FETCH LOCAL CONTROL BLOCK...
	VIRTUAL FETCH RETRY INHIBITED
CSVVFIND	VIRTUAL FETCH JPQ FAILURE
	VIRTUAL FETCH LOCAL CONTROL BLOCK...
	VIRTUAL FETCH RETRY INHIBITED
ERBMFDEA	RMF DEA
ERBMFEEQ	COMPON=RMF-ENQ EVENT HANDLER...
ERBMFSDE	RMF SDE
HASPCKPT	DUMP OF JES2 DATA...
HASPFSSM	JES2 FSI ERROR...
HASPTERM	ABEND code AT hhhhhhhh (nnnnnn) + X'nnnn' cc-
or	HASPDUMP SUBSYS=ssss...
HASPRAS	title - supplied by the system operator
IATABN0	IAT3702 dspname (ddd) ABENDED/FAILED...
IATDMFR	ERROR IN IATSIDMO FOR SYSOUT DATA SET
	IAT1801 ERROR IN IADMDKT - IATYISR POSSIBLY LOST
IATIIII	none
IATIISB	IAT4830 IATIISB MASTERTASK ABEND
	IAT4831 IATIISB SUBTASK ABEND
IATLVLC	JES3 LOCATE SUBTASK ABEND
IATSIJS	IATSIJS JSESEXIT
IATSNDF	JES3 SNA FRR IATSNDF
IATSNLS	IATSNLS - ESTAE EXIT
IATSSCM	IATSSCM READ-END FAILURE
	SSICS ABEND 6FB
	SSICS ESTAE-IATSSCM
IATSSRE	COMPON=JES3 SUBSYS COMMUNIC...
IATSSXM	COMPON=JES3 SUBSYS COMMUNIC...
ICBRECRD	ICBRECRD RECORDING FAILED
ICBVPR00	ICB425I ABEND IN PROCESS - MSVC TASK (nnnnnnn)

ICHRST00	ICHRST00 - RACF SVCS, ABEND CODE=xxx,...
ICHSEC02	RACF INITIALIZATION FAILURE

ICTMCS01	ICTMCS01, CRYPTOGRAPHY INITIALIZATION
ICTMKG00	ICTMKG00, KEY GENERATOR PROGRAM
ICTMKG01	ICTMKG01 HANDLE SYSIN MODULE
ICTMKM01	ICTMKM01, START CRYPTOGRAPHY COMMAND
ICTMKM04	ICTMKM04 - KEY MANAGER
ICTMSM07	ICTMSM07 - ICTMSM07 - CIPHER DUMP
ICTMSM08	ICTMSM07 - ICTMSM08 TRNSKEY DUMP
ICTMSM09	ICTMSM07 - ICTMSM09 EMK DUMP

IDACKRA1	VSAM CHECKPOINT (IDA0xxxx)...
	VSAM RESTART (IDA0xxxx)...
IDAICIA1	ISAM INTRFC, OPEN/CLOSE, IDA0192I/IDA0200S...
IDA019SB	IDA019SB:IDA121F7 - ABEND FROM BUILD IDACPA
IDA019S2	FIOD:IDA019S2 - ABEND FROM FIOD FRR
IDA0200T	IEC251I, VSAM GSR FORCE DLVRP DUMP DATA
IDA121A2	ABP:IDA121A2 - ABEND FROM ABP FRR
IDA121A3	ABP:IDA121A3 - ABEND FROM NORMAL END FRR
IDA121A4	ABP:IDA121A4 - ABEND FROM ABNORMAL END FRR

IEASMFSP	SMF ERRMOD=IEASMFSP, RECVRMOD=IEASMFSP
IEAVAD01	ERROR DURING SNAP
	ERROR IN QMNGRIO PROCESSING
	FAILURE DURING SNAP RECOVERY
IEAVAR00	RCT DUMPING LSQA
IEAVCARR	PGM CHECK IN IEAVGCAS
IEAVEAC0	IEAVEAC0 IEAVEAC0 IEAVEAC3
IEAVECH0	variable title - supplied by caller
IEAVEMCR	IEAVEMCR - MEMORY CREATE ABNORMAL TERMINATION
IEAVEMDL	IEAVEMDL - MEMORY DELETE ABNORMAL TERMINATION
IEAVEMRQ	IEAVEMRQ UNEXPECTED ABEND
	IEAVEMRQ - UNEXPECTED ABEND WITH DISPATCHER LOCK
IEAVETCL	COMPON=SUPV CNTL,COMPID=SC1C5
IEAVGFRR	ERROR IN GETMAIN/FREEMAIN
IEAVGPRR	PGM CHECK IN IEAVPRT0
IEAVMFRR	COMPON=COMMTASK...ISSUER=IEAVMFRR...
IEAVN700	COMPON=COMMTASK...ISSUER=IEAVN700...
IEAVN701	COMPON=COMMTASK...ISSUER=IEAVN701...
IEAVRCV	ERROR IN REAL STORAGE MANAGER
	ERROR IN REAL STORAGE MANAGER FRR
IEAVRTI1	TIMER FRR DUMP
IEAVSTAA	COMPON=COMMTASK...ISSUER=IEAVSTAA...
IEAVSY50	IEAVSY50 IGC001 IGC002 XMPOST FAIL - NO ERRET
IEAVTABD	ABDUMP ERROR
	ENQUEUE FAILURE IN ABDUMP
IEAVTGLB	ABEND IN IEAVTGLB
IEAVTJBN	ABEND IN IEAVTJBN
IEAVTLCL	ABEND IN IEAVTLCL
IEAVTRET	RECORD PERMANENT ERROR
	RECORD TEMPORARY ERROR
IEAVTRTE	no title
IEAVTRT2	IEAVTRT2 - UNRECOVERABLE ABEND FAILURE
IEAVTSLP	SLIP ID=xxxx
IEAVTSLR	ERROR IN IEAVTSLP
IEAVXPCR	ABEND=aaa,REASON=xxyy,GPRrr=zzzzzzzz,...
IEAVXSTK	ABEND=aaa,COMPON=PC/AUTH-PCLINK UNSTACK...

IECIHIO	IOS - IECIHIO ERROR
IECIOSCN	IOS - IECIOSCN ERROR
IECVBRSV	COMPON=IOS...ISSUER=IECVBRSV

IECVDAVV	DAVV ERROR
IECVDPTH	COMPON=IOS,COMPID=SC1C3,ISSUER=IECVDPTH
IECVERPL	IOS - IECVERPL ERROR
IECVESIO	IECVESIO ERROR
IECVFCHN	COMPON=IOS...ISSUER=IECVFCHN,FCHNFRR
IECVFDEV	COMPON=IOS...ISSUER=IECVFDEV,FDEVFRR...
IECVGENA	COMPON=IOS...ISSUER=IECVGENA
IECVHREC	COMPON=IOS...ISSUER=IECVHREC
IECVIOPM	IECVIOPM PROGRAM ERROR
IECVIOSI	COMPON=IOS,COMPID=SC1C3,ISSUER=IECVIOSI
IECVIRST	IECVIRST ERROR
IECVPST	IOS - POST STATUS ERROR
IECVRDIO	IECVRDIO ERROR
IECVRRSV	COMPON=IOS...ISSUER=IECVRRSV
IECVSMGR	IOS - IECVSMGR ERROR IOS - SMGR SQA EXHAUSTED

IEECB800	IEECB800 - TRACK COMMAND
IEECB860	IEECB861 - FAILURE IN COMMAND xxxx
IEECB862	COMPON=M S CMNDS,COMPID=SC1B8,...
IEECB866	variable title - supplied by the system operator
IEECB906	IEECB906 SLIP ESTAE DUMP
IEECB914	IEECB914 SLIP TSO COMM RTN ESTAE DUMP
IEEMB803	SYSTEM LOG DUMP
IEEMB804	SYSTEM LOG SVC DUMP
IEEMB806	SYSTEM LOG DUMP
IEEMB812	STORAGE DUMP TAKEN AT ENTRY TO IEEMB812 ESTAE
IEEMB816	COMPID=SC1B8,xxx ABEND IN MASTER TR...
IEEMB825	SMF ABEND, ERRMOD=IEEMB829/IEFU29,...
IEEMB827	SMF INITIALIZATION, RECVRMOD=IEEMB827
IEEMB830	SMF ABEND, ERRMOD=xxxxxxxx, RECVRMOD=IEEMB830
IEEMB834	SMF ABENDED, ERRMOD=IEEMB834, RECVMOD=IEEMB834
IEEMB835	SET SMF COMMAND - IEEMB835
IEEMB836	ABEND IN SMF INTERVAL PROCESSING - ROUTINE...
IEEMB839	SMF TIMER - IEEMB839
IEEMB860	IEEMB860
IEEMB881	COMPON=M S CMNDS,COMPID=SC1B8,...
IEEMB883	COMPON=M S CMNDS,COMPID=SC1B8,...
IEEMB887	COMPON=MS CMNDS,COMPID=SC1B8,ISSUER=IEEMB887...
IEEMPDM	IEEMPDM - DUMP OF MAIN WORKAREA
IEEMPS03	IEEMPS03 - DUMP OF MAIN WORKAREA
IEEMPVST	IEEMPVST - DUMP OF MAIN WORKAREA
IEESB665	STARTED TASK CONTROL
IEESB670	STARTED TASK CONTROL
IEEVIPL	IEEVIPL - ERROR IN MASTER SCHEDULER INIT
IEEVLDWT	IEEVLDWT ERROR
IEEVPTH	IEEVPTH - MAIN WORKAREA DUMP
IEEVWAIT	COMPON=MSTR-WAIT...ISSUER=IEEVWAIT...
IEE24110	D U,,ALLOC ABEND
IEE5103D	IEE5103D - FAILURE IN SVC34/COMMAND xxxx

