GC28-0725-2
File No. S370-37

Systems

OS/VS2 System Programming Library: MVS Diagnostic Techniques

Release 3.7

Includes Selectable Units:

Scheduler Improvements	VS2.03.804
Supervisor Performance #1	VS2.03.805
Supervisor Performance #2	VS2.03.807
Service Data Improvements	VS2.03.817
JES2 Release 4.1	5752-825
3838 Vector Processing Subsystem Support	5752-829
Dumping Improvements	5752-833
Attached Processor System for Models 158/168	5752-847
Hardware Recovery Enhancements	5752-855
Interactive Problem Control System (IPCS)	5752-857

Third Edition (September, 1978)

This is a major revision of, and obsoletes, GC28-0725-1 incorporating changes released in the following System Library Supplement:

Interactive Problem Control System (IPCS)	5752-857	GD23-0095-0 (dated March 31, 1978)

See the Summary of Amendments following the Contents for a summary of the changes that have been made to this manual. A vertical line to the left of the text or illustration indicates a technical change made in this edition; revision bars are not used, however, to indicate changes made in previous editions, technical newsletters, or supplements.

This edition applies to release 3.7 of OS/VS2 and to all subsequent releases of OS/VS2 until otherwise indicated in new editions or Technical Newsletters. Changes are continually made 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.

Publications are not stocked at the address given below; requests for IBM publications should be made to your IBM representative or to 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, Publications Development, Department D58, Building 706-2, PO Box 390, Poughkeepsie, NY 12602. IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

Guide for Using This Publication

The following is a list of the requirements for using this publication.

- This publication contains information for the following Selectable Units:

  Scheduler Improvements – SU4
  Supervisor Performance # 1 – SU5
  Supervisor Performance # 2 – SU7
  Service Data Improvements – SU17
  JES2 Release 4.1 – SU25
  3838 Vector Processing Subsystem Support – SU29
  Dumping Improvements – SU33
  Attached Processor System for Models 158/168 – SU47
  Hardware Recovery Enhancements – SU55
  Interactive Problem Control System (IPCS) – SU57

- To use this publication, you must have installed at least SUs 4, 5, 7, 17, 25, (if you are a JES2 user), 33, and 55.

- The implied date of this publication, for the purpose of adding new supplements/TNLs, is September 30, 1978. Always use the page with the latest date (shown at the top of the page) when adding pages from different supplements/TNLs.

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.

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. – 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
OS/VS2 MVS Interactive Problem Control System (IPCS) User's Guide and Reference	GC34-2006
OS/VS Environmental Recording Editing and Printing (EREP) Program	GC28-0772
OS/VS2 System Programming Library:	
Initialization and Tuning Guide	GC28-0681
Supervisor	GC28-0628
Job Management	GC28-0627
Service Aids	GC28-0674
SYS1.LOGREC Error Recording	GC28-0677
Debugging Handbook (2 volumes)	GC28-0751 and GC28-0752
JES3 Debugging Guide	GC28-0703
OS/VS2 TCAM System Programmer's Guide, TCAM Level 10	GC30-2051
OS/VS TCAM Debugging Guide, TCAM Level 10	GC30-3040
OS/VS2 MVS VTAM Debugging Guide	GC27-0023
Operator's Library:	
OS/VS2 MVS System Commands	GC38-0229
OS/VS2 MVS JES2 Commands	GC23-0007
OS/VS2 MVS JES3 Commands	GC23-0008
VTAM Network Operating Procedures	GC27-6997
OS/VS TCAM Level 10	GC30-3037
OS/VS Message Library:	
VS2 System Messages	GC38-1002
VS2 System Codes	GC38-1008
3704/3705 Program Reference Handbook	GY30-3012
OS/VS2 I/O Supervisor Logic	SY26-3823
OS/VS2 System Initialization Logic	SY28-0623
OS/VS2 VSAM Logic	SY26-3825
OS/VS2 Catalog Management Logic	SY26-3826
OS/VS2 VTAM Data Areas	SY27-7267
OS/VS2 Access Method Services Logic	SY35-0010
OS/VS2 VTAM Logic	SY28-0621
OS/VS2 System Logic Library (7 volumes)	SY28-0713 through SY28-0719

OS/VS2 CVOL Processor Logic	SY35-0011
OS/VS2 MVS JES2 Logic	SY24-6000
OS/VS2 VIO Logic	SY26-3834
OS/VS2 MVS JES3 Logic	SY28-0612
OS/VS2 TCAM Level 10 Logic	SY30-3032
IBM 3704 and 3705 Communications Controllers NCP/VS Logic	SY30-3013
OS/VS2 Data Areas (microfiche)	SYB8-0606
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 Controllers NCP/VS Generation and Utilities Guide and Reference Manual	GC30-3007

Contents

Summary of Amendments for GC28-0725-2 VS2 Release 3.7

Changes have been made throughout this publication to reflect a Service Update to OS/VS2 Release 3.7 and to include the following topics:

Diagnostic Aids Information

Information from *OS/VS2 System Logic Library, Volume 7,* SY28-0719, was added in the following topics:

- Started task control (STC) abend and reason codes.
- Scheduler work area (SWA) manager reason codes.
- Auxiliary storage manager (ASM) diagnostic aids and serialization information.
- Allocation/unallocation reason codes.
- TSO logon scheduling.
- Communications task overview and diagnostic aids.
- DIDOCS diagnostic aids.

Also, diagnostic aids information was added for:

- Error recovery procedures (ERPs).
- Converting virtual addresses to real addresses.
- JES2 miscellaneous hints.

Interactive Problem Control System (IPCS), SU57

Overview information was added for IPCS.

Miscellaneous Changes

Throughout the text:

- Minor technical and editorial changes were made.
- References to DSS (dynamic support system) were removed.
- References to EREP0 were changed to EREP1 (environmental recording editing and printing).

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 system.
- ***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 reuseable resources. (In this manual, MP refers to multiprocessing on both multiprocessors and attached processors.)
- ***Locking Mechanism*** which facilitates Enabled System Design and Multi-processing 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 large number of new 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. Unfortunately, the latter is more often the case.

- ***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. The program check message gives the location of the failing operation and the condition code. 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 *OS/VS2 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 Multi-Processing on Problem Analysis
- MVS Trace Analysis
- Miscellaneous Debugging Hints
- Additional Data Gathering Techniques

2.1.2 OS/VS2 System Programming Library: MVS Diagnostic Techniques

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 Standalone Dumps:

   - 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 the other processor requesting a STORE STATUS operation. As a result, these values in the unprefixed PSA are *overlaid* by the second processor's values.

     Therefore, in an MP environment the status in the unprefixed PSA is always that of the 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 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" appears 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
	– 00001201	SIGP-emergency signal
	– 00001202	SIGP-external call

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'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.

6. **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.

7. **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.

8. **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.

9. **PSA + X'2F8' (PSAHLHI)** – 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.)

10. **PSA + X'380' (PSACSTK)** – This is the address of the active recovery stack which contains the addresses 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'C34,' 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'10' into the stack is non-zero, also check for an SDWA address at X'44' 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.

    At this point you should understand each processor's current activity, any possible errors that have been detected by recovery, and the current system state or mode.

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.

Once you understand the system's history from a trace, LOGREC, and error viewpoint, you should examine the work to be done as your next step to understanding the problem.

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 or local service manager queues (GSMQ/LSMQ) can indicate that the dispatcher has not received control since these SRBs were scheduled. This is an unusual condition that should be investigated. It can also indicate that the CVT anchors for these queues have been inadvertently altered. This again is an error condition.

Elements on the 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 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 wish 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 or FFFFFFFF (the lock suspend/interrupt IDs), refer to the chapter on "Locking" later in this section for an explanation.

  Note: When holding a suspend lock, code can only be suspended because it attempts to obtain an unavailable higher suspend lock or because of a page fault. 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 (middle four digits) 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.)

- 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.

- ASCBSRBS (ASCB + X'76') – number of SRBs active (currently executing or 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'7C') – number of ready TCBs.

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

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 started task control 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. If an explicit WAIT is not obvious, consider the following suspension possibilities and their associated key indicators:

1. If ASCBLOCK = X'7FFFFFFF' or X'FFFFFFFF', the status (registers and PSW) of the suspended or interrupted task is saved in the IHSA (ASCB + X'6C' points to ASXB; ASXB + X'20' points to IHSA). 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, find out of the state of that page fault. 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 interrupt, the TCB active indicators and the TCBCPUID (last processor on which this task was dispatched) is set on. If no TCB in the task structure has these indicators set, you can assume an SRB owned the lock. If no SRBs are on the CMS suspend queue, the suspension is probably the result of a page fault.

   An SRB can be suspended because of a page fault or a request for an unavailable suspend lock. The save area for the suspended SRB is the SSRB (see the *Debugging Handbook*). If suspended for page fault processing, the SSRB is pointed to by the corresponding PCB+1C. 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 CMS lock request, the SSRB is on the CMS lock suspended queue. See the chapter on "Waits" in Section 4 for details on how to locate the SSRB. For Local lock suspensions, the SSRBs are chained together on a queue anchored in the ASCB (ASCB + X'84').

   A locked TCB can be suspended for the same reasons as an SRB. The save area is the IHSA (described in the *Debugging Handbook*). The IHSA is valid during a page fault if the corresponding PCB+8 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 CMS lock suspension if the ASCB is on the CMS lock suspend queue at label CMSASBF in IEANUC01. The TCB can be suspended because of a page fault while holding both the local and CMS locks. One way to tell is that the ASCB+X'67' flag for the CMS lock is turned on and the ASCB address is in the CMS lockword.

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.)

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
- Page fault 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.

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:

1. Task mode
2. SRB mode
3. Physically disabled mode
4. 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.

Task Mode

Task mode describes code that is executing in the system because the dispatcher selected work from the task control block (TCB) chain. To start execution, the dispatcher sets up the environment (registers and PSW) and then passes control to the code to be executed. The registers and PSW are found in one of two places:

1. In the TCB at TCBGRS (TCB+X'30'), which is a register save area used when unlocked, enabled TCB mode work is interrupted. The PSW is obtained from the request block (RB) that is found through the TCB+0.
2. In the IHSA (interrupt handler save area), which is used to save registers when locally locked task mode code is interrupted. IHSA is found through ASXB+X'20'; ASXB is found through ASCB+X'6C'. The PSW for locally locked tasks is obtained from the IHSA.

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 finds an SRB on one of the SRB queues. SRB set-up is started by the SCHEDULE macro. SCHEDULE is an in-line macro that places the requestor-furnished SRB on one of two service queues, local or global, depending on the requestor's specification. These queues can be found from the CVT at CVTGSMQ (CVT+X'264'), which contains the address of the global service manager queue, or at CVTLSMQ (CVT+X'268'), which contains the address of the local service manager queue. Whenever the dispatcher finds work on either queue, the SRBs are moved to the corresponding system priority list queue. The global system priority list queue (GSPL), which contains globally scheduled SRBs, is found from the CVT at CVTGSPL (CVT+X'26C').

There is also one local system priority list queue (LSPL) per address space. Each LSPL, which is found from the ASCB at ASCBSPL (+X'1C'), contains all SRBs locally scheduled by the requestor and also those SRBs that were globally scheduled when the targeted address space was swapped out.

SRBs are selected from these LSPLs 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 (CMS or local) is unavailable.

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 also always 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.

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.

For programs executing in locked mode, status is saved according to the condition surrounding the programs, as follows:

Locally locked task is interrupted. A new area, the IHSA interrupt handler save area (IHSA), has been defined in MVS to contain the status when a locally locked task is interrupted. The IHSA is found from the ASCB + X'6C,' which points to the ASXB; the ASXB + X'20' points to the IHSA.

Locally locked SRB is interrupted. When locally locked SRBs are interrupted, there is no problem because SRBs are non-preemptable. The registers and PSW are saved in the LCCA. When the system has handled the interrupt, the SLIHs return to the FLIHs, the status is restored from the LCCA, and control is returned to the interrupted SRB routine.

Locally locked SRB is suspended. Locally locked 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), which will contain the interrupted status and will also serve as the control block 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.

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 are two bytes of important dispatcher flags in the LCCA + X'21C'. At location X'21D', the LCCADSRW flag is turned on just prior to any LPSW (Load PSW) for a global SRB, a Local SRB, or task dispatch. For a global SRB, the LCCAGSRB and LCCASRBM flags are also set on. For a Local SRB, only the LCCASRBM flag is set on in addition to LCCADSRW.

PSA Indicators

- *Super Bits* – Flags in the supervisor control word located at PSA + X'228' indicate whether the dump was taken while in one of the interrupt handlers or dispatcher.

- *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. 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 the dispatcher dispatched an address space or a global SRB was dispatched. 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'1C' – Address of the local service priority list, which contains SRBs queued for dispatching.

X'66-67' – RCT flags.

X'72-73' – Non-dispatchability flags.