IEFAB4ED	LOAD MOD-IEFW21SD EXIT RTN-xxxxxxxx
IEFAB4E6	ABEND=aaa,COMPON=ALLOC,COMPID=SC1B4, ERRMOD=...
IEFCMAUT	COMMON AUTHORIZATION ROUTINE ...
IEFENFFX	ENF ABEND ERRORMOD=IEFENFFX
IEFAB4ED	LOAD MOD-IEFW21SD EXIT RTN-xxxxxxxx
IEFENFFX	ENF ABEND ERRORMOD=IEFENFFX
IEFENFNM	ENF ABEND ERRORMOD=IEFENFNM
IEFENFWT	EVENT NOTIFICATION FACILITY ERROR
IEFIB620	IEFIB620
IEFIB645	SWA CREATE
IEFISEXR	RESOURCE MANAGER
IEFJRASP	ERROR IN BROADCAST FUNCTION,ABEND=aaa,...
IEFJSBLD	ERROR IN SUBSYSTEM SERVICE RTN, COMPON=INIT-SSI

IEFJSIN2	ERROR IN SUBSYSTEM INITIALIZATION, COMPON = INIT-SSI...
IEFNB9CR	ABEND = aaa,COMPON = CONVERTER...
	RESTART INTERRUPT IN CONVERTER**IEFNB9CR**
IEFNB9IR	ABEND = aaa,COMPON = INTERPRETER...
	RESTART INTERRUPT IN INTERPRETER**IEFNB9IR**
IEFSJCNL	SCHEDULER JCL FACILITY...
IEFXB609	ERROR DURING RESTART PROCESSING - IEFXB609

IFG0RR0A	IEC999I IFG0RR0A,error-module,jobname,...
IFG0RR0E	IEC999I IFG0RR0A,error-module,jobname,...
IFG0RR0F	IEC999I IFG0RR0A,IFG0RR0F,jobname,stepname...
IFG0TC0A	IEC999I IFG0TC0A/4A/5A,subroutine,jobname...

IGCT0018	IGCT0018,jobname,stepname
IGCT002D	IGCT002D,jobname,stepname
IGCT002E	IGCT002E,jobname,stepname
IGCT0021	IGCT0021,jobname,stepname
IGCT005C	IGCT005C,jobname,stepname
IGCT005G	IGCT005G,jobname,stepname
IGCT006H	IGCT006H,jobname,stepname,procstepname,744
IGCT0069	IGCT0069,jobname,stepname,
IGCT010E	IGCT010E,jobname,stepname
IGCT105C	IGCT105C,jobname,stepname
IGCT1081	IGCT1081,jobname,stepname
IGC0001F	no title
IGC0002F	IGC0002F CATLG CTLR 3
IGC121	ABP:IGC121 - ABEND FROM SIOD FRR

IGFDE1	DYNAMIC DEVICE RECOVERY ERROR DUMP
IGFTMCHK	IGFTMCHK MIH PROGRAM ERROR

IGG0CLA9	JOB = jobname hh:mm:ss yy.ddd DUMP BY IGG0CLA9...
IGG0CLCA	SDUMP - IGG0CLCA CVOL CATALOG MANAGEMENT
IGG0CLCB	no title
IGG0CLCD	SDUMP - IGG0CLCD - CVOL CATALOG MANAGEMENT

IKJCT460	TSO OUTPUT CP ESTAE
IKJEFLGB	TSOLOGON ESTAI
IKJEFLGM	IKJEFLGM REQUEST
IKJEFLS	TSOLOGON ESTAE
IKJEFT05	TSO SDUMP FROM IKJEFT05 ...

IKTCAS52	TCAS DUMP
IKTLTERM	IKTLTERM - I/O ERROR

IOSVRSUM	COMPON = IOS...ISSUER = IOSVRSUM...

IRARMERR	SRM RECOVERY ENTERED,COMPON = SRM
IRARMSRV	SRM - IRARMSRV 55F ABEND DURING XMPOST

ISGBERCV	COMPON = GRS-RING-PROC...ISSUER = ISGBERCV
ISGBFRCV	COMPON = GRS-RING-PROC...ISSUER = ISGBFRCV
ISGCRCV	REQUESTOR = xxx,ISSUER = ISGCRV...
ISGCRET0	COMPON = GRS-COMMANDS...ISSUER = ISGCRET0...

ISGCRET1	COMPON = GRS-COMMANDS...ISSUER = ISGCRET1...
ISGDSNAP	COMPON = GRS...ISSUER = ISGDSNRV
ISGGFRR0	COMPON = GRS...ISSUER = ISGGFRR0
ISGJENF0	COMPON = GRS-CTC-DRIVER...ISSUER = ISGJENF0
ISGJRCV	COMPON = GRS-CTC-DRIVER...ISSUER = ISGJRCV
ISGQSCNR	COMPON = GRS-QUEUE SCANNING...ISSUER = ISGQSCNR
ISGSMI	COMPON = GRS,COMPID = SCSDS,ISSUER = ISGSMIFR

ISTAPCES	ISTAPCES - ACF/VTAM PSS ESTAE ROUTINE
ISTAPCFR	ISTAPCFR - ACF/VTAM PSS FUNCTIONAL RECOVERY
ISTAPCMT	ISTAPCMT - ACF/VTAM ABEND IN MEMORY TERMINATION
ISTAPC61	ISTAPC61 - VTAM IRB ABEND
ISTAPC62	ISTAPC62 - VTAM SRB ABEND
ISTAPC66	ISTAPC66 - VTAM ABEND
ISTATM00	ISTATM00 - VTAM TERMINATION SUBTASK ESTAE
ISTINCST	ISTINCST - VTAM STAE EXIT
ISTORMMG	ISTORMMG - ACF/VTAM FRR DUMP
ISTRAMA2	ISTRAMA2 DUMP FOR TRAMA6 ISTRAMA2 DUMP FOR TRAMST
ISTRAMA3	ISTRAMA3 - VTAM TASK TERM FAILS
ISTRAMA4	ISTRAMA4 - VTAM MEMORY TERMINATION ESTAE
ISTZRM01	no title

Appendix D. Abbreviations

ABP	- Actual block processor
ACA	- ASM control area
ACB	- Access method control block
ACE	- ASM control element
ACF/VTAM	- Advanced Communications Function for VTAM (Program Product)
ACP	- Automatic command processing
ACR	- Alternate CPU recovery
ACT	- Account control table
ADA	- Automatic data area
ADB	- Allocation descriptor block
AFQ	- Available frame queue
AIA	- ASM I/O request area
ALCWA	- Allocation work area
ALPAQ	- Active link pack area queue
AMB	- Access method block
AMBL	- AMB list
AMCBS	- Access method control block structure
AMDSB	- Access method data statistics block
AP	- Attached processor
APF	- Authorized program facility
APG	- Automatic priority group
ASCB	- Address space control block
ASID	- Address space identification
ASM	- Auxiliary storage manager
ASMHD	- Auxiliary storage management header
ASMVT	- ASM vector table
ASPCT	- Auxiliary storage page correspondence table
ASST	- Address space sector table
ASTE	- Address space second table entry
ASVT	- Address space vector table
ASXB	- Address space extension block
AT	- Authorization table
ATA	- ASM tracking area
AVT	- TCAM address vector table
AX	- Authorization index
AXAT	- Authorization index allocation table
BASEA	- Master scheduler resident data area
BPCB	- Buffer pool control block
BUFC	- Buffer control area
CA	- Control area or channel adapter
CAW	- Channel address word
CAXWA	- Catalog ACB extended work area
CCA	- Catalog communications area
CCH	- Channel check handler
CCW	- Channel command word
CDE	- Contents directory entry
CEPL	- Command ESTAE parameter list
CFQ	- Common frame queue
CHAP	- Change priority
CI	- Control interval
CIDF	- Control interval definition field
CKB	- Checkpoint buffer
CMB	- Console message buffer

CML	- Cross memory lock
CMS	- Cross memory services or catalog management services
CMSWA	- CMS work area
CPA	- Channel program area
CPAB	- Cell pool anchor block
CPB	- Channel program block
CPPL	- Command processor parameter list
CPU	- Central processing unit
CPUID	- CPU identification
CQE	- Console queue element
CRA	- Component recovery area
CRB	- Command request block
CRWA	- Command recovery work area
CSA	- Common service area
CSCB	- Command scheduling control block
CSD	- Common system data area
CTGPL	- Catalog parameter list
CVT	- Communications vector table
CXSA	- Communications extended save area