X'76' – Count of SRBs dispatched in this address space.

X'7C' – Number of ready TCBs in this address space.

X'80' – Local lock (see the chapter on "Locking" later in this section for how to interpret this field when ≠ 0).

X'84' – Address of the SRB suspend queue for unavailable local lock requestors.

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 (task mode only) – Initially:

  registers saved at LCCA+X'380' (LCCASGPR)

  Then for Type 1 and Type 4 SVCs:

  registers moved to TCB+X'30' (TCBGRS)

  PSW moved from PSA to requestor's RB

  Then for Type 2, 3, and 4 SVCs:

  Registers moved to SVRB

  PSW moved from PSA to requestor's RB

  I/O FLIH – Initially:

  registers saved at LCCA+X'1C0' (LCCAGPGR)

  PSW saved at LCCA+X'200' (LCCAIOPS)

  Then for unlocked tasks:

  Registers moved to TCB

  PSW moved to RB

For locked tasks (CMS or local):

registers moved to IHSA — ASCB+X'6C' → ASXB; ASXB+X'20' → IHSA

PSW moved to IHSA

For SRBs: registers remain in LCCA

PSW remains in LCCA

External FLIH – Initially:

registers saved at LCCA+X'A0'	(LCCAXGR1)

Then for recursion purposes:

registers moved to LCCA+X'E0'	(LCCAXGR2)
PSW is in PSA+X'240'	(PSAEXPS1)

If first recursion:

registers moved from LCCA+X'A0'	(LCCAXGR1)
to LCCA+X'120'	(LCCAXGR3)
PSW is in PSA+X'248'	(PSAWXPS2)

If second recursion:

registers moved to LCCA+X'A0', where they stay	(LCCAXGR1)
PSW is in PSA+X'18'	(FLCEOPSW)

Note: Subsequent status manipulation for tasks and SRBs is the same as for the I/O FLIH (that is, the movement from LCCA to TCB or IHSA is identical).

Program check – Initially:

registers saved at LCCA+8	(LCCAPGR1)

Then:

registers moved to LCCA+X'48'	(LCCAPGR2)
PSW is in LCCA+X'88'	(LCCAPPSW)

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.

Note: For SRB code, status is not moved from the LCCA save areas. SRBs are non-preemptable and are given control back immediately, with the status being restored from the LCCA.

- ***Locally Locked Task Suspension:*** Status saving is the same as for locked task interruptions (described earlier under "I/O FLIH") except that IHSA also contains the floating point registers, the FRR stacks, and the PSW. The ASCBLOCK field is updated to contain X'7FFFFFFF'.

- ***SRB Suspension:*** An SRB can be suspended in two 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 re-queued 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.

  The second 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 a local lock that is unavailable, the SSRB is queued from the ASCB. For a request for an unavailable CMS lock, the SSRB is queued on the CMS suspend queue header. (For more detail see the chapter on "Locking" later in this section.) In both cases of SRB suspension, resumption is at the appropriate entry in the lock manager to try to acquire the lock. Upon release of the CMS lock by the holder, any SSRBs are rescheduled. Upon release of the local lock by the holder, the first SSRB that was suspended is given the local lock and rescheduled.

  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'24') for a private area page.

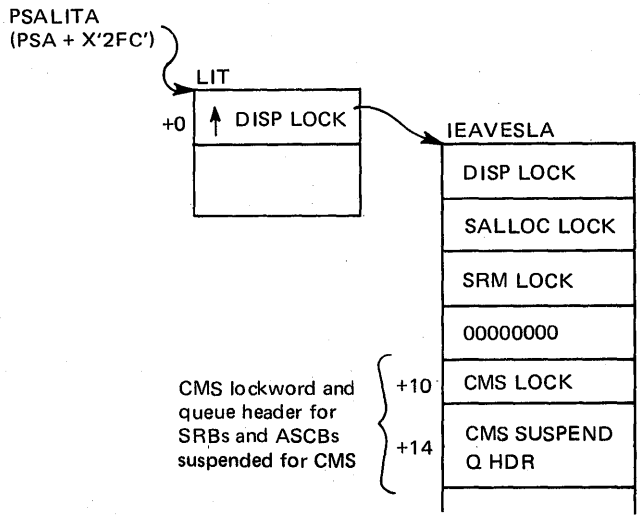
  PCB+X'1C' points to SSRB.

  Local Lock Requests

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

  CMS Locked

  SSRB is queued from the CMS SRB suspend queue in IEAVESLA as shown:



2.2.8 OS/VS2 System Programming Library: MVS Diagnostic Techniques

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 central lock manager 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.

Classes of Locks

MVS locks are divided into two classes:

- *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 dispatcher 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 lock, are global locks.

Locking (continued)

Name	Description
DISP	Global dispatcher lock – serializes all functions associated with the dispatching queues.
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.
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.
LOCAL	Local storage lock – serializes functions and storage within a local address space. There is one lock per address space.
Note: Locks are listed in hierarchical order, with DISP being the highest lock 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 cleared by the other processor. The requesting processor enters a loop in the lock manager (IEAVELK) that keeps testing the lock until the other processor releases it. As soon as the resource is free, the first processor can obtain the resource and continue processing.

- Suspend locks – prevent the requesting program 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, the highest priority queued requestor is given control of the lock, except in the case of the local lock. Upon release of the local lock, the first SSRB will be given the lock and rescheduled.

Combining classes and types of locks provide three categories of locks:

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. 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 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 CMS (cross memory services) lock is the only supported global suspend lock. The local lock must be held in order to obtain the CMS lock. An unconditional request for the CMS 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.

The local lock must be held by the caller when requesting the CMS lock. Also, the local lock cannot be released while holding the CMS 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.

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 PSAHLHI (PSA highest lock held indicator). 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 corresponding processor. Figure 2-2 shows the bit assignments.

(*Note:* When a holder of a CMS or local lock is suspended, the corresponding bit in the PSAHLHI field is reset to 0 even though the lock is still held.)

PSAHLHI (location X'2F8' in PSA)

2FA	*2FB*	
10	00	DISP
08	00	ASM
04	00	SALLOC
02	00	IOSYNCH
01	00	IOSCAT
00	80	IOSUCB
00	40	IOSLCH
00	20	not assigned
00	10	not assigned
00	08	not assigned
00	04	SRM
00	02	CMS
00	01	LOCAL

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 type of lock they describe:

Global Spin Lockword

X'00000000' – Lock is available.

X'00000040' – Lock is held on processor 0.

X'00000041' – Lock is held on processor 1.

Global Suspend Lockword (CMS Lock)

X'00000000' – Lock is available.

X'00xxxxxx' – ASCB address of owner of lock. If an address space owned the CMS lock but was interrupted or suspended, the ASCBCMSH flag in ASCBFLG1 is turned on and the CMS lock-held bit in PSAHLHI is turned off until the address space is redispatched. The ASCB address remains in the CMS lock until it is released.

Local Suspend Lockword (Local Lock)

X'00000000' – Lock is available.

X'00000040' – Lock is held on processor 0.

X'00000041' – Lock is held on processor 1.

X'7FFFFFFF' – Task or SRB suspended while holding the lock. The reason for suspension is either a page fault or an unconditional request for the CMS lock while it was unavailable.

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

- A page fault has been resolved for a locked task or SRB.
- The CMS lock, at one time unavailable, is now available.
- A higher priority address space was given control over this locked task.

How To Find Lockwords

Lockwords for single system locks are located in a table called IEAVESLA (PSA + X'2FC' points to the lock interface table (LIT); LIT + 0 points to IEAVESLA). They 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 location of all the lockwords are shown in Figure 2-3. Note that all lockwords must reside in fixed common storage.

Lock Name	Class	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+4	
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+8	
CMS	Global	Suspend	1	IEAVESLA+X'10'	
LOCAL	Local	Suspend	1 per address space	ASCB+X'80'	
*PSA+X'2FC' points to the lock interface table; the lock interface table +0 points to IEAVESLA.					

Figure 2-3. Classification and Location of Locks

Results of Requests for Unavailable Locks

Global Spin Locks – An unconditional request for a global spin lock results in a disabled loop in IEAVELK. In this case, register 11 contains the address of the requested lock and register 14 contains the address of the requestor.

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. *Note:* The dispatcher will not dispatch any task in the address space other than the holder of the lock until the lock is released.

SRBs requesting an unavailable local lock are suspended. In each case, the lock manager obtains an SSRB and places the GPRs and the current FRR stack there.

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 on the local lock suspend queue are chained together at SRB+4.
2. When interrogating a given address space, if the ASCBLOCK field is not 0, check the ASCBLSQH to determine the SRB work being delayed in this address space because of lock contention.

CMS Lock – Tasks unconditionally requesting the CMS lock when it is unavailable are suspended. For each task:

- GPRs are saved in the IHSA which is pointed to from ASXB + X'20'.
- The resume PSW is set to re-enter the lock manager.
- The ASCB is queued on the CMS suspend queue. (The first element of the CMS suspend queue is anchored in CSECT IEAVESLA + X'14'; this anchor points to either an SSRB or an ASCB which is suspended for the CMS lock. There is only one queue for suspended CMS lock requesters.) *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.

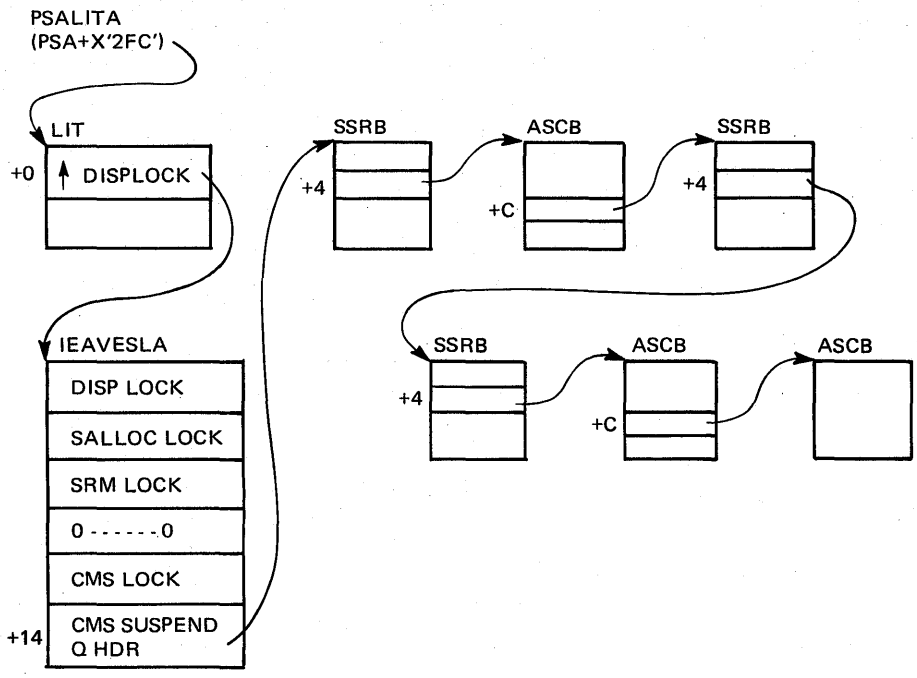
The address spaces suspended on the CMS lock are represented by the ASCBs on the CMS suspend queue. The ASCBs are chained together at the field ASCBCMSF (forward pointer).
Note: When an ASCB is on the CMS suspend queue, the ASCBLOCK contains X'7FFFFFFF'.

When the CMS lock is released, the ASCBLOCK is changed to X'FFFFFFFF', which indicates that work was interrupted but it is now ready to be resumed.

SRBs unconditionally requesting the CMS lock when it is unavailable are suspended. For each SRB, the lock manager:

- Obtains an SSRB from SQA
- Saves GPRs and the FRR stack in the SSRB
- Sets ASCBLOCK to X'7FFFFFFF'
- Chains the SSRB on the CMS SRB suspend queue located in IEAVESLA (IEAVESLA + X'14')

Note: Since there is only one queue for suspended CMS lock requesters, the SSRBs and ASCBs are chained on the CMS suspend queue using either ASCBCMSF (ASCB + X'C') or SRBFLNK (SSRB + 4). There are no backward pointers. Thus the CMS suspend queue could have the following appearance:



Use of Recovery Work Areas For Problem Analysis

Recovery processing, which is unique to MVS, enhances the reliability of the 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 and to take a stand-alone dump. 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 conrol 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 help 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.

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 whenever they are entered. 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 the IFCEREP1 service aid as described in *OS/VS Environmental Recording Editing and Printing (EREP) Program.* (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.) IFCEREP1 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. IFCEREP1 will put the variable area, the last X'6C' bytes of each SDWA, in an alphameric or hexadecimal format, whichever is specified. This variable area is

used by the recovery routines to construct messages and to provide data that often contains valuable debugging information.