DADSM	Direct access device space management
DAIT	- Display allocation index table
DALT	- Display allocation lookup table
DAM	- Direct access method
DAT	- Dynamic address translation
DAVV	- Direct access volume verification
DCB	- Data control block
DCM	- Display control module
DCT	- Device control table
DDR	- Dynamic device reconfiguration
DDRCOM	- Dynamic device reconfiguration communication table
DE	- Directory entry
DEB	- Data extent block
DECB	- Data event control block
DEPL	- SDUMP ESTAE parameter list
DIDOCS	- Device independent display operators console support
DIE	- Disabled interruption exit
DIR	- Deferred incident record
DMDT	- Domain descriptor table
DMVT	- Domain vector table
DPL	- DEQ purge list
DQE	- Descriptor queue element
DRQ	- Data ready queue
DSAB	- Data set association block
DSCB	- Data set control block
DSPCT	- Data set page correspondence table
DSPL	- Dump sort parameter list
DVT	- Destination vector table or display allocation vector table

ECB	- Event control block
ECC	- Error checking and correction
ECT	- Environment control table
EDB	- Extent descriptor table
EDL	- Eligible device list
EDT	- Eligible device table
EED	- Extended error descriptor
EIL	- Event indication list
EIP	- EXCP intercept processor
EMS	- Emergency signal
ENF	- Event notification facility
ENFPM	- ENF event parameter list
EOA	- End of address
EOE	- End of extent
EOV	- End of volume
EP	- Emulator program
EPATH	- Error path (recovery audit trail area)
EPS	- External page storage
ERP	- Error recovery procedures

ERPIB	- Error recovery procedures interface block
ESTAE	- Extended STAE
ESTAI	- Extended STAI
ET	- Entry table
ETE	- Entry table entry
ETIB	- Entry table information block
ETIX	- ETIB extension
EVNT	- Event table
EWA	- Common ERP work area
EX	- Entry table index

FBQE	- Free block queue element
FDB	- Feedback data block
FETWK	- Fetch work area
FIFO	- First in first out
FLIH	- First level interrupt handler
FMCB	- VTAM function management control block
FOE	- Fixed ownership element
FOT	- Fixed ownership table
FQE	- Free queue element
FRR	- Functional recovery routine
FRRS	- FRR stack
FSA	- Functional subsystem application
FSACB	- FSA control block
FSAXB	- FSACB extension
FSB	- Feedback status block
FVT	- Field vector table
FSI	- Functional subsystem interface
FSS	- Functional subsystem
FSSCB	- Functional subsystem control block
FSSWORK	- FSS PCE work area
FSSXB	- FSSCB extension

GCB	- Global resource serialization CTC-driver request block
GCC	- Global resource serialization CTC-driver control card table
GCL	- Global resource serialization CTC-driver link control block
GCP	- Global resource serialization CTC-driver buffer prefix
GCQ	- Global resource serialization CTC-driver queueing element
GCT	- Global resource serialization CTC-driver branch table
GCV	- Global resource serialization CTC-driver vector table
GCX	- Global resource serialization CTC-driver extract table
GDA	- Global data area
GPR	- General purpose register
GQHT	- Global queue hash table
GRS	- Global resource serialization
GSMQ	- Global service manager queue
GSPL	- Global system priority list
GSR	- Global shared resource
GTF	- Generalized trace facility
GVT	- Global resource serialization vector table
GVTX	- GVT extension

HFCT	- HASP FSS communications control block
HIR	- Hardware instruction retry

IC	- Instruction counter
ICNCB	- Intermediate controller node control block
IHSA	- Interrupt handler save area
ILC/CC	- Instruction length condition code
IOB	- Input output block
IOE	- I/O request element
IOMB	- I/O management block
IOQE	- I/O queue element
IORB	- I/O request block
IOS	- I/O supervisor
IOSB	- I/O supervisor block
IOT	- I/O table
IOWA	- I/O work area
IPC	- Inter-processor communication

IPCS	- Interactive problem control system
IPL	- Initial program load
IPS	- Installation performance specifications
IQE	- Interrupt queue element
IRB	- Interrupt request block
IRT	- IOS recovery table
ISAM	- Indexed sequential access method

JCL	- Job control language
JCT	- Job control table
JDT	- JCL definition table
JDVT	- JCL definition vector table
JES	- Job Entry Subsystem
JES2 NJE	- JES2 Network Job Entry (Program Product)
JESCT	- JES control table
JFCB	- Job file control block
JFCBX	- Job file control block extension
JIB	- JOE information block
JIX	- Job queue index
JOE	- Job output element
JOT	- Job output table
JPQ	- Job pack queue
JQE	- Job queue element
JSCB	- Job step control block
JSEL	- Job scheduling entry list
JSXL	- Job scheduling exit list

KSDS	- Key sequence data set

LCB	- TP line control block
LCCA	- Logical configuration communication area
LCCAVT	- LCCA vector table
LCH	- Logical channel queue
LCPB	- Logical channel program block
LCT	- Linkage control table
LDA	- Local data area
LFQ	- Local frame queue
LG	- Logical group
LGF	- Line group block
LGCB	- Logical group control block
LGE	- Logical group entry
LGN	- Logical group number
LGVT	- Logical group vector table
LGVTE	- Logical group vector table entry
LIFO	- Last in first out
LIT	- Lock interface table
LLE	- Load list element
LLQ	- Load list queue
LPA	- Link pack area or latent parameter area
LPDE	- Link pack directory entry
LPID	- Logical page identifier
LPME	- Logical to physical mapping entry (or) logical page mapping entry
LQHT	- Local queue hash table
LRB	- Logrec buffer
LSID	- Logical slot ID
LSMQ	- Local service manager queue
LSPL	- Local service priority list
LSQA	- Local system queue area
LT	- Linkage table
LTE	- Linkage table entry
LUB	- NCP logical unit block
LWA	- Logon work area
LWK	- Local work area
LX	- Linkage index
LXAT	- Linkage index allocation table

MCH	- Machine check handler
MCIC	- Machine check interrupt code

MCP	- Message control program
MCS	- Multiple console support
MFA	- Malfunction alert
MH	- Message handler
MIH	- Missing interrupt handler
MLPA	- Modified link pack area
MP	- Multiprocessor
MPF	- Message processing facility
MPFT	- MPF table
MPST	- Memory process scheduling table
MRB	- Message request block
MSFAB	- MSSFCALL SVC attention block
MSFCB	- MSSFCALL SVC control block
MSFKB	- MSSFCALL SVC communication block
MSS	- Mass storage subsystem
MSSC	- Mass storage system communicator
MSSF	- Monitoring and system support facility
MSVC	- Mass storage volume control
MVS	- Multiple Virtual Storage
MWA	- Module work area
NCB	- VTAM node control block
NCP	- Network Control Program
NIP	- Nucleus initialization program
OCR	- Output control record
OCT	- Output control table
OPWA	- Open work area
ORE	- Operator reply element
OUCB	- SRM-user control block
OUSB	- SRM-user swappable block
OUXB	- SRM-user extension block
PAB	- Process anchor block
PART	- Paging activity reference table
PARTE	- PART entry
PAT	- Page allocation table
PC/AUTH	- Program call/authorization
PCB	- Page control block
PCCA	- Physical configuration communication area
PCCAVT	- PCCA vector table
PCCB	- Private catalog control block
PCCW	- Paging channel command work area
PCE	- Processor control element
PCRA	- Program call recovery area
PDDB	- Peripheral data definition block
PDS	- Partitioned data set
PEL	- Parameter element
PEP	- Partitioned emulator program
PER	- Program event recording
PEXB	- Pool extent block
PFT	- Page frame table
PFTE	- Page frame table entry
PGT	- Page table
PGTE	- Page table entry
PICA	- Program interrupt control area
PIE	- Program interrupt element
PIT	- Partition information table
PIU	- Physical information unit
PLH	- Place holder
PLPA	- Pageable link pack area
PLPAD	- PLPA directory
PQCB	- Placeholder queue control block
PQE	- Partition queue element
PRB	- Program request block
PSA	- Prefixed save area
PSCB	- Protected step control block
PSS	- Process scheduling service
PST	- Process scheduling table

PSW	- Program status word
PTLB	- Purge translation lookaside buffer
PVT	- Paging vector table
PVTAFC	- PVT available frame count
PWF	- Power Warning Feature Support
PWKA	- Paging work area

QAB	- Queue anchor block
QCB	- Queue control block
QEL	- Queue element
QFPL	- ENQ/DEQ FRR parameter list
QFPL1	- Queue scanning services FRR parameter list
QHT	- Queue hash table
QWA	- Queue work area
QWB	- Queue work block
QXB	- Queue extension block

RACF	- Resource Access Control Facility (Program Product)
RB	- Request block
RBA	- Relative byte address
RBN	- Real block number
RCA	- RSM recovery communication area
RCB	- Resource control block
RCT	- Region control task
RDCM	- Resident display control module
RDF	- Record definition field
RDT	- Resource definition table
RDTE	- Resource definition table entry
REPL	- Ring processing ESTAE parameter list
RIB	- Resource information block
RIBE	- Resource information block extent
RIM	- Resource initialization module
RJE	- Remote job entry
RMCT	- Resource manager control table
RMF	- Resource Measurement Facility (Program Product)
RMS	- Recovery management support
RNLE	- Resource name list entry
RPH	- Request parameter header
RPL	- Request parameter list
RPT	- Request pool table
RQA	- Resource queue area
RQE	- Request queue element
RRPA	- Recovery routine parameter area
RSA	- Ring processing system authority message
RSAIRCD	- Ring processing information record
RSC	- Ring status change parameter list
RSL	- Ring processing system link block
RSM	- Real storage manager
RSMHD	- RSM header
RST	- Ring processing status table
RSV	- Ring processing system vector table
RTAM	- Remote terminal access method
RTCA	- Recovery termination control area
RTCT	- Recovery termination control table
RTM	- Recovery termination manager

S/A	- Stand-alone (dump program)
SAHT	- System/ASID hash table
SAM	- Sequential access method
SART	- Swap activity reference table
SAST	- Subsystem allocation sequence table
SAT	- Swap allocation table or system authorization table
SCCB	- Service call control block
SCCW	- Swap channel control work area
SCT	- Step control table
SDWA	- System diagnostic work area
SFT	- System function table or swap function table
SGT	- Segment table

SGTE	- Segment table entry
SHAS	- Subsystem hash table
SIC	- System initiated cancel
SIGP	- Signal processor instruction
SIO	- Start input/output
SIOT	- Step I/O table
SJF	- Scheduler JCL facility
SLIH	- Second level interrupt handler
SLIP	- Serviceability level indication processing
SLT	- System linkage table
SMF	- System management facility
SMPL	- Storage management parameter list entry
SMS	- Storage management services
SNA	- System Network Architecture
SPCT	- Swap control table
SPQE	- Subpool queue element
SQA	- System queue area
SRB	- Service request block
SRM	- System resources manager
SRR	- Serially reusable resource
SRRA	- Service routine recovery area
SSCP	- System services control point
SSCVT	- Subsystem communications vector table
SSI	- Subsystem interface
SSIB	- Subsystem identification block
SSOB	- Subsystem options block
SSQ	- SVRB suspend queue
SSRB	- Suspended service request block
SSVT	- Subsystem vector table
STAE	- Specify task abnormal exit
STAI	- Subtask abend intercept
STC	- Start task control
STCB	- Subtask control block
STKE	- Stack element
STOR	- Segment table origin register
SVC	- Supervisor call
SVRB	- Supervisor request block
SVT	- Supervisor vector table
SWA	- Scheduler work area
SWB	- Scheduler work block

TCAM	- Telecommunications Access Method
TCB	- Task control block
TCH	- Test channel
TDCM	- Pageable display control module
TEA	- Translation exception address
TH	- Transmission header
TIOC	- Terminal I/O coordinator
TIOT	- Task input/output table
TLB	- Translation lookaside buffer
TMC	- Task mode controller
TME	- Task mode element
TMP	- Terminal monitor program
TOD	- Time of day
TSB	- Terminal status block
TSO	- Time Sharing Option
TTE	- Trace table entry

UADS	- User attribute data sets
UCB	- Unit control block
UCM	- Unit control module
UCME	- Unit control module entry
UIC	- Unreferenced interval count
UPT	- User profile table

VBN	- Virtual block number
VBP	- Virtual block processor
VDSCB	- Virtual data set control block
VFCB	- Virtual fetch control block

VFHE	- Virtual fetch hash entry
VFPM	- Virtual fetch parameter list
VFVT	- Virtual fetch vector table
VFWK	- Virtual fetch work area
VIO	- Virtual I/O
VSAM	- Virtual storage access method
VSM	- Virtual storage management
VTAM	- Virtual Telecommunications Access Method
VTOC	- Volume table of contents
VUT	- Volume unload table
WAST	- Workload activity specification table
WMST	- Workload manager specification table
WQE	- Write queue element
WSC	- Wait state code
WTQE	- Wait queue element
XL	- Extent list
XMD	- Cross memory directory
XPTE	- External page table entry
XRBN	- Extended real block number
XSB	- Extended status block

Index

A

- abbreviations
  - list of D-1
- abend codes
  - C0D in ASM 5-134
  - EXCP 5-32
  - Service Processor Call SVC 5-279, 5-284
  - started task control 2-93
  - SWA manager 2-94
  - symptoms of IOS problems 5-28
  - 0B0 in allocation 5-194
  - 0C4 in allocation 5-195
  - 08x series in ASM 5-132
  - 306 abend in program manager 5-71
  - 806 abend in program manager 5-66
- abend dump debugging 2-86
- abend resource manager 5-66
- abnormal end appendages
  - with ERPs 5-49
- abnormal task termination (RTM) 5-225
- ACB (access method control block)
  - how to locate 5-160
  - major fields in 5-161
- ACCOUNT command processor A-41
- ACF/TCAM
  - See TCAM
- ACF/VTAM
  - See VTAM
- ACR
  - See alternate CPU recovery
- ACTION keyword
  - SLIP 2-112
- action message retention facility
  - debugging aids 5-251
- additional data gathering techniques 2-95
- address space
  - analysis 2-6
  - ASM's 5-117
  - dispatchable work in B-6
  - dispatcher's 5-8
  - initialization 5-84
  - OUCB queues 5-147
  - states recognized by SRM 5-142
  - termination 5-229
  - tests made by dispatcher 5-13
  - wait 5-9
- addresses
  - commonly bad 2-80
- AIA
  - RSM test on A-3
- allocation
  - of virtual storage 5-87
- allocation/unallocation
  - abends
    - 0B0 5-194
    - 0C4 5-195
  - address space (ALLOCAS) 5-180
  - address space termination 5-194
  - allocation
    - common 5-174
    - description 5-172
    - fixed device 5-175
    - generic 5-175
    - recovery 5-175
    - serialization 5-192
    - TP 5-175
  - batch initialization 5-173
  - common allocation 5-174
  - common unallocation 5-175
  - component analysis 5-172
  - control block processing 5-177
  - debugging aids 5-176
  - debugging hints 5-192
  - device selection 5-193
  - dynamic initialization 5-173
  - ESTAE processing 5-180
  - functional description 5-172
  - JFCB housekeeping 5-174
  - module naming conventions 5-176
  - reason codes 5-197
  - registers and save ares 5-177
  - unallocation
    - common 5-175
    - description 5-173
  - unit status recording 5-180
  - volume mount and verify 5-176, 5-195
- alternate CPU recovery (ACR)
  - problem analysis 2-75
- AMCBS
  - major fields in 5-159
- AMDPRDMP
  - control statements 2-97
  - example of use of data 4-10
  - GRSTRACE option
    - use for loop analysis 4-17
    - use for wait analysis 4-10
  - how to copy tapes 2-99
  - QCBTRACE option
    - use for loop analysis 4-17
    - use for wait analysis 4-10
- APF authorization 5-70
- appendage footprint table
  - with PWF 5-265
- appendages
  - abnormal end
    - with ERPs 5-49
- ASCB (address space control block)
  - analysis 2-6
  - indicators 2-14
- ASM (auxiliary storage manager)
  - address space structure 5-117

cell pools 5-117
component analysis 5-112
component functional flow 5-113
control blocks 5-116, 5-135
converting a slot number to full seek address 5-124
C0D abend 5-134
error analysis suggestions 5-127
finding the LSID for a given page 5-121
footprints and traces 5-118
FRR/ESTAE work areas 5-132
general debugging approach 5-119
incorrect pages 5-120
interfaces with other components 5-118
operating characteristics 5-117
page/swap data set errors 5-127
paging interlocks 5-119
recovery
as a debugging tool 5-133
considerations 5-131
footprints 5-133
structure 5-132
traces 5-132
register conventions 5-118
requesting I/O 5-114
requesting swap I/O 5-115
saving an LG 5-113
SDWA variable recording area 5-134
serialization 5-129
SRB structure 5-117
storage considerations 5-117
system mode 5-117
task structure 5-117
unuseable paging data sets 5-125
validity checking 5-128
08x abend codes 5-132
ASMDATA statement (of PRDMP) 2-99
ATA (ASM tracking area) 5-135
ATTACH (program manager function) 5-61
attention processing (TSO) A-46
audit trail area (EPATH) 5-136
auxiliary storage manager
See ASM

B

backout (for DEFINE/DELETE) 5-168
batch initialization 5-173
BLDL table analysis 4-27
BSHEADER data area 5-139
BUFCONBK data area 5-139
buffer
external call 2-59
LOGREC 2-38
translation lookaside 2-43

C

caller-defined SVC dump titles C-82
cancel process (RTM) 5-227
catalog communications area
See CCA
catalog management
backout 5-168
CMS function gate 5-167
component analysis 5-158
debugging aids 5-170
diagnostic output 5-167
establishing/releasing a recovery environment 5-165
how to find registers 5-158
maintaining a pushdown list end mark 5-166
major control blocks 5-159
major registers 5-159
module structure 5-164
recovery routine functions 5-167
tracking GETMAIN/FREEMAIN activity 5-166
VSAM catalog recovery logic 5-165
catalog parameter list (CTGPL)
major fields 5-162
CAXWA
major fields 5-161
major flags 5-162
CCA (catalog communication area)
major fields 5-163
major flags 5-164
CCH (channel check handler)
diagnostic aids 5-266
message IGF0021 5-266
PCCA fields 5-267
CDE (contents directory entry)
allocation 5-70
analysis 4-27
initialization by IDENTIFY 5-65
order of on ALPAQ 5-68
pool control 5-72
cell pool anchor block
See CPAB
cell pool management
VSM 5-92
channel check handler
See CCH
channel program
with ERPs 5-50
channel scheduler
invoked for IOS 5-25
channel set ID 5-32
channel set switching 5-32
checkpoint/restart
with SLIP command 2-128
CHNGDUMP command
to change SDUMP contents 2-95, 3-5
to override SVC dump parameters 2-99
class locks
with ASM 5-130
CML (cross memory lock)
classification 2-26
definition 2-22
location 2-26