There are five different types of software incidents for which the failure is written to SYS1.LOGREC. They are:

1. ABEND (SVC 13)
2. Invalid SVC
3. MCH software recovery attempt
4. Program check
5. Restart key depressed

SYS1.LOGREC Records

This section contains examples and explanations of three different types of error records that you can obtain from SYS1.LOGREC.

SYS1.LOGREC Software Incident Record 1

Figure 2-4 is an example of the data that is recorded in SYS1.LOGREC when a software (source) entry is recorded as the result of an SVC 13. The following explanations are called out by Notes A-E in the example:

Note A: The CSECT name is IDAVBPP1; it can be found in module IDDWI. IDAVBPR1 is the FRR that processed the error under consideration. The EC PSW indicates that SVC D was issued at location X'F4BB64'.

Note B: Approximately midway into the formatted record shown, you find more specific information about why this particular LOGREC entry was made. Note B points out three bits that reflect the status of the system at the time this failure was detected:

- SVC was issued by a locked or SRB routine.
- Logically disabled (physically disabled, locked, or SRB) routine was in control.
- Type 1 SVC routine was in control.

Note C: For this LOGREC record there is no formatted entry for the system completion code. Only a portion of the recorded software incidents are assigned a system completion code. A system completion code can be found at X'04' bytes into the SDWA, which can be found unformatted at the bottom of the record. Also, if the cause of a record is an abend SVC, a completion code is contained in register 1 under "Regs at time of error". The system completion code for this failure is 0E3.

```
                                                                   DATE      TIME         CPU      CPU       RELEASE
                                                                   DAY YR  HH MM SS.TH    SERIAL   ID        LEVEL
--- RECORD ENTRY SOURCE - SOFTWARE ---  TYPE   SOFTWARE(SVC 13)    117 75  20 31 01 93    023782   0158      VS 2 REL. 03
        ERRORID=SEQ00056 CPU0040 ASID0002 TIME 20.31.01.0
JOBNAME                           NW01ADP2
ABENDING PROGRAM NAME             N/A            BC MODE PSW AT TIME OF ERROR              BC MODE PSW OF LAST RB
NAME OF MODULE INVOLVED           IDDWI
NAME OF CSECT INVOLVED            IDAVBPP1           00000000 00000000                        00000000 00000000
FUNCTIONAL RECOVERY ROUTINE       IDAVBPR1

REGS AT TIME OF ERROR

REGS 0-7     00000000  000E3000  00F4EE9C  00000C9C  00CA73F4  00CA1F08  00100001  00CA7CD0
REGS 8-15    00000000  40F4B774  C0CC00C0  00CA7828  50F4B9B4  00CA7B64  50F4B8E6  00000004

EC PSW AT TIME OF ABEND     070C2000 00F4BB66        EC PSW FROM ESTAE RB(0 FOR ESTAI)    070C0000 00F4EE90
ADDITIONAL INFO:                                     ADDITIONAL INFO:
INST LENGTH CODE            02                       INST LENGTH CODE         02
INTERRUPT CODE              000D                     INTERRUPT CODE           000D
VIRT ADDR OF TRANS EXCEP    00202720                 VIRT ADDR OF TRANS EXCEP 00202720

REGS OF RB LEVEL OF ESTAE EXIT OR ZERO FOR ESTAI

REGS 0-7     00000000  00CE300C  00F4EE9C  00000C9C  00CA73F4  00CA1F08  00100001  00CA7CD0
REGS 8-15    00000000  40F4B774  CCC00000  00CA7828  50F4B9B4  00CA7364  50F4B8E6  00000004

   MCH FLAG BYTE                  MCK INPUT INFO                 FRAME ERROR INDICATORS          STORAGE ERROR INDICATORS
STORAGE ADDRS ARE VALID     0   STORAGE KEY FAILURE        0   STORAGE ERROR ALREADY SET 0   FRAME OFFLINE(OR SCHED)   0
MCK RECORD NOT RECORDED     0   REGISTERS UNPREDICTABLE    0   CHANGE INDICATOR ON       0   INTERCEPT                 0
TIME STAMP IS VALID         0   PSW UNPREDICTABLE          0                                 STORAGE ERROR PERMANENT   0
STORAGE IS RECONFIGURED     0   STORAGE DATA CHECK         0                                 PERMANENT RES. STORAGE    0
RECONFIGURE STATUS AVAIL    0   ACR REQUEST                0                                 FRAME IN SQA              0
RECONFIGURE NOT ATTEMPTED   0   INSTRUCTION FAILURE        0                                 FRAME IN LSQA             0
                                SOFT ERROR                 0                                 FRAME IS PAGE FIXED       0
                                TIMER ERROR                0                                 FRAME IS V=R              0
                                                               TIME STAMP OF ASSOCIATED MACHINE CHECK RECORD
BEGINNING VIRT ADDR OF STORAGE CHECK   00000000                         DATE      TIME
ENDING VIRT ADDR OF STORAGE CHECK      00000000                         DAY YR    HH MM SS.TH
REAL STORAGE FAILING ADDRESS           00000000                         000 00    00 00 00 00

MACHINE CHECK               0   TYPE 1 SVC IN CONTROL      1   PREV ESTA OR FRR FAILED   0   EXIT TO CLEANUP ONLY      0
PROGRAM CHECK               0   ENABLED RB IN CONTROL      0   (E)STAI PREV IN CONTROL   0   RB OF ESTA NOT IN CONTROL 0
RESTART KEY DEPRESSED       0   DISABLED RTN IN CONTROL    1   IRB PRECEDED RB           0   ESTA EXIT FOR PREV ABEND  0
TASK ISSUED SVC 13          0   SYSTEM IN SRB MODE         0   THIS RTN PERCOLATED TO    0   STEP ABEND REQUESTED      0
SYSTEM FORCED SVC 13        0                                  LOWER LEVEL EXIT INFO     0   TASK ANCESTOR ABENDED     0
SVC BY LOCKED OR SRB RTN    1                                                                REGS AND PSW UNAVAILABLE  0
TRANSLATION FAILURE         0                                                                MCK INFO UNAVAILABLE      0
PAGE I/O ERROR              0

                                CURRENT I/O STATUS
MEMORY ASID              0000   I/O IS RESTOREABLE         0
RECOVERY RETURN CODE       00   I/O IS NOT RESTOREABLE     0
                                NO I/O OUTSTANDING         0
                                NO I/O PROCESSING          0
```

Note A (points to NAME OF MODULE INVOLVED / NAME OF CSECT INVOLVED / FUNCTIONAL RECOVERY ROUTINE and to EC PSW AT TIME OF ABEND)

Note B (points to SVC BY LOCKED OR SRB RTN, TYPE 1 SVC IN CONTROL and DISABLED RTN IN CONTROL)

Note C (points to REGS 0-7 register 1, 000E3000)

Figure 2-4. SYS1.LOGREC Software Incident Record 1 (Part 1 of 2)

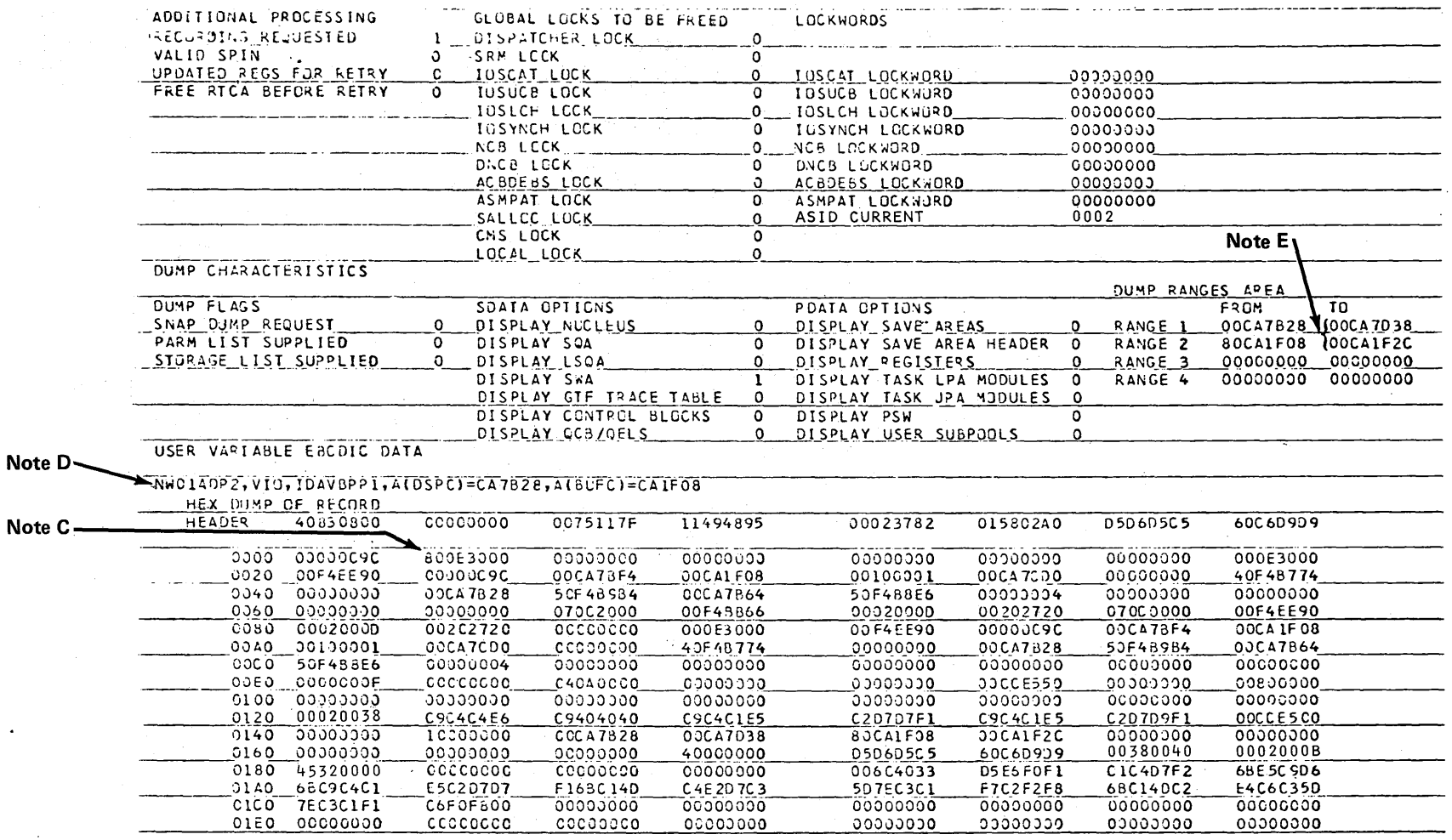


Figure 2-4. SYS1.LOGREC Software Incident Record 1 (Part 2 of 2)

Given the name of the module involved in the error, you can determine the id of the failing component by using the "Module Summary" section of the *Debugging Handbook.* This summary also names the corresponding PLM for each component. Component microfiche numbers are found in the "Component Summary" section of the *Debugging Handbook, Volume 1.*

Note D: The "Diagnostic Aids" section of the *OS/VS2 VIO Logic* describes the diagnostic output for module IDAVBPPI. It explains that the recovery routine sets starting and ending addresses for the DSPCT header and the BUFC in the SDWADPSL field of the SDWA. A diagnostic message is then built in the variable recording area in the SDWA (at X'194'). This message is formatted in the LOGREC record under the 'User Variable EBCDIC Data' field, just above the unformatted SDWA. (Also see Note P.)

Note E: The entries in the 'Dump Characteristics' section of this LOGREC record reflect the SDATA and PDATA options specified by the recovery routine for a SYSABEND, SYSMDUMP, or SYSUDUMP. All recovery routines can specify exactly what portions of storage are dumped. In addition, the recovery routines can specify a list of storage ranges that are to be dumped. In the dump for the failure in this example, the only area of storage displayed would be the SWA. A range of addresses would also be included. Range 1 is from CA7B28 to CA7D38; range 2 is from CA1F08 to CA1F2C.

In summary, from studying this LOGREC entry you find that the module IDAVBPP1 has detected an error and has issued a 0E3 ABEND, with a return code of 4. At the time of the failure, the system was logically disabled and a type 1 SVC was in control. SVC 13 was issued while the system was logically disabled, which is why the LOGREC entry was written. A functional recovery routine, module IDAVBPR1, was given control and tried to recover from the error. It was unsuccessful so it dumped the scheduler work area (SWA) and two sections of storage where the DSPCT and an important parameter list were located. The module then percolated to the next higher FRR in the stack. Note that the 'Recovery Return Code' field (SDWA + X'FC') = 00; this indicates percolation. A code of '04' indicates that retry was requested.

SYS1.LOGREC Software Incident Record 2

Figure 2-5 is another example of the type of data recorded in SYS1.LOGREC when a software incident occurs. Compare this example with record 1 in order to understand the different types of information that you can obtain from SYS1.LOGREC.