CMS function gate 5-167
CMS lock 2-22
CMS lockword
  contents 2-25
  suspend queues 2-28
CMSEQDQ lock 2-22
CMSSMF lock 2-22
COMM task
  See communications task
command processing (global resource serialization)
  control block overview 5-314
  introduction 5-306
  module flow 5-331
command processor
  and TMP interface A-37
  parameter list A-39
common allocation 5-174
common service area
  See CSA
common unallocation 5-175
communications task
  component analysis 5-244
  control blocks 5-246
  current status 4-13
  debugging hints 5-248
  functional description 5-244
  sequence of processing 5-245
compare and swap
  serialization with ASM 5-131
completion codes in IOSB for ASM errors 5-126
component analysis 5-1
considerations
  MVS 2-1
console
  not responding to attention 5-248
console switching 5-251
contents directory entries
  See CDE
contents supervision
  See virtual fetch
CONTROL Q command debugging aids 5-253
converting virtual to real addresses 5-108
CPAB 5-92
CPUDATA statement (of PRDMP) 2-98
CQE control block 5-246
cross memory lock
  See CML
cross memory services
  component analysis 5-288
  lock description 2-22, 2-28
  lock suspend queues 2-28
cross memory services lock 2-28
CSA (common service area)
  analysis of use of 4-28
CTC processing
  control block overview 5-312
  debugging hints 5-344
  introduction 5-307
  module flow 5-322, 5-323
CTGPL (catalog parameter list)
  major fields 5-162
current recovery stack
  See FRR stacks
CVOL processor 5-165
CVTMAP statement (of PRDMP) 2-98
CXSA control block 5-246
C0D abend in ASM 5-134

D

DASD ERPs 5-53
data gathering techniques 2-95
data sets
  page/swap errors 5-127
DDR (dynamic device reconfiguration)
  DDRCOM table 5-268
  DERPLIST parameter list 5-269
  diagnostic aids 5-268
  return codes 5-272
  software recording 5-272
  storage dump 5-272
  task description 5-268
DDRCOM table 5-268
debugging hints (chapter) 2-75
DEFINE/DELETE backout 5-169
DELETE (function of program manager) 5-64
delete listen request (ENF)
  description 5-212
  example 5-219
DERPLIST parameter list 5-269
DIAGNOSE instruction
  description 5-277, 5-281
  trace entry 2-64
diagnostic materials approach 3-1
diagnostic techniques
  component analysis 5-1
  diagnostic approach 3-1
  important considerations 2-1
  introduction 1-1
  process flows A-1
  stand-alone dump analysis B-1
  SVC dump title directory C-1
  sympton analysis approach 4-1
DIDOCS
  in-operation indicator 5-252
  locking 5-252
  trace table 5-252
disabled loop
  See loops
disabled mode 2-12
disabled wait
  See waits
DISP lock
  description 2-22
dispatchability tests
  address space 5-13
  SRB 5-13
  task 5-13
dispatchable units of work

in an address space B-6
priority and location 5-4
dispatcher
abend codes 5-16
component analysis 5-2
determining the last dispatch 5-14
dispatchability tests 5-13
error conditions 5-16
important entry points 5-2
processing overview 5-10
recovery considerations 5-15
DISPLAY DUMP command 2-95
DISPLAY GRS command
introduction 5-305, 5-306
module flow 5-332
DSNLIST data area 5-140
dummy task B-6
dump analysis
abend dumps 2-86
areas 4-24
MP 2-45
problem program 2-86
stand-alone 2-2, 3-2, B-1
summary dump 2-89
tracing procedure 2-66
DUMP command 2-95
dump footprint table
with PWF 5-265
dumps
how to copy tapes 2-99
how to print 2-97
SVC dump titles C-1
tailoring with SLIP 2-117
dynamic device reconfiguration
See DDR

E

EDIT command processor A-41
EDIT statement (of PRDMP) 2-98
EDT
See eligible device table
EED
important fields 2-41
EIL control block 5-246
eligible device table (EDT)
description 5-193
verification routine 5-194
Emergency Signal instruction
See EMS
EMS (function of SIGP)
definition 2-49
process flow 2-56
enabled loop
See loops
enabled loop exception 4-17
enabled wait
See waits
ENF (event notification facility)
component analysis 5-212
control blocks 5-215
event codes 5-212
examples of logic flow 5-217
exit routines 5-214
initialization 5-216
processing 5-217
recovery routines 5-220
requests for services 5-212
return codes 5-217
ENFCT control block 5-215
ENFDS control block 5-215
ENFLS control block 5-215
ENFPM parameter list 5-212
ENFVT control block 5-215
ENQ/DEQ
analysis for enabled waits 4-10
analysis for performance degradation 4-23
common ENQ resource names 4-12
enqueue lockout B-7
global save area 4-24
ENQ/DEQ/RESERVE services
control block overview 5-314, 5-315
debugging hints 5-345
introduction 5-305, 5-307
module flow 5-339, 5-340
EP mode traces 4-22
EPATH (error path) 5-136
ERPs (error recovery procedures)
abnormal end appendages 2-77, 5-49
description 5-48
EWA (ERP work area) 5-30
traps 5-54
with ACR 2-76
error id 5-232
error interpreter table 5-50
error recovery procedures
See ERPs
event notification facility
See ENF
event parameter list (ENFPM) 5-212
EWA (ERP work area) 5-30
EXCP abend codes 5-32
EXCP debugging area (XDBA) 5-33
EXCP major control block relationships 5-27
EXCP/IOS process flow A-12
execution modes
See system mode
exit resource manager 5-64
exit routines (ENF) 5-214
explicit waits 2-7
extended error descriptor (EED) 2-41, 5-222
external call (XC function of SIGP)
description 2-49
process flow 2-55
external symptoms 1-2

F

FETCH
  program manager work area (FETWK) 5-72
FORCE command 5-228
FORMAT statement (of PRDMP) 2-98
formatted RTM control blocks 2-42
formatting (LOGREC buffer) 2-38
FREEMAIN
  See GETMAIN/FREEMAIN
FRR (functional recovery routine)
  ASM's 5-132
  FRR stacks
    important field contents 2-40, B-15
  GETMAINS's 5-90
  RSM's 5-104
FRR/ESTAE
  ASM work areas 5-133
FSS (functional subsystem)
functional recovery routine
  See FRR
functional subsystem
  See FSS
functional subsystem application
  See FSA
functional subsystem interface
  See FSI

G

GDA (global data area) for VSM 5-89
GETMAIN processing A-15
  FREEMAIN processing A-16
GETMAIN/FREEMAIN
  GETMAIN FRR 5-90
  indication in trace table 2-71
  process flow A-15
  SVC 120 5-94
  virtual storage allocation 5-87
GETPART/FREEPART 5-86
global data area for VSM 5-89
global indicators of current system state 2-2
global locks
  definition 2-21
  error status B-12
  requests for unavailable 2-26
  spin locks
    content of lockword 2-25
    definition 2-23
  suspend locks
    content of lockword 2-25
    definition 2-23
global resource serialization
  component analysis 5-305
  control blocks 5-308, 5-348
  control blocks overviews 5-310
  diagnostic aids 5-343
  functional overview 5-306
  module flow diagrams 5-320
  module flow overview 5-321
  module naming conventions 5-306
  probe points 5-344
  recovery routines 5-351
  serialization 5-350
  system indicators 5-343
  SYS1.LOGREC recording 5-351
global SRBs
  control block relationships 5-6
  dispatching 5-5
  mode indicators set by dispatcher 5-14
  queue structure 5-6
  status indicators B-17
global system analysis (chapter) 2-2
GQSCAN macro
  See also queue scanning services
  introduction 5-305
GRSTRACE (AMDPRDMP option)
  use for loop analysis 4-17
  use for wait analysis 4-10
GRSTRACE statement (of PRDMP) 2-98
GSMQ/LSMQ 2-5
GSPLs/LSPLs 2-5
GTF (generalized trace facility)
  bypassing lost events 2-68
  I/O and SIO trace (EP) 4-21
  RNIO trace 4-21
  trace examples 2-64
  with SLIP command 2-115

H

hardcopy log
  master trace 2-71
hardware-detected errors
  analysis 3-8
hierarchy of locks 2-23

I

I/O
  capability in MP 2-59
  enabled B-6
  incomplete B-7
  NCP trace
    See NCP
  problems in enabled waits 4-8
  requesting (ASM) 5-114
  requesting swap 5-115
  trace entries 2-69
  VTAM I/O trace
    See VTAM
I/O manager
  debugging 5-157
  modules 5-157
I/O request
  information in PLH 5-152
IDENTIFY (function of program manager) 5-65
IEAVGFA tests by RSM A-2
IEAVIOCP tests by RSM A-6

IEAVPIOP tests by RSM A-3
IEAVPIX tests by RSM A-2
IEAVSWIN A-7
ILC/CC important field contents 2-3
in-operation indicator
  DIDOCS 5-252
incorrect output (chapter)
  analyzing system functions 4-30
  initial analysis 4-29
  isolating the component 4-29
installation performance specifications (IPS) 5-141
inter-processor communication 2-49
interactive problem control system (IPCS) 1-4
intercept condition
  ERPs 5-51
interruptions
  status saving 2-15
intersect
  description 2-31
  service routine 4-16
IOB
  See IOMB
IOMB 5-157
IOS (I/O supervisor)
  ABEND codes 5-28
  back-end processing 5-25, A-14
  component analysis 5-25
  diagnostic aids 5-32
  ERP processing 5-48
  EXCP/IOS process flow A-12
  front-end processing 5-25, A-12
  general hints 5-30
  loops 5-28
  major control block relationships 5-27
  message table 5-46
  output of recovery procedures 5-34
  POST STATUS A-14
  problem analysis . 5-25
  processing overview 5-26
  return codes 5-47
  save areas 5-30
  SDWA variable recording area 5-34
  storage manager queues 4-27
  VTAM interaction A-18
  wait state codes 5-47
  wait states 5-29
IOS/EXCP process flow A-13
IOSB fields 5-43
IOSCAT lock 2-22, 2-26
IOSCOD field 5-44
IOSDRVID field 5-43
IOSLCH lock 2-22, 2-26
IOSPROC field 5-44
IOSUCB lock 2-22, 2-26
IOSYNCH lock 2-22, 2-26
IPC
  See inter-processor communication
IPCS 1-4

J

JES2 (job entry subsystem)
  note to readers 5-201
  operator commands for status information 4-24
JES3
  See note in Preface
JFCB housekeeping 5-174
join processing
  module flow 5-338

K

K Q command debugging aids 5-253
key-length-data format
  SDWAVRA 2-35

L

LCCA indicators 2-13
LCH queues
  analysis for enabled waits 4-9
LDA
  important flags 5-87
LG
  saving 5-113
line drop (TSO processing) A-34
LINK (function of program manager)
  description 5-60
  module search sequence 5-68
listen request (ENF)
  description 5-212
  example 5-217
  exit routine 5-214
LMOD map
  how to print 2-103
LOAD (function of program manager)
  description 5-64
  module search sequence 5-68
local lock
  definition 2-21
  lockword contents 2-25
  lockword location 2-26
  requests for unavailable 2-27
  suspend (definition) 2-23
  with ASM 5-130
local SRBs
  dispatching 5-5, 5-7
  dispatching priority in address space 5-8
  mode indicators set by dispatcher 5-15
  status indicators B-16
locating status information in a storage dump 2-15
lock interface table (IEAVESLA) 2-26
locked mode
  definition 2-13
  status saving during execution 2-11
locking (chapter) 2-21
locks (see also lockwords)

  - categories 2-21
  - classification 2-26
  - determining which held on a processor 2-24
  - hierarchy 2-23
  - location of 2-26
  - PSAHLSI bits 2-24
  - requests for unavailable 2-26
  - table of definitions 2-22
  - types 2-22
  - with ASM 5-117, 5-129
- lockwords
  - contents of 2-25
  - how to find 2-26
- LOGDATA verb 2-38
- logging
  - ERPs 5-51
- logical groups
  - assigning 5-112
  - releasing 5-113
- logon
  - command processor A-41
  - diagnostic aids A-32
  - initialization A-27
  - monitor A-29
  - post codes A-33
  - process overview A-24
  - scheduler A-29, A-31
  - scheduler router A-27
  - TMP exit A-31
  - verification A-29, A-30
  - work area A-28, A-32
- LOGREC
  - analysis 2-34
  - considerations 2-36
  - for debugging SVC dump 5-234
  - formatting 2-38
  - how to print 2-104
  - listing LOGREC data set 2-34
  - recording control buffer 2-38
- loop recording option 4-16
- loops
  - analysis 4-15
  - disabled
    - definition 4-15
    - system mode 4-18
  - enabled
    - definition 4-15
    - exception 4-17
  - in lock manager code B-16, B-18
  - symptoms of IOS problems 5-28
- low storage overlays 2-78
- LPAMAP statement (of PRDMP) 2-98
- LPSW
  - common uses of 4-2
- LSID
  - finding for a page 5-121
- LSMQ 2-5
- LSPL 2-5

M

- machine check handler
  - See MCH
- machine checks
  - debugging 2-81
  - interruption code (MCIC) 2-81
  - reference matrix 2-84
- master trace
  - debugging aids 5-254
  - description 2-71
  - trace table 2-72
  - trace table entry 2-73
- MCH (machine check handler)
  - diagnostic aids 5-256
  - PWA 5-257
  - return codes 5-256
- MCIC (machine check interruption code) 2-81
- message flow
  - through the system 4-20
- message processing facility (MPF)
  - description 2-74
  - table (MPFT) 2-74
- messages
  - ERPs 5-51
  - IGF0021 5-266
  - IOS message table 5-46
  - lost 5-249
  - master trace 2-71
  - missing on one console 5-250
  - processing facility 2-74
  - routed to wrong console 5-250
  - truncated 5-250
- MIH (missing interruption handler)
  - diagnostic aids 5-273
  - process 5-273
  - software recording 5-276
  - storage dumps 5-276
  - work area 5-273
- miscellaneous debugging hints (chapter) 2-75
- missing interruption handler
  - See MIH
- module search sequence
  - for LINK,ATTACH, XCTL, LOAD 5-68
  - of private libraries 5-69
- module subpools 5-71
- monitoring and system support facility
  - See MSSF
- MP (multiprocessing)
  - activity in trace table 2-70
  - associated data areas 2-45
  - debugging hints 2-58
  - direct services 2-50
  - dump analysis 2-45
  - dump analysis hints 2-48
  - effects on problem analysis 2-43
  - features of MP environment 2-43
  - parallelism 2-46
  - PSA analysis B-3
  - remote immediate services 2-52

remote pendable services 2-51
SIGP instruction 2-49
system stop routine 2-133
MPF (message processing facility) 2-74
MPFT (MPF table) 2-74
MSGBUFFER data area 5-140
MSSF (monitoring and system support facility)
with MSSFCALL DIAGNOSE instruction 5-277, 5-281
with Service Processor Call SVC 5-277
MSSFCALL data block
description 5-278
MSSFCALL DIAGNOSE instruction
condition codes 5-282
description 5-277, 5-281
trace entry 2-64
MSSFCALL SVC
See Service Processor Call SVC
multiprocessing
See MP
MVS trace
See trace table

N

NCP (network control program)
traces 4-22
no-work wait (see also waits) 4-7
normal task termination 5-224

O

O/C/EOV (open/close/end-of-volume)
abends 2-80
DEB chaining 5-156
debugging aids 5-155
ENQs issued by 5-156
messages 5-155
online problem analysis 1-4
open/close/end-of-volume
See O/C/EOV
OPERATOR command processor A-41
operator commands
for status information 4-23
to identify performance degradation 4-23
operator-defined SVC dump titles C-82
ORE control block 5-246
OUCB (SRM user control block)
analysis 5-147
important fields 5-142
OUTPUT command processor A-41
output of IOS recovery procedures 5-34
overlays
storage
cause of wait state PSWs 4-3
how to locate the trace table 2-62
in low storage 2-78
pattern recognition 2-77

P

page control block
See PCB
page fault
process flow A-2, A-4
reclaim 5-102
trace examples 2-64
waits 4-9
page frame table entries
See PFTE
page stealing 5-101
page waits B-8
page/swap data set errors 5-127
paging
finding the LSID 5-121
incorrect pages 5-120
interlocks 5-119
process 5-101
unuseable data sets 5-125
paging requests
analysis 4-27
parallelism 2-46
PART/PAT bit
locating 5-123
pattern recognition 2-77
PC/AUTH services
control blocks 5-291
debugging hints 5-299
description 5-288
module structure 5-289
process flow 5-290
recovery considerations 5-296
recovery data areas 5-297
SLIP traps 5-301
PCB (page control block)
important fields in 5-99
trace facility 5-111
use in debugging A-9
PCCA fields for CCH 5-267
PCCB major fields and flags 5-159
PCLINK services
debugging hints 5-304
description 5-302
module structure 5-304
STKE control block 5-302
PER activation/deactivation
debugging aids 5-240
process flow 5-230
recovery 5-240
PER monitoring 2-105
PER traps 2-127
performance degradation
chapter on 4-23
dump analysis area 4-24
operator commands to identify 4-23
PFTE (page frame table entries)
analysis 4-27