First compare the time stamp at the top of this record with that in record 1. These times are either identical or just a fraction of a second apart whenever the system is percolating through FRRs.

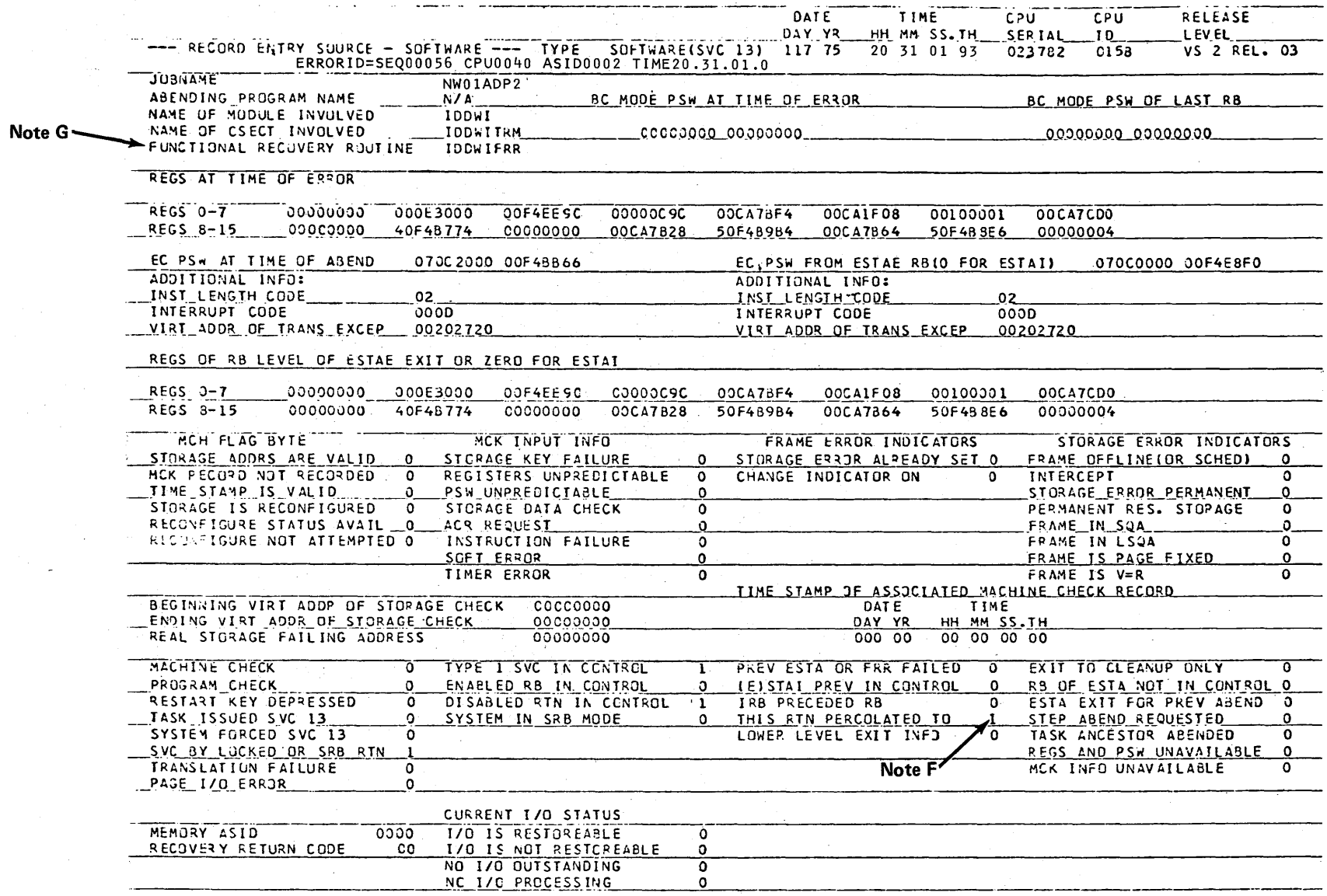


Figure 2-5. SYS1.LOGREC Software Incident Record 2 (Part 1 of 2)

```
ADDITIONAL PROCESSING               GLOBAL LOCKS TO BE FREED          LOCKWORDS
RECORDING REQUESTED           1     DISPATCHER LOCK               0
VALID SPIN                    0     SRM LOCK                      0
UPDATED REGS FOR RETRY        0     IOSCAT LOCK                   0   IOSCAT LOCKWORD          00000000
FREE RTCA BEFORE RETRY        0     IOSUCB LOCK                   0   IOSUCB LOCKWORD          00000000
                                    IOSLCH LOCK                   0   IOSLCH LOCKWORD          00000000
                                    IOSYNCH LOCK                  0   IOSYNCH LOCKWORD         00000000
                                    NCB LOCK                      0   NCB LOCKWORD             00000000
                                    DNCB LOCK                     0   DNCB LOCKWORD            00000000
                                    ACBDEBS LOCK                  0   ACBDEBS LOCKWORD         00000000
                                    ASMPAT LOCK                   0   ASMPAT LOCKWORD          00000000
                                    SALLOC LOCK                   0   ASID CURRENT             0002
                                    CMS LOCK                      0
                                    LOCAL LOCK                    0
DUMP CHARACTERISTICS
                                                                                     Note I    DUMP RANGES AREA
DUMP FLAGS                          SDATA OPTIONS                     PDATA OPTIONS                       FROM      TO
SNAP DUMP REQUEST             0     DISPLAY NUCLEUS               0   DISPLAY SAVE AREAS          0  RANGE 1  00CB2A28  00CB2C28
PARM LIST SUPPLIED            0     DISPLAY SQA                   0   DISPLAY SAVE AREA HEADER    0  RANGE 2  00CB2B00  00CB2B20
STORAGE LIST SUPPLIED         0     DISPLAY LSQA                  1   DISPLAY REGISTERS           0  RANGE 3  80CB2AC8  00CB2AF8
                                    DISPLAY SWA                   1   DISPLAY TASK LPA MODULES    0  RANGE 4  00000000  00000000
                                    DISPLAY GTF TRACE TABLE       0   DISPLAY TASK JPA MODULES    0
                                    DISPLAY CONTROL BLOCKS        0   DISPLAY PSW                 0
                                    DISPLAY QCB/QELS              0   DISPLAY USER SUBPOOLS       0
USER VARIABLE EBCDIC DATA                          Note H

JOBNAME=NW01ADP2,VIO LMOD=IDDWI,WIEXCP,A(IOB) = 00CB2B00,A(VDSCB)=00CA7E70
   HEX DUMP OF RECORD
   HEADER    40830800   CC0C0000   0075117F   11494895   00023782   015802A0   D5D6D5C5   60C6D9D9

      0000   00000C7C   800E3000   00000000   00000000   00000000   00000000   00000000   000E3000
      0020   00F4EE90   00000C9C   00CA7BF4   00CA1F08   00100001   00CA7CD0   00000000   40F4B774
      0040   00000000   00CA7B28   5CF4B9B4   00CA7B64   50F4B8E6   00000004   00000000   00000000
      0060   00000000   00000000   070C2000   00F4BB66   0002000D   00202720   070C0000   00F4E8F0
      0080   0002000D   002C2720   CC000000   000E3000   00F4EE90   00000C9C   00CA7BF4   00CA1F08
      00A0   00100001   00CA7CD0   CCCCCC00   40F4B774   00000000   00CA7B28   50F4B9B4   00CA7B64
      00C0   50F4B6E6   00000004   00000000   00000000   00000000   00000000   00000000   00000000
      00E0   0000000F   CCCCC000   04CA1000   00000000   00000000   00CCE550   00000000   00800000
      0100   00000000   00000000   00000000   00000000   00000000   00000000   00000000   00000000
      0120   0000C000   C9C4C4E6   C9404040   C9C4C4E6   C9E3D9D4   C9C4C4E6   C9C6D9D9   00CCE500
      0140   00000000   30000000   00CB2A28   00CB2C28   00CB2B00   00CB2B20   80CB2AC8   00CB2AF8
      0160   00000000   00000000   00000000   40000000   D5D6D5C5   60C6D9D9   00380040   0002000B
      0180   45340000   CCCCCC00   C0000000   00000000   006C404B   D1D6C2D5   C1D4C57E   D5E6F0F1
      01A0   C1C4D7F2   6BE5C9D6   40D3D4D6   C47EC9C4   C4E6C96B   E6C9C5E7   C3D76BC1   4DC9D6C2
      01C0   5D407E40   F0F0C3C2   F2C2F0F0   6BC14DE5   C4E2C3C2   5D7EF0F0   C3C1F7C5   F7F040C0
      01E0   00000000   CCCC0000   CC0000C0   00000000   00000000   00000000   00000000   00000000
```

Figure 2-5. SYS1.LOGREC Software Incident Record 2 (Part 2 of 2)

The following explanations refer to Notes F-I in Figure 2-5.

Note F: Look at the status bits that appear approximately midway through the example. One additional bit has been turned on in this entry that was off in record 1. This indicates that this routine received control through percolation (SDWA + X'EA' = X'10'). This indicator (that is, SDWA + X'EA') is not set when recovery processing goes from the last active FRR to the current ESTAE.

Note G: The name of the FRR in this example is IDDWIFRR. The completion code and the register contents are the same as in record 1.

Note H: Look at the 'User Variable EBCDIC Data' field. This area gives the location of two more control blocks that can be used in determining exactly what failed. These two control blocks are:

- IOB, located at address X'CB2B00'
- VDSCB, located at address X'CA7E70'

Note I: Compare the IOB and VDSCB addresses to the 'Dump Ranges Area' that have been specified. The VDSCB does not fall in one of these ranges because it is part of the SWA. Using the 'Diagnostic Aids' section of *OS/VS2 VIO Logic*, you can identify the other two dump ranges that are printed. Included in these ranges are the current channel program and the DEB.

SYS1.LOGREC Software Incident Record 3

Figure 2-6 illustrates a SYS1.LOGREC software (source) entry that has been recorded as a result of a program check (type). The following explanations refer to Notes J-N in Figure 2-6.

Note J: Because there is no completion code in register 1 for this type of entry, look for the completion code in SDWA+X'4' (in this case 0C9).

Note K: Check the status bits. They confirm the fact that the failure was a program check that occurred while an enabled RB was in control.

Note L: The 'Dump Characteristics' bits are on only if the functional recovery routine issues a SETRP macro with the DUMP=YES operand. This macro uses the SDWA to contain its dump options and these are the fields formatted in the LOGREC entry. Functional recovery routines can also take dumps by issuing the SDUMP macro. The SDUMP macro uses a different area for its dump options. You might receive a dump of certain failures even though the LOGREC 'Dump Characteristics' are zeros. Check the byte at displacement X'4' into the SDWA. This flag is turned on if a dump was requested by a SETRP, CALLRTM, or ABEND macro. As a general rule, ESTAE routines are the most common users of the DUMP and DUMPOPT operands of the SETRP macro. Since the 0C9 abend code in this LOGREC entry was for a problem program (an enabled RB in control), a dump would also be taken if the job had a SYSUDUMP, SYSMDUMP, or SYSABEND DD statement in its JCL.

Note M: There is a dump associated with this failure because location SDWA+4 (X'80') indicates a request for a dump. This can be seen from the unformatted record.

Note N: For this entry, the data in the variable recording area (at X'194' under 'Hex Dump of Record') is not formatted under 'User Variable EBCDIC Data'. This data is formatted by specifying an option (see Note P) in the individual recovery routine.

Note P: The two bytes of SDWA + X'190' specify the length of the variable recording area that starts at X'194'.