important fields 5-101
PGTE
RSM test on A-2
physically disabled mode 2-12
PLH (place holder) 5-152
post codes
LOGON A-33
Power Warning Feature Support
See PWF
PRB initialization 5-61
PRDMP
See AMDPRDMP
PRE-TMP exit A-31
PRINT statement (of PRDMP) 2-98
printer ERP 5-53
private libraries
module search sequence 5-69
problem analysis techniques 1-1
problem program dump debugging 2-86
process flows
chapter A-1
EXCP/IOS A-12
GETMAIN/FREEMAIN A-15
page faults (RSM processing) A-2
TSO A-21
VTAM A-18
processor work area (PWA) 5-257
program call/authorization
See PC/AUTH services
program checks
interrupts 5-16
program manager
APF authorization 5-70
ATTACH 5-61
CDE
allocation 5-70
CDE pool control 5-72
component analysis 5-55
control blocks 5-55, 5-57
DELETE 5-64
exit resource manager 5-64
FETCH/program manager work area 5-72
functional description 5-55
functional flow 5-60
IDENTIFY 5-65
LINK 5-60
LOAD 5-64
module description 5-56
module search sequence
for LINK, ATTACH, XCTL, LOAD 5-68
of private libraries 5-69
module subpools 5-71
organization 5-55
overview 5-57
queue validation 5-58
queues
description 5-55, 5-57
RB extended save area 5-72
SYNCH 5-65
system initialization 5-58
XCTL 5-61
306 abend 5-71
806 abend 5-66
PSA (prefixed save area)
contents of important fields 2-4
indicators 2-13
used to determine current system state 2-2, 2-13
using as a patch area 2-104
PSW (program status word)
analysis 2-3
wait state B-2
pushdown list end mark
maintaining 5-166
PWA work area 5-257
PWF (Power Warning Feature)
appendage footprint table 5-265
communications area 5-258
data areas 5-257
diagnostic aids 5-257
dump footprint table 5-265
LOGREC recording 5-266
return codes 5-257

Q

QCBTRACE (AMDPRDMP option)
use for loop analysis 4-17
use for wait analysis 4-10
QCBTRACE statement (of PRDMP) 2-98
QTIP
processing A-24
queue scanning services
control block overview 5-316, 5-317
introduction 5-305, 5-308
module flow 5-342

R

RB (request block)
analysis 2-8
extended save area (RBEXSAVE) 5-72
manipulation by XCTL 5-63
new RB initialization for XCTL 5-62
RCB (recording control buffer) 2-38
RDCM (resident display control module) 5-246
real addresses
converting 5-108
real frame shortage
indicators 4-28
real storage manager
See RSM
reason codes
allocation 5-197
RSM 5-105
started task control 2-93
SWA manager 2-94
reclaim (function of RSM) 5-102
record management

  - debugging aids 5-153
  - processing 5-151
- recovery audit trail (ASM) 5-136
- recovery management support
  - See RMS
- recovery procedures
  - output of IOS modules 5-34
- recovery stack 2-13
- recovery work areas
  - use of 2-33
- register conventions
  - ASM 5-118
- relate (function of RSM) 5-103
- replies
  - lost 5-249
- requesting
  - I/O (ASM) 5-114
  - swap I/O (ASM) 5-115
- reset services
  - See stop/reset services
- resource request processing
  - See global resource serialization
- RESUME service
  - See SUSPEND/RESUME/TCTL services
- retry process (RTM) 5-226
- retry/restart
  - with ERPs 5-50
- return codes
  - Service Processor Call SVC 5-279, 5-284
- ring processing
  - control block overview 5-313
  - debugging hints 5-345
  - introduction 5-306
  - module flow 5-324, 5-325, 5-326, 5-327, 5-328, 5-329, 5-330
- RMCT (SRM control table)
  - system indicators 5-143
- RMS (recovery management support)
  - component analysis
  - diagnostic aids
    - CCH 5-256, 5-266
    - DDR 5-256, 5-268
    - MCH 5-256
    - MIH 5-256, 5-273
    - PWF 5-256, 5-257
- RPL error fields 5-151
- RSM (real storage manager)
  - abend reason codes 5-105
  - component analysis 5-97
  - debugging tips 5-107
  - major control blocks 5-97
  - page fault processing A-2
  - page stealing process 5-101
  - reclaim 5-102
  - recovery 5-104
  - relate 5-103
- RTM (recovery termination manager)
  - cancel 5-227
  - component analysis 5-221
  - error id 5-232
  - extended error descriptor (EED) 5-222
  - FORCE command 5-228
  - functional description 5-221
  - hardware error processing 5-222
  - major RTM modules 5-221
  - PER processing 5-230
  - process flow 5-222
  - retry 5-226
  - RTM1 5-221
  - RTM2 5-221
  - SLIP command debugging aids 5-238
  - stack vector table 2-13
  - SVC dump debugging aids 5-233
  - system diagnostic work area (SDWA) 5-222
  - termination
    - abnormal task 5-225
    - address space 5-229
    - normal task 5-224
- RTM2WA
  - definition 2-41

S

- SALLOC lock
  - description 2-22, 2-26
  - with ASM 5-129
- SCCB (service call control block) 5-284
- SCHEDULE macro 2-11
- scheduler work area
  - See SWA manager
- SDUMP macro C-1
- SDUMPs (see also SVC dumps)
  - analysis 3-4
  - how to change contents of 3-5
  - parameter list 3-5
  - titles C-1
- SDWA (system diagnostic work area)
  - data recorded by dispatcher 5-15
  - use by SYS1.LOGREC 2-34
  - use in FRR stack 2-41
  - use in RTM2 2-42
  - with SVC dump 5-234, 5-235
- SDWAVRA (SDWA variable recording area)
  - entries 5-90
  - error indicators 5-91
  - key-length-data format 2-35
  - use by ASM 5-134
  - use by catalog management 5-167
- sense command
  - with ERPs 5-52
- serialization
  - ASM 5-129
- service call control block (SCCB) 5-284
- SERVICE CALL instruction
  - condition codes 5-287
  - description 5-283, 5-287
  - trace entry 2-64
- Service Processor Architecture 5-283
- Service Processor Call SVC

abend codes 5-279, 5-284
control blocks 5-280, 5-286
data block 5-278
description 5-277, 5-283
introduction 5-277, 5-283
processing 5-279, 5-285
return codes 5-279, 5-284
trace entry 2-64
with ACR 2-76
with MSSFCALL DIAGNOSE instruction 5-277
with SERVICE CALL instruction 5-283

signal request (ENF)
description 5-212
example 5-218
exit routine 5-214

SIGP (signal processor) instruction
description 2-49
return codes 2-50
XC function 2-49

SLIP command
ACTION keyword 2-112
control blocks 5-242
controlling traps 2-125
debugging aids 5-238
description 2-105
designing an effective trap 2-125
dump tailoring 2-117
event qualifiers 2-105
examples 2-118
examples with TSO 2-124
PER monitoring 2-105
placement of PER traps 2-127
processor recovery 5-239
summary 2-129
trap design 2-125
using 2-105
with checkpoint/restart 2-128

SLIP processor
PER activation/deactivation 5-240
recovery 5-239

slot number
converting to full seek address 5-124

SMF suspension 2-19

software-detected errors
analysis 3-7

SPCT (swap control table)
important fields in 5-100

special exits
dispatching 5-4

spin locks
definition 2-22

SRB (see also local and global SRBs)
dispatching queues 2-5, 5-5
local 5-7
mode 2-11
mode interruptions 2-15
suspension 2-8, 2-17, 2-27
tests made by dispatcher 5-13

SRB/SSRB pool manager
description 5-17
entry points 5-17
error conditions 5-19
recovery considerations 5-18

SRM (system resources manager)
address space states 5-142
component analysis 5-141
control blocks 5-145
entry point summaries 5-149
error recovery 5-148
indicators 5-143
objectives 5-141
SDWA data 5-149
system (RMCT) indicators 5-143
user (OUCB) indicators 5-147

SSB (SMF suspend block) 2-19

SSI (subsystem interface)
component analysis 5-202
debugging hints 5-211
function codes 5-211
initialization processing 5-202
logic flow examples 5-208
major control blocks 5-203
requesting services 5-206
return codes 5-208
subsystem
allocation sequence table 5-202
communications vector table (SSCVT) 5-202
vector table (SSVT) 5-202

stand-alone dump
analysis 2-2
analysis procedure B-6
chapter on 3-2
description B-1
determining system mode from 2-13
flowchart B-5
how to print 2-97
special notes 2-2

started task control
See STC

status information
locating in storage dump 2-15

STC (started task control)
abend codes 2-93
reason codes 2-93

step initiation/termination 5-86

STKE control block 5-302

stop/reset services
description 5-19
entry points 5-19
error conditions 5-21
recovery considerations 5-20

storage management (global resource serialization)
control block overview 5-318
debugging hints 5-348
introduction 5-308