In the two bytes at SDWA + X'192': the first byte specifies how the routine wants its data in the variable recording area printed (X'80' for unformatted hexadecimal, X'40' for hexadecimal and formatted under 'User Variable EBCDIC Data'); the second byte gives the length of the data. It is often helpful while reading LOGREC entries to refer to the SDWA layout in the *Debugging Handbook* for additional information about individual bit settings.

```
                                                                              DATE       TIME          CPU      CPU      RELEASE
                                                                              DAY YR   HH MM SS.TH    SERIAL    ID       LEVEL
--- RECORD ENTRY SOURCE - SOFTWARE ---   TYPE    PROGRAM CHECK                117 75   20 31 01 50    023782    0158     VS 2 REL. 03
             ERRORID=SEQ00055 CPU0040 ASID001B TIME20.31.01.0
JOBNAME                              NW01ACP2
ABENDING PROGRAM NAME                N/A              BC MODE PSW AT TIME OF ERROR                   BC MODE PSW OF LAST RB
NAME OF MODULE INVOLVED              IGG0325A
NAME OF CSECT INVOLVED               IGG0325A              FF040009 40021CC0                          FE850020 50065890
FUNCTIONAL RECOVERY ROUTINE          IFGCRR0A

REGS AT TIME OF ERROR

REGS 0-7       00003230   00CB2E88   00CB2EE0   00CB2CB0   00E6313C   0000FFFF   FFFF7FFF   80007E70
REGS 8-15      00000000   00000000   00000000   00003230   0002FFCA   00CB2E88   60E6323E   00021C4E

EC PSW AT TIME OF ABEND     070C0000 00021CC0                 EC PSW FROM ESTAE RB(0 FOR ESTAI)    070C0000 00021CC0
ADDITIONAL INFO:                                              ADDITIONAL INFO:
INST LENGTH CODE            02                                INST LENGTH CODE          02
INTERRUPT CODE              0009                              INTERRUPT CODE            0009
VIRT ADDR OF TRANS EXCEP    00D44640                          VIRT ADDR OF TRANS EXCEP  00D44640

REGS OF RB LEVEL OF ESTAE EXIT OR ZERO FOR ESTAI

REGS 0-7       00003230   00CB2E88   00CB2EE0   00CB2CB0   00E631BC   0000FFFF   FFFF7FFF   80007E70
REGS 8-15      00000000   00000000   00000000   00003230   0002FFCA   00CB2E88   60E6323E   00021C4E

    MCH FLAG BYTE                   MCK INPUT INFO                   FRAME ERROR INDICATORS          STORAGE ERROR INDICATORS
STORAGE ADDRS ARE VALID   0   STORAGE KEY FAILURE         0   STORAGE ERROR ALREADY SET 0   FRAME OFFLINE(OR SCHED)    0
MCK RECORD NOT RECORDED   0   REGISTERS UNPREDICTABLE     0   CHANGE INDICATOR ON       0   INTERCEPT                  0
TIME STAMP IS VALID       0   PSW UNPREDICTABLE           0                                 STORAGE ERROR PERMANENT    0
STORAGE IS RECONFIGURED   0   STORAGE DATA CHECK          0                                 PERMANENT RES. STORAGE     0
RECONFIGURE STATUS AVAIL  0   ACR REQUEST                 0                                 FRAME IN SQA               0
RECONFIGURE NOT ATTEMPTED 0   INSTRUCTION FAILURE         0                                 FRAME IN LSQA              0
                              SOFT ERROR                  0                                 FRAME IS PAGE FIXED        0
                              TIMER ERROR                 0                                 FRAME IS V=R               0
                                                              TIME STAMP OF ASSOCIATED MACHINE CHECK RECORD
BEGINNING VIRT ADDR OF STORAGE CHECK    00000000                   DATE      TIME
ENDING VIRT ADDR OF STORAGE CHECK       00000000                   DAY YR    HH MM SS.TH
REAL STORAGE FAILING ADDRESS            00000000 Note K            000 00    00 00 00 00

MACHINE CHECK             0   TYPE 1 SVC IN CONTROL       0   PREV ESTA OR FRR FAILED   0   EXIT TO CLEANUP ONLY       0
PROGRAM CHECK             1   ENABLED RB IN CONTROL       1   (E)STAI PREV IN CONTROL   0   RB OF ESTA NOT IN CONTROL  0
RESTART KEY DEPRESSED     0   DISABLED RTN IN CONTROL     0   IRB PRECEDED RB           0   ESTA EXIT FOR PREV ABEND   0
TASK ISSUED SVC 13        0   SYSTEM IN SRB MODE          0   THIS RTN PERCOLATED TO    0   STEP ABEND REQUESTED       0
SYSTEM FORCED SVC 13      0                                   LOWER LEVEL EXIT INFO     0   TASK ANCESTOR ABENDED      0
SVC BY LOCKED OR SRB RTN  0                                                                 REGS AND PSW UNAVAILABLE   0
TRANSLATION FAILURE       0                                                                 MCK INFO UNAVAILABLE       0
PAGE I/O ERROR            0

                              CURRENT I/O STATUS
MEMORY ASID            0000   I/O IS RESTOREABLE          0
RECOVERY RETURN CODE     00   I/O IS NOT RESTOREABLE      0
                              NO I/O OUTSTANDING          0
                              NO I/O PROCESSING           1
```

Figure 2-6. SYS1.LOGREC Software Incident Record 3 (Part 1 of 2)

```
ADDITIONAL PROCESSING              GLOBAL LOCKS TO BE FREED          LOCKWORDS
RECORDING REQUESTED          1     DISPATCHER LOCK            0
VALID SPIN                   0     SRM LOCK                   0
UPDATED REGS FOR RETRY       0     IOSCAT LOCK                0      IOSCAT LOCKWORD           00000000
FREE RTCA BEFORE RETRY       0     IOSUCB LOCK                0      IOSUCB LOCKWORD           00000000
                                   IOSLCH LOCK                0      IOSLCH LOCKWORD           00000000
                                   IOSYNCH LOCK               0      IOSYNCH LOCKWORD          00000000
                                   NCB LOCK                   0      NCB LOCKWORD              00000000
                                   DNCB LOCK                  0      DNCB LOCKWORD             00000000
                                   ACBDEBS LOCK               0      ACBDEBS LOCKWORD          00000000
                                   ASMPAT LOCK                0      ASMPAT LOCKWORD           00000000
                                   SALLOC LOCK                0      ASID CURRENT              001B
                                   CMS LOCK                   0
                                   LOCAL LOCK                 0
DUMP CHARACTERISTICS
                                                                                          DUMP RANGES AREA
DUMP FLAGS                         SDATA OPTIONS                     PDATA OPTIONS                              FROM      TO
SNAP DUMP REQUEST            0     DISPLAY NUCLEUS            0      DISPLAY SAVE AREAS          0   RANGE 1   00000000  00000000
PARM LIST SUPPLIED           0     DISPLAY SQA                0      DISPLAY SAVE AREA HEADER    0   RANGE 2   00000000  00000000
STORAGE LIST SUPPLIED        0     DISPLAY LSQA               0      DISPLAY REGISTERS           0   RANGE 3   00000000  00000000
                                   DISPLAY SWA                0      DISPLAY TASK LPA MODULES    0   RANGE 4   00000000  00000000
                                   DISPLAY GTF TRACE TABLE    0      DISPLAY TASK JPA MODULES    0
                                   DISPLAY CONTROL BLOCKS     0      DISPLAY PSW                 0
                                   DISPLAY QCB/QELS           0      DISPLAY USER SUBPOOLS       0
  HEX DUMP OF RECORD
  HEADER    42830800  CCC0[illegible]008  0C75117F  11494869    00023782  015802A0  D5E6F0F1  C1C4D7F2

      0000  00CB2C90  EC0C9000  FF040009  40021CC0    FF850020  50065890  00003230  00CB2E88
      0020  00CB2EE0  00CB2CB0  00E6318C  0000FFFF    FFFF7FFF  80007E70  00000000  00000000
      0040  00000000  00003230  0002FFCA  00CB2E88    60E6323E  00021C4E  00CC0958  00000000
      0060  00000000  00000000  070C0C00  00021CC0    00020009  00044640  070C0000  00021CC0
      0080  00020009  00044640  00003230  C0CB2E88    00CB2EE0  00CB2CB0  00E6318C  0000FFFF
      00A0  FFFF7FFF  80007E70  00000000  00000000    00000000  00003230  0002FFCA  00CB2E88
      00C0  60E6323E  CCC21C4E  E6000248  00000000    00000000  00000000  00000000  00000000
      00E0  0000000F  00000000  40040000  00001000    00000000  00CA2F48  00000000  00800000
      0100  00000000  00000000  00000000  00000000    00000000  00000000  00000000  00000000
      0120  00000000  C9C7C7F0  F3F2F5C1  C9C7C7F0    F3F2F5C1  C9C6C7F0  D9D9F0C1  00CA2EF8
      0140  00000000  00000000  00000000  00000000    00000000  00000000  00000000  00000000
      0160  00000000  CC000000  00000000  00000000    [illegible]00000  00000000  00370000  001B000B
      0180  45300000  00000000  000000C0  00000000    006C8018  A0000000  00CB2A30  00EACCAC
      01A0  00000000  00000000  00CA2F88  C47EC9C4    C4E6C96B  E6C9C5E7  C3D76BC1  4DC9D6C2
      01C0  5D407E40  F0F0C3C2  F0C2F0F0  6BC14DE5    C4E2C3C2  5D7EF0F0  C3C2F2C3  C6F04000
      01E0  00000000  00000000  00000000  00000000    00000000  00000000  00000000  00000000
```

Note L → DUMP CHARACTERISTICS

Note N → HEX DUMP OF RECORD

Note M → header, third word (first byte)

Note J → header, third word (remaining bytes)

Note P → offset 0160/0180, fifth column

Figure 2-6. SYS1.LOGREC Software Incident Record 3 (Part 2 of 2)

Important Considerations About SYS1.LOGREC Records

As shown in the three incident 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:

- 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.

- "BC mode PSW at Time of Error, of Last RB" – You can ignore these fields.

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

  a. 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.

  b. 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.

  c. 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":

  a. 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".

  b. 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.

c. 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 because the ESTAE routine may be executing on a processor other than the mainline). 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.

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.

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 Retain for the CSECT, FRR, 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

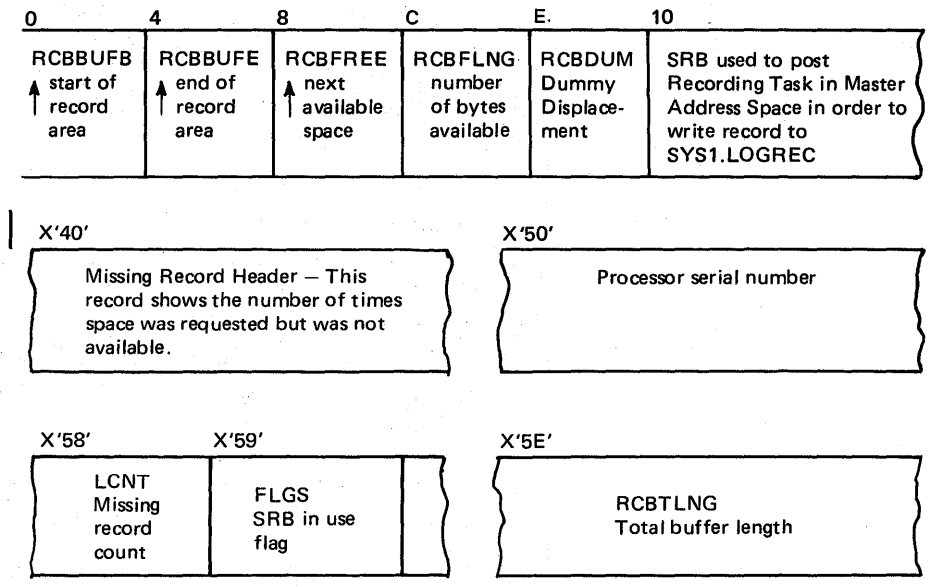
The in-storage LOGREC buffer can be formatted by specifying the LOGDATA verb under AMDPRDMP. This verb causes the entries still in the buffer to be formatted in the same manner as those printed from SYS1.LOGREC. For detailed information on how to invoke the AMDPRDMP service aid, see *OS/VS2 SPL: Service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

Finding the LOGREC Recording Control Buffer

The CVT + X'23C' points to the RTCT (recovery termination control table); and RTCT + X'20' points to the RTMRCB (LOGREC recording control buffer). The buffer always resides in SQA on a page boundary, is 4K bytes in length, and is generally located just beyond the trace table. Scanning the EBCDIC portion of the dump following the trace table usually leads you to a series of module/job names that are part of the individual records.

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 2-7 shows the format of the buffer and Figure 2-8 shows the format of individual records within the buffer.



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 2-7. 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 2-8. 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' points to the current recovery stack entry. (Unless the FRR stack +X'C' matches FRR stack +0, in which case no recovery is present on the stack.) This entry +0 points to the recovery routine that is to gain control in case of error. The entry +4 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 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 (at FRR stack +X'10') 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:

Use of Recovery Work Areas For Problem Analysis (continued)

1. FRR stack + X'10' – RTM 1 work area (RT1W)

   In the case of an error, the RT1W + 2 (FRR stack + X'12') field indicates the error type as follows:

   1 – program check
   2 – restart key
   3 – SVC error (SVC was issued while in locked, disabled, or SRB mode)
   4 – DAT error
   5 – machine check
   10 – paging I/O error
   11 – abnormal termination
   12 – branch entry to abnormal termination (compatibility interface)
   13 – cross memory abnormal termination
   15 – memory termination
   20 – MCH (machine check handler)

2. RT1W + X'34' (FRR Stack + X'44') – address of system diagnostic work area (SDWA)

   If no pointers can be found, the SDWA for each supervisor FRR stack can be found at X'20' bytes past the start of the last entry in the respective stack. (FRR +4 points to the last entry.) The SDWA for disabled errors on the normal stack is at X'330' bytes past the start of the last entry on the restart stack. (PSA +X'3B8' points to the restart stack.)