SUBMIT command processor A-42

subpools for modules 5-71

subsystem interface
See SSI or FSI

SUMDUMP output 2-89

- SUMDUMP statement (of PRDMP) 2-98
- summary dump 2-89
- SUMMARY statement (of PRDMP) 2-98
- super bits
  - See PSASUPER
- supervisor control 5-2
- superzaps
  - system stop routine 2-133
  - to expand trace table 2-133
- suspend locks
  - definition 2-22
- SUSPEND/RESUME/TCTL services
  - description 5-22
  - entry points 5-22
  - error conditions 5-24
  - recovery considerations 5-23
- suspended
  - locally locked tasks 2-17
  - SRB status 2-8
  - SRB/task with lock held B-13
  - tasks or address space caused by unsatisfied ENQ request 4-10
- suspension
  - SMF 2-19
- SVC D entries in trace table 2-70
- SVC dump title directory C-1
- SVC dumps
  - analysis 3-4
  - caller-defined titles C-82
  - debugging of
    - control blocks, use of 5-237
    - SLIP traps, use in 5-234
    - SYS1.LOGREC, use in 5-234
  - entry points
  - error conditions 5-234
  - how to override parameters 2-99
  - module cross reference C-85
  - operator-defined titles C-82
  - resource cleanup 5-238
  - SDUMP macro BRANCH= parameter
  - system-defined titles C-2
  - titles C-1
  - without titles C-83
- SVC 122 5-277, 5-283
- SWA (scheduler work area) manager
  - reason codes 2-94
- swap transition flags 5-142
- swap-in process A-7
- swap-out process A-9, A-10
- swapping
  - process flow A-7
- SWIN (IEVSWIN) A-7
- sympton analysis approach 4-1
- SYNCH (function of program manager) 5-65
- SYSABENDs
  - analysis approach 3-7
- SYSMDUMPs
  - analysis approach 3-7
- system degradation
  - See performance degradation
- system diagnostic work area
  - See SDWA
- system execution modes and status saving 2-10
- system hung
  - See waits
- system modes
  - determining from stand-alone dump 2-13
  - locked mode 2-13
  - physically disabled mode 2-12
  - SRB mode 2-11
  - task mode 2-10
- system options for SVC dump 2-99
- system resources manager
  - See SRM
- system stop routine 2-133
- system-defined SVC dump titles C-2
- SYSUDUMPs
  - analysis approach 3-7
- SYSZEC16-PURGE 4-12
- SYSZVARY-x 4-12
- SYS1.COMWRITE data set
  - how to print 2-103
- SYS1.DUMP
  - how to clear without printing 2-102
  - how to print 2-101
- SYS1.LOGREC
  - See LOGREC
- SYS1.STGINDEX
  - how to recreate 2-103
- SYS1.UADS
  - how to rebuild 2-100

T

- tape ERP 5-53
- task
  - analysis 2-7
  - locally locked interrupted 2-13
  - locally locked suspended 2-17, 2-25, 2-27, 4-9
  - mode indicators set by dispatcher 5-15
  - mode interruptions 2-15
  - RB structure 2-8
  - tests made by dispatcher 5-13
- task mode
  - description 2-10
- TCAM
  - organization after a TSO logon A-26
  - TIOC logon processing A-27
  - traces 4-22
  - TSO terminal I/O diagnostic techniques A-45
- TCB (task control block)
  - analysis 2-7
  - dispatching priority in address space 5-8
  - summary report 2-5
  - suspended with lock held B-6
  - with global resource serialization 5-311
- TCTL services
  - See SUSPEND/RESUME/TCTL services
- TDCM (pageable display control module) 5-246

teleprocessing
  See TP
timer value in trace table 2-69
TMP/command processor
  interface A-37
  work area A-39
TP (teleprocessing)
  typical problems 4-20
TP traces (see also trace table)
  types of 4-21
trace entry
  Service Processor Call SVC 2-64
trace table
  cautionary notes 2-69
  how to expand 2-133
  how to locate 2-61
  master trace 2-72
  traps
    SLIP command 2-105
  types of entries 2-63
  with DIDOCS 5-252
traces (see also trace table)
  analysis of 2-61
  events not traced 2-70
  examples 2-64
  interpreting 2-66
  master trace 2-71
  PCB facility 5-111
  summary of 4-21
  summary of TP traces 4-21
  unit exception on 3705 2-70
TSO (time sharing option)
  APAR documentation A-48
  attention processing A-46
  command processor recovery A-41
  line drop processing A-34, A-35
  overview of logon processing A-22
  process flow A-21
  terminal I/O flow A-42
  time sharing initialization A-21
  TSO/TIOC terminal I/O diagnostic techniques A-45

U

UCB
  analysis for enabled waits 4-9
UCM (unit control module) 5-246
UCME (UCM entry) 5-246
unallocation
  description 5-173
unit allocation status recording
  control blocks 5-181
  debugging hints 5-183
  description 5-180
  recovery considerations 5-183
unit check
  with ERPs 5-52
use of recovery work areas for problem analysis 2-33

V

VARY GRS command
  introduction 5-305, 5-306
  PURGE module flow 5-332
  QUIESCE module flow 5-333, 5-334
  RESTART module flow 5-335, 5-336, 5-337
virtual addresses
  converting 5-108
virtual fetch
  control blocks 5-78
  debugging hints 5-80
  description 5-74
  functional flow 5-75
  module organization 5-74
  recovery processing 5-79
virtual storage access method
  See VSAM
virtual storage manager
  See VSM
virtual telecommunications access method
  See VTAM
volume mount and verify 5-176, 5-195
VSAM (virtual storage access method)
  component analysis 5-151
  debugging aids 5-153
  error codes 5-154, 5-155
  I/O manager debugging 5-157
  O/C/EOV debugging aids 5-155
  O/C/EOV messages 5-155
  record management
    buffer control block (BUFC) 5-152
  request parameter list (RPL) 5-151
VSM (virtual storage manager)
  address space initialization 5-84
  allocation 5-87
  basic functions 5-82
  cell pool management 5-92
  control block usage 5-85
  debugging hints 5-92
  GETMAIN/FREEMAIN process flow A-15
  global data areas (GDA) 5-89
  step initialization/termination 5-86
  view of MVS storage 5-83
VTAM (virtual telecommunications access method)
  note to readers 5-150
  process flow A-18
  traces 4-22

W

wait address space 5-9
waits
  chapter on 4-2
  disabled
    analysis approach 4-3
    characteristics of 4-2

        locked console exception   4-3
        with communications task   5-249
    enabled
        analysis approach   4-5
        analysis via trace table   2-69
        characteristics of   4-4
        with communications task   5-248
        with global resource serialization   5-343
    enabled loop exception   4-17
    explicit   2-7, B-8
    in user code   B-18
    indications of paging interlocks   5-119
    no-work wait   4-7
    OUCB analysis   5-147
    page fault waits   4-9
    page waits   B-8
    wait state PSW   B-2
window spin   2-52
work queues
    TCBs
        address space analysis   2-5
WQE control block   5-246

X

XC (SIGP external call)
    definition   2-49
    process flow   2-55
XCTL (function of program manager)
    description   5-61
    module search sequence   5-68
    RB manipulation   5-63
XDBA (EXCP debugging area)   5-33
XPTE
    RSM test on   A-2

Z

zaps
    See superzaps

0

0B0 abend in allocation   5-194
0C4 abend in allocation   5-195
08x abends in ASM   5-132

3

306 abend in program manager   5-71

8

806 abend in program manager   5-66

IBM /Technical Newsletter

This Newsletter No.	SN28-5095
Date	December 27, 1985
Base Publication No.	SY28-1133-2
File No.	S370-37
Prerequisite Newsletters/ Supplements	None

MVS Diagnostic Techniques

MVS/System Product - JES3, Program No. 5665-291
MVS/System Product - JES2, Program No. 5740-XC6

This Technical Newsletter contains replacement pages for *MVS Diagnostic Techniques* in support of MVS/System Product Version 1 Release 3.6.

Before inserting any of the attached pages into *MVS Diagnostic Techniques*, 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*
Cover - Edition Notice	Cover - Edition Notice
vii - viii	vii - viii
xxv - xxvi	xxv - xxvi
5-303 - 5-304	5-303 - 5-304
C-3 - C-4	C-3 - C-4
C-33 - C-34	C-33 - C-34
C-81 - C-82	C-81 - C-82
C-85 - C-86	C-85 - C-86

*If you are inserting pages from different Newsletters/Supplements and ***identical*** page numbers are involved, always use the page 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.

A change to the text or to an illustration is indicated by a vertical line to the left of the change.

Summary of Amendments

This Technical Newsletter contains updates in support of Version 1 Release 3.6 of MVS/System Product and several technical corrections.

Note: Please file this cover letter at the back of the publication to provide a record of changes.

**IBM Corporation, Information Development, Dept. D58, Building 921-2,
P.O. Box 390, Poughkeepsie, New York 12602**

©Copyright IBM Corp. 1985

Printed in U.S.A.

MVS Diagnostic Techniques
SY28-1133-2

READER'S
COMMENT
FORM

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Note: ***Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.***

Possible topics for comment are:

Clarity   Accuracy   Completeness   Organization   Coding   Retrieval   Legibility

If you wish a reply, give your name, company, mailing address, and date:

____

____

____

____

What is your occupation? ______________________________

How do you use this publication? __________________________

Number of latest Newsletter associated with this publication: ______________

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.)

Note: Staples can cause problems with automated mail sorting equipment.
Please use pressure sensitive or other gummed tape to seal this form.

Cut or Fold Along Line

MVS Diagnostic Techniques

SY28-1133-2

S370-37

Reader's Comment Form

Cut or Fold Along Line

Fold and tape

Please Do Not Staple

Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Department D58, Building 921-2
PO Box 390
Poughkeepsie, New York 12602

Fold and tape

Please Do Not Staple

Fold and tape

Printed in U.S.A.

IBM®

MVS Diagnostic Techniques

SY28-1133-2 S370-37

Printed in U.S.A.