3. RT1W + X'40' (FRR stack + X'50') – 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'51').

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' (FRR stack + X'4C'), at TCBRTM12 (TCB + X'104'), or in the RTM2 SVRB at X'7C'. The EED is generally not present because RTM2 releases it early in its processing. The EED is described in the *Debugging Handbook* as part of the RT1W. Important EED fields are:

EED + 0 – pointer to next EED

EED + 4 (byte 0) – 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' – registers 0-15

EED + X'4C' – PSW/Instruction Length Code (ILC)/Translation Exception Address (TEA) at time of 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', 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 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.

The multiprocessing (MP) capability of MVS allows two 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 the other processor.

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 determined by the processor's prefix register.

Inter-Processor Communication – Malfunction alerts (MFA) are automatically generated by failing processors before entering the check-stop state. Other interprocessor 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.

Effects of Multiprocessing On Problem Analysis (continued)

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 – 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 attempts to take the failing processor offline so that system operation can continue with the remaining processor. (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 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 two, 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 hardware 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' points to the virtual address of the PSA for that processor

PCCA + X'1C' points to the real address of the PSA 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 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 interrupt (LCCA)
- The mode (SRB or task) of each processor (LCCA)
- The last program interrupt on each processor (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 characteristics of the MVS MP capability is parallelism. In looking at MP dumps, you must always remember that any two processes 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 undoubtedly 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) instruction
- Non-dispatchability
- Test and Set (TS) instruction
- RESERVE/RELEASE

Obviously all users of a particular SRR must use the same serialization mechanism. The integrity of an SRR is not enhanced if one user employs locking and another uses ENQ/DEQ. The point is to understand the processes going on in *both* processors at the time of the failure. The processor on which the failure occurred may *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 both 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 CS instruction is. CS serializes the word containing the byte against the other processor. The point is that in debugging an MP dump, *both* processors must be considered because interaction between processes and shared resources is generally the key to solving the problem.

When a program serializes an SRR incorrectly, other programs can alter the SRR before the first program completes its update. The other programs may be running on the other processor, or they may have received control on the same processor because the first program was pre-empted (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 window (an interval in which a program opens a serialization exposure is called a window).

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

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

Try to relate these two processes to the serialization problem that caused the dump. The existence of the window is confirmed by reading the code that alters the state of the SRR and finding where the two programs serialize improperly.

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 to both processors, tasks of different priority may be executing on both processors simultaneously.

2. The CHAP (change priority) SVC does not ensure that tasks are dispatched in the expected order when dispatching on two 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 storage from 0-4K 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 either processor unless they are scheduled with affinity for a particular processor.

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

9. Recovery routines can run on either processor, not necessarily the one on which the error was detected.

10. STATUS STOP 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 – uses the XC function of SIGP
   - Immediate service – uses 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 2-9.

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 2-9. 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 specifc 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 2-9). There are currently six functions that can be initiated via external call:

1. **Switch** – specifies that the service routine (IEAVEMSI) used by the memory/task switch function is to be executed.
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. **MF1TCH** – specifies that the MF1 service routine (pointed to by CVT + X'320') is to be executed. This routine executes TCH (Test Channel) instructions on the processor to which the channels are attached.

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'
MF1TCH	X'02'

Then IEAVERP sets the external call (XC) function code (X'02') in register 0 and uses the DSGNL macro instruction to cause the SIGP instruction to be issued. 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 2-10 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 2-9).

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 issues the DSGNL macro 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 2-11 shows the EMS process flow.

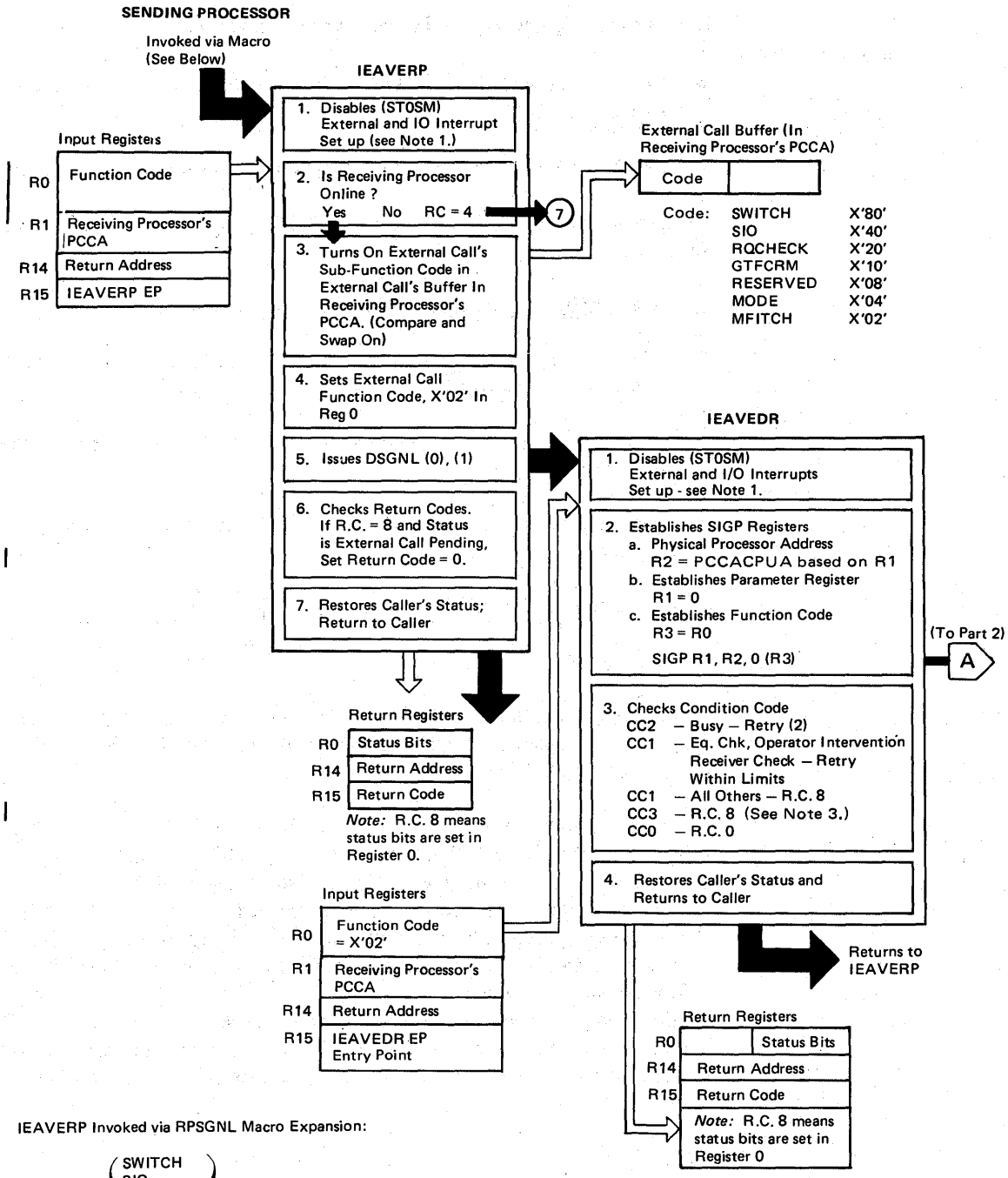
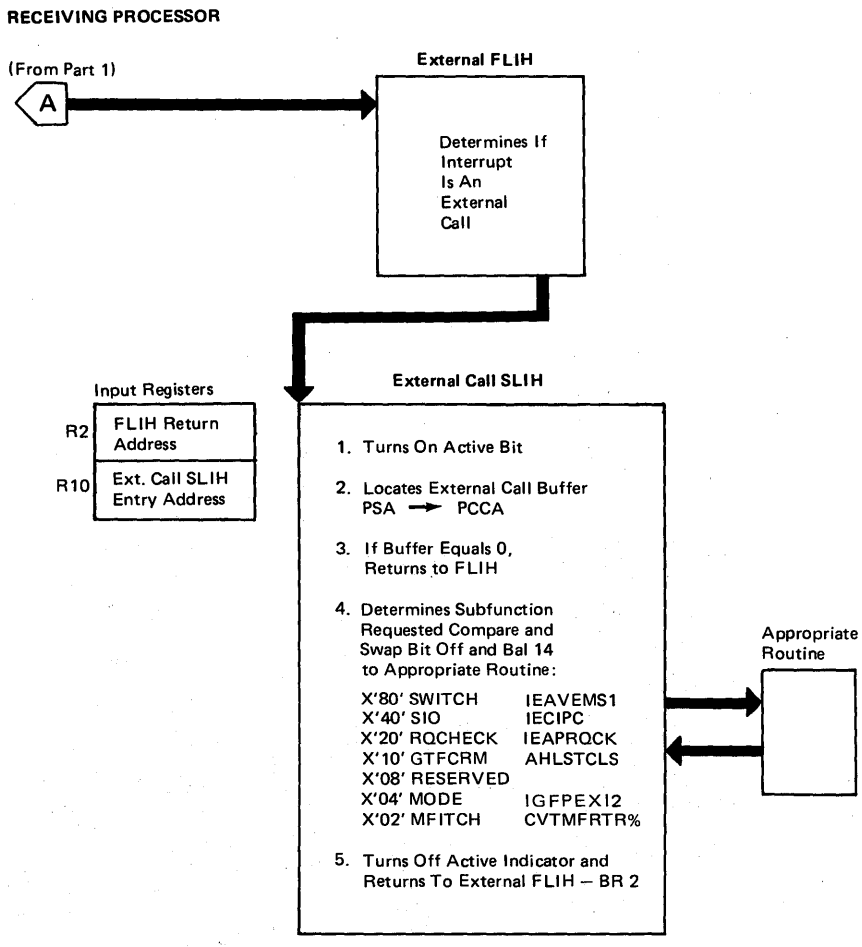


Figure 2-10. External Call (XC) Process Flow (Part 1 of 2)



Notes:

1. Turns on active indicator
   Saves callers registers
   Establishes addressability
2. Disables/Enables Spin
   1. Turns on SPIN indicator
   2. Enables for MFA and emergency signal interrupts
   3. Disables
   4. Turns off SPIN indicator
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.

Figure 2-10. External Call (XC) Process Flow (Part 2 of 2)

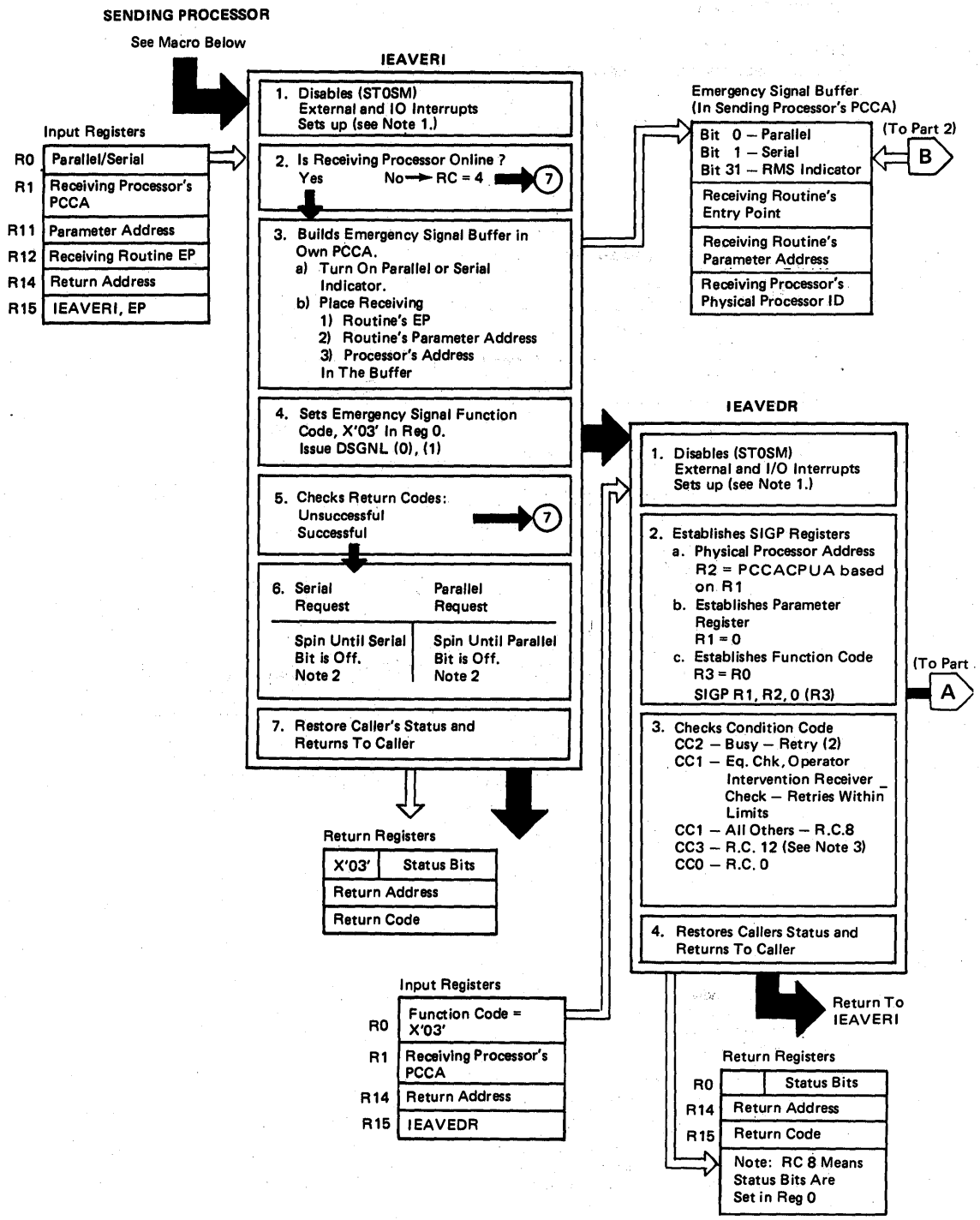
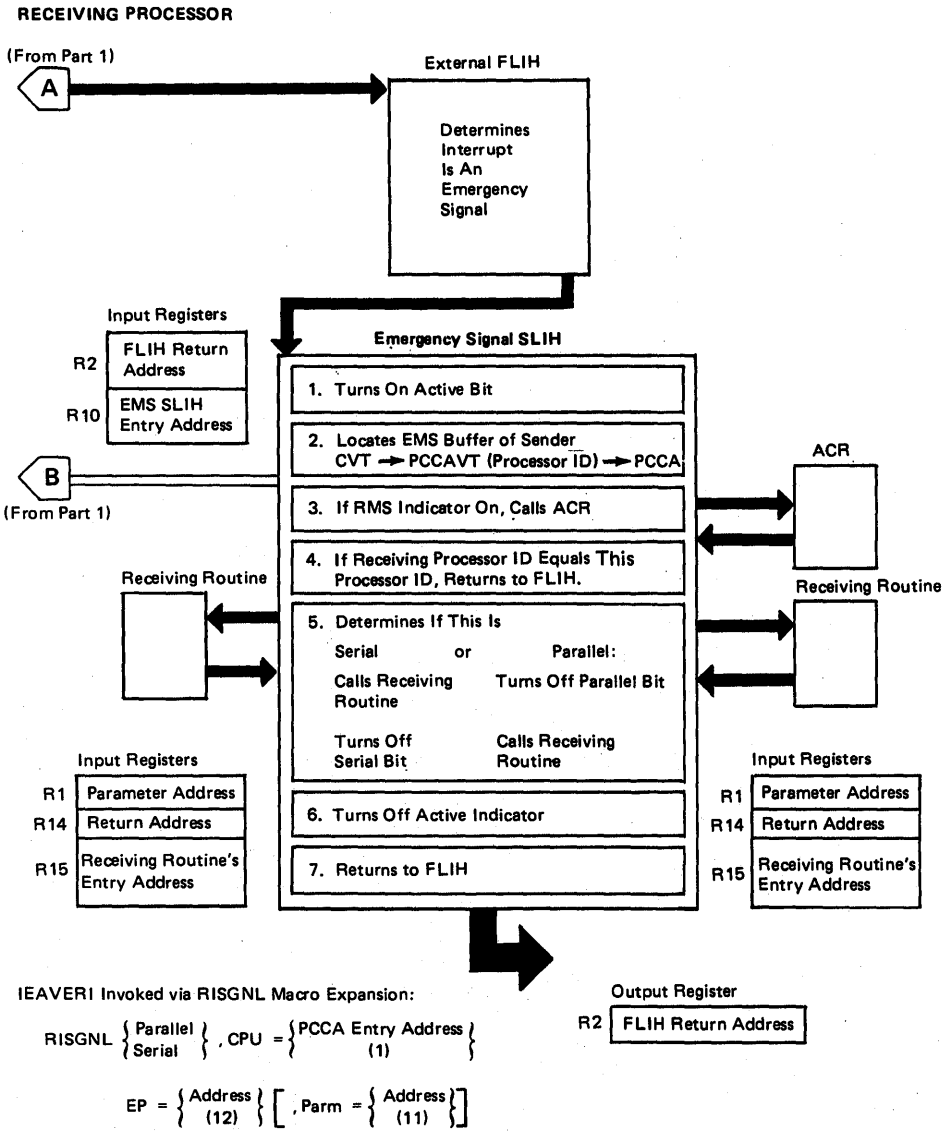


Figure 2-11. Emergency Signal (EMS) Process Flow (Part 1 of 2)



Notes:

1. Turns on active indicator
   Saves callers registers
   Establishes addressability
2. Disables/Enables Spin
   1. Turns on SPIN indicator
   2. Enables for MFA and emergency signal interrupts
   3. Disables
   4. Turns off SPIN indicator
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.

Figure 2-11. Emergency Signal (EMS) Process Flow (Part 2 of 2)

MP Debugging Hints

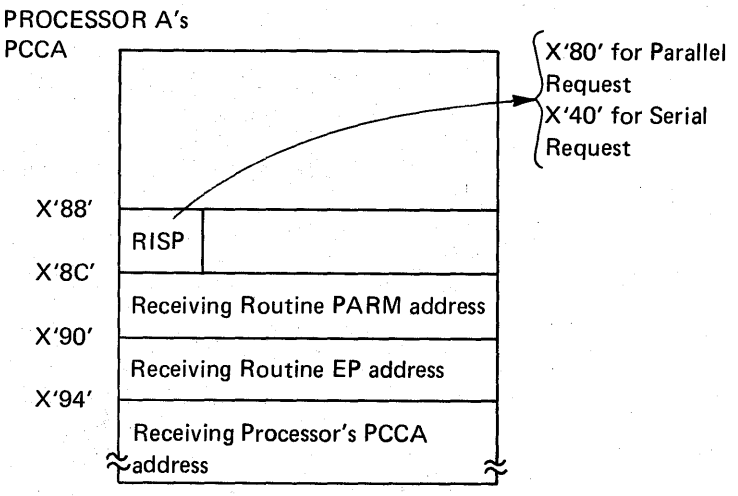
1. Apparent disabled loop in IEAVERI on processor A.

   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.

   CVT + X'2FC' points to the PCCAVT

   PCCAVT + 4 (CPUID for processor A) points to processor A's PCCA.

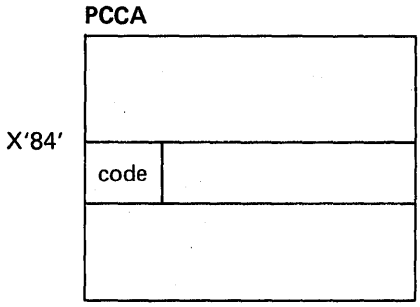


   By locating the proper PCCA (in this case processor A's), you can determine whether the EMS request was parallel or serial, the entry point, and, therefore, the name of the receiving routine. Although this information tells quite a bit about the current process on processor A, the real problem, however, is most likely on processor B. Three past experiences can help determine the state of processor B.

   - 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.
   - Processor B was stopped in order to take a stand-alone dump. Before the dump program was IPLed or processor A was stopped, processor A issued the EMS for page invalidation. Thus, when the dump occurred, processor A was looping while waiting for the page invalidation to complete. So it appeared that processor A's looping was the problem when actually it was caused by a previously-identified problem on processor B.

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 XC work to process, as indicated by the request codes below:



Request Code:

X'80'	–	SWITCH
X'40'	–	SIO
X'20'	–	RQCHECK
X'10'	–	GTFCRM
X'08'	–	RESERVED
X'04'	–	MODE
X'02'	–	MF1TCH

   The code is set in the *receiving* processor's PCCA so that a bit on in processor B's PCCA, for example, means that processor A initiated the request.

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.)

   IEEVCPU: 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 physcially online.

   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.

This chapter reviews the trace formats found in VS2 storage dumps. 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 Areas" section (see TTE-Trace Table 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 these formats in the following examples and descriptions.

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 2-12 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.)

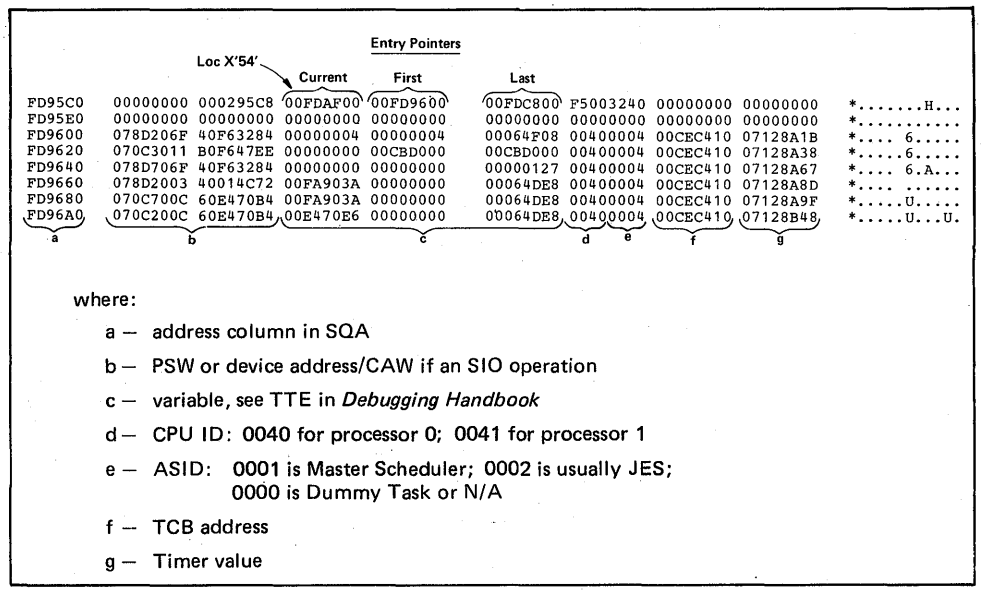


Figure 2-12. How to Locate the Trace Table

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 2-13 illustrates and explains each of the trace entry types.

```
                Position 5
                    V
(1)-FD98A0  00000250 000060F8 001F54F8 0C000000   00FF5194 00400004 00CEC410 07128D81  *.......8...8.........
    FD98C0  078D7000 00F62F84 00F62F30 00000001   00064F08 00400004 00CEC410 07128D8E  *.....6...6...........
    FD98E0  078D206F 40F63284 00000004 00000004   00064F08 00400004 00CEC410 07128DA9  *.... 6...............
(2)-FDB900  078D1004 00F63284 00000000 00000000   000002B0 00400000 00FE99B8 0712B582  *.....6..........
    FDB920  078D706F 40F63284 00000000 00000000   000002B0 00400004 00CEC410 0712B5A8  *.... 6.........
    FDB940  078D2003 40014C72 00FA903A 00000000   00064DE8 00400004 00CEC410 0712B5C4  *.... ..........
(3)-FDB9C0  078D206F 40F63284 00000004 00000004   00064F08 00400004 00CEC410 0712B6B2  *.... 6...............
(4)-FDB9E0  07003011 B0F647EE 00000000 00CB5000   00CB5000 00400004 00CEC410 0712B6D0  *.....6...............
    FDBA00  078D706F 40F63284 00000000 00000000   000002BF 00400004 00CEC410 0712B700  *.... 6...............
    FDBA20  078D2003 40014C72 00FA903A 00000000   00064DE8 00400004 00CEC410 0712B716  *.... ..............Y.


    FD9740  078D2003 40014C72 00FA903A 00000000   00064DE8 00400004 00CEC410 07128BDF  *.... ..............Y.
(6)-FD9760  07005250 3001A128 001F5258 0C000000   00000000 00400004 00CEC410 07128BEE  *.....................
(5)-FD9780  070C4000 0001C760 00000004 00FF5168   00FF5194 00400004 00CEC410 07128C01  *.. ...G..............
    FD97A0  070C700C 7001A128 00000000 00000088   00CC3CB0 00400004 00CEC410 07128C10  *.....................
(7)-FD97C0  07006000 0003054C 00000004 0000FFFF   00FF5258 00400004 00CEC410 07128C1C  *.....................
    FD97E0  00000250 000060F8 001F5258 0C000000   00FF5194 00400004 00CEC410 07128C58  *.......8.............
    FD9800  070C700C 60E470B4 00FA903A 00000000   00064DE8 00400004 00CEC410 07128C65  *.....U.............Y.
    FD9820  070C200C 60E470B4 00E470E6 00000000   00064DE8 00400004 00CEC410 07128D10  *.....U...U.W.......Y.
(8)-FD9840  078D7000 00E470E6 00E470E6 00000000   00064DE8 00400004 00CEC410 07128D25  *.....U.W.U.W.......Y.
    FD9860  078D5250 00F62F84 001F54F8 0C000000   00000000 00400004 00CEC410 07128D32  *.....6.....8.........
    FD9880  070C4000 0001C760 00000004 00FF5168   00FF5194 00400004 00CEC410 07128D43  *.. ...G..............
```

where:

① – Fifth digit in first word is 0. This is an SIO entry for device 250. The CAW address is '60F8'. The CSW is residual from the previous I/O interrupt. The IOSB address is 'FF5194'.

② – Fifth digit = 1, an external type. This entry has an interrupt code of X'1004' so it was generated by a clock comparator interrupt.

③ – Fifth digit = 2, an SVC interrupt. An SVC '6F' was issued from location F63284 (minus the interruption length code – ILC). Variable fields are registers 15, 0 and 1.

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

⑤ – Fifth digit = 4, an SRB dispatch. The address in the PSW (1C760) is the entry point address. Word 3 contains the ASID to be dispatched. This illustrates the scheduling of POST status after an I/O interrupt.

⑥ – Fifth digit = 5, an I/O interrupt. The device address (250) has been moved into the PSW. Words 3 and 4 are the CSW with the channel end/device end.

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

⑧ – Fifth digit = 7, Task dispatch. Interrupt code is from the last task interrupt. If the interrupt code is 0, it is a return from SVC 0 or the first dispatch of this request block (RB) for the task.

Figure 2-13. 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 Examples

Figures 2-14 through 2-17 illustrate different kinds of MVS and GTF traces, as follows:

Figure 2-14. MVS Trace of a Page Fault Without I/O

Figure 2-15. MVS Trace of a Page Fault With I/O

Figure 2-16. GTF Trace of a Page Fault Without I/O

Figure 2-17. 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 2-14 illustrates a page fault that did not require I/O for completion.

```
      DSP  NEW PSW   071C700A 4009247E  R15/R0 00000000 00000178  R1  00098E88  IDS 0041000C  TCB 008FC728  TME 48CB062C
(a) – PGM  OLD PSW   071C3011 80092480  R15/R0 00000000 00098E8C  R1  00098E88  IDS 0041000C  TCB 008FC728  TME 48CB0635
      SVC  OLD PSW   071C200A 600925F0  R15/R0 00000000 00000178  R1  00098E88  IDS 0041000C  TCB 008FC728  TME 48CB0670
      DSP  NEW PSW   071C700A 600925F0  R15/R0 00000000 00099000  R1  00098E88  IDS 0041000C  TCB 008FC728  TME 48CB06A8

(a) – Fifth digit = 3 and the interrupt code is X'11.' The faulting instruction is at
      X'92480' and is referencing X'98E8C.' 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 and there are no additional trace entries illustrating the process.
```

Figure 2-14. MVS Trace of a Page Fault Without I/O

Figure 2-15 illustrates another possible format of a page fault. It shows an MVS trace (formatted by the SNAP routine) and how it would appear in an SYSUDUMP or SYSABEND dump if the TRT operand was specified in SYS1.PARMLIB or by the CHNGDUMP command.

```
(1)
(2)  DSP  NEW PSW    070C7038 50040162  R15/R0 00040146 C9C7C7F0  R1  F1F9F9C6  IDS 0041000A  TCB 008DD1C8  TME 48CB023B
(2)  PGM  OLD PSW    075C3011 80F8A000  R15/R0 00F8A000 00F8A000  R1  F1F9F9C6  IDS 0041000A  TCB 008DD1C8  TME 48CB0249
(3)  ISD  OLD PSW    070C4000 0005B4A0  ASD/R0 00000001 00FE7400  R1  00000000  IDS 00410001  TCB 00000000  TME 48CB028A
(3)  SIO  CC/DEV/CAW 00005312 0000E5A0  CSW    00076470 0C000001  ISB 00FF0188  IDS 00410001  TCB 00000000  TME 48CB02C2
     DSP  NEW PSW    070E7000 00000000  R15/R0 00000000 00000000  R1  00000000  IDS 00410000  TCB 0001D8A8  TME 48CB03B6
(4)  I/O  OLD PSW    070E5312 00000000  CSW    000768F0 0C000001  RES 00000000  IDS 00410000  TCB 0001D8A8  TME 48CB04D9
(5)  ISO  OLD PSW    070C4000 00025B7A  ASD/R0 0000000A 00FF8510  R1  00000001  IDS 0041000A  TCB 00000000  TME 48CB0874
(5)  DSP  NEW PSW    075C7038 40F8A000  R15/R0 00F8A000 C9C7C7F0  R1  F1F9F9C6  IDS 0041000A  TCB 008DD1C8  TME 48CB08A0
(6)
```

where:

(1) – The page exception.

(2) – The SRB dispatch to initiate ASM's processing in address space 1.

(3) – The SIO by IOS after a branch entry from ASM.

(4) – The I/O interrupt with channel end/device end.

(5) – SRB scheduled in page faulting address space to post the suspend task that I/O is complete.

(6) – Redispatch at page faulting location.

Figure 2-15. MVS Trace of Page Fault With I/O (Formatted by SNAP in SYSUDUMP/SYSABEND)

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	USR=YES
SVC=ALL	GTF=NO
SIO=ALL	DSP=YES
PI=ALL	PCI=YES
IO=ALL	RNIO=NO
EXT=YES	SRM=YES
RR=YES	USERTIME=YES

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

Figure 2-16 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 2-16. GTF Trace of a Page Fault Without I/O

Figure 2-17 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
SRB        ASCB 00FD5858  CPU 0000  JOBN USRT085   SRB PSW 070C0000 0004B6FA  SRB 00FFB480  PARM 00000001  TYPE LOCAL
           TIME    44413.373942
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.
SRB        — The dispatch of RSM's IEAVIOCP page-in completion processor, to
             validate the page table entry and post the faulter as ready to run.
DSP        — The faulter resumed where he left off.
```

Figure 2-17. 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 last 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, 33 – (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.

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 in MVS generally results 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 the middle four bytes of the eight-byte TOD clock at the time the entry was made. The clock can be of considerable importance when trace entries and various system fields (such as the ASCB or LCCA, which also contain TOD clock values) are used to determine how much time has elapsed between significant events. The last digit represents a value that is increased every 16 microseconds. Also, the fourth digit represents the value to be increased every second.

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 causes a clock comparator interrupt (code X'1004') approximately every 1/2 second. These external interrupts are recorded in the trace table. You then see the re-dispatch of the no-work wait task followed by another clock comparator interrupt, and so on. Even though this occurs, the sequence is not repeatedly traced. In addition, in an MP 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. At approximately 12- or 13-seconds intervals, an SRB is dispatched in the master scheduler address space to run a section of SRM in order to gather system statistics. When the SRB has completed, the no-work wait task is again dispatched.

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

4. *MP Activity:*

   The communication between the two processors in the MP 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'191' value =X'FA'.

   Note: When SVC dump turns off the MVS trace, it sets bit 0 on in the CPU identifier (offset X'14') 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.

7. *Important events not traced:*

   MVS design prevents locked, disabled, or SRB code from issuing SVCs. The SVC FLIH abnormally terminates code that issues an SVC. Note that in this case, the erroneous SVC invocation is *not* traced. Also note that locked, disabled, or SRB code that issues SVC D does so as a means of entering RTM1; this is a common technique used by IBM SCP code in order to invoke recovery. RTM indicators show SVC error, but the real problem is why the SVC D was issued.

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'3CC' bytes is often seen soon after an SVC D is issued:*

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

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 the other 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, emergency signal (EMS) operand, which generates an external interrupt on the receiving processor.

When the 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 is 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 consists of cleanup activities performed by other components and by ACR. Post-processing invokes I/O restart (IECVRSTI) to initialize the channel reconfiguration hardware (CRH) function on a Model 168 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. Additionally, the system resources manager (SRM) is notified of the loss of the processor. Finally, ACR performs additional cleanup activities and sets the CSDACR flag to X'00'.

Historically, the parts of the ACR process that have had software problems are the FRRs (written by component developers to protect particular mainline functions) and the ERPs (device-dependent routines). The mainline ACR routine (IEAVTACR) is basic and has been quite free of problems.

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 of a channel control check and interface control check (X'00000000 00060000') and sets 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 via the I/O FLIH.
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 compatability 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 compatability 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 above 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 is a common location for storage overlays. The following should be noted:

- 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.
- Location X'14' should always be 0.
- Locations X'18' through X'3F' (old PSWs) should always contain valid PSWs. The mask (first byte) of each PSW should be X'07', with the exception of X'30' which can contain X'0', X'04', or X'07'.
- 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'0'. 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

Three 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 first level interrupt handlers or in code pointed to by the 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'C34', X'C50', X'C54', X'C5C', X'C7C', 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, describes the error causing the interrupt. 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 interupt. 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 are listed below, 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).

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.

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.

Miscellaneous Debugging Hints (continued)

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 (processor 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.

Miscellaneous Debugging Hints (continued)

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	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) = 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:

- The RTM2ABNM and RTM2ABEP fields do not contain information about the abending program if an SVC has abended.
- 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 *OS/VS Message Library: VS2 System Codes.* A summary of system codes is in the *Debugging Handbook Volume 1.*

   Note: Completion codes are not printed at the top of abend dumps that are formatted with the AMDPRDMP service aid. System completion codes can be found in the third to fifth digits (00xxx000) of the abend completion code in the RTM2 work area. User completion codes are located in the sixth to eighth digits (00000xxx) of the abend code in the RTM2 work area, and in this case are in 3-digit hexadecimal form.

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 request blocks (RBs) that are formatted in the dump. 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.*

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 general' 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	–	–

BRANCH-ENTRY TABLE (Continued)

Record ID Dec.	Hex	*Title*	*Mapping Macro*	*Fields used to Dump PSW or Register Areas*
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	–	–
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 Stacks (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.

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 issues message IEE824I.

0BA – 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.

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 alteration 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

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

There are no defaults for an SVC dump other than SQA, ALLPSA, and SUMDUMP, which are assumed by the dump program if the options NOSQA, NOALLPSA, and NOSUM are not specified.

The CHNGDUMP operator 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: OS/VS2 MVS System Commands* for the format of the CHNGDUMP 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: OS/VS2 MVS 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 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=(TRK,(400,20)),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
```

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.

The following statements should be included in every run of PRDMP:

SUMMARY – defines and prints the dump ranges of the dump, active processor, active tasks, etc.

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.

QCBTRACE – formats the END/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. However, it should be included in the initial run of PRDMP because 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 should also be included in every initial run of PRDMP. It 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 OS trace is not valid if GTF is running.

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.

- 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: OS/VS2 MVS 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
   CVTMAP
  END
/*
```

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 TSOTEST * IKJACCNT) UNIT(SYSDA) ACCT OPER JCL MOUNT
ADD (USER02 TSOTEST * IKJACCNT) UNIT(SYSDA) ACCT OPER JCL MOUNT
ADD (USER03 TSOTEST * IKJACCNT) UNIT(SYSDA) ACCT OPER JCL MOUNT
ADD (USER04 TSOTEST * IKJACCNT) UNIT(SYSDA) ACCT OPER JCL MOUNT
ADD (USER05 TSOTEST * IKJACCNT) UNIT(SYSDA) ACCT OPER JCL MOUNT
ADD (USER06 TSOTEST * IKJACCNT) UNIT(SYSDA) ACCT OPER JCL MOUNT
ADD (USER07 TSOTEST * IKJACCNT) UNIT(SYSDA) ACCT OPER JCL MOUNT
ADD (USER08 TSOTEST * IKJACCNT) UNIT(SYSDA) ACCT OPER JCL MOUNT
ADD (USER09 TSOTEST * IKJACCNT) UNIT(SYSDA) ACCT OPER JCL MOUNT
ADD (USER0A TSOTEST * IKJACCNT) UNIT(SYSDA) ACCT OPER JCL MOUNT
ADD (USER0B TSOTEST * IKJACCNT) UNIT(SYSDA) ACCT OPER JCL MOUNT
ADD (USER0C TSOTEST * IKJACCNT) UNIT(SYSDA) ACCT OPER JCL MOUNT
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))
  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 SYSTEMS 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
```

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 *OS/VS2 System Programming Library: 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 coldstart (CLPA,CVIO) option. Loss of this data set prevents warmstarting the system or restarting jobs using VIO data sets.

The following job has been run successfully to 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(12 8)-
  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 two areas in the PSA reserved for future expansion. They can be used for quick implementation of a trap without having to consider base registers. They are X'410' – X'BFF' and X'E54' – X'FFF'. Both of these areas are frequently used in examples throughout this chapter.

CAUTION: Use extreme care when you use this method. Patches should be made only to disabled code unless the patch is completly 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 from RTM 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, usually entered by a system programmer, either at the console or via the input stream, can also reside in the COMMNDxx parmlib member. The SLIP command's purpose is to establish SLIP definitions of the error circumstances under which interception of an error is to occur, and of the action to be taken following the interception.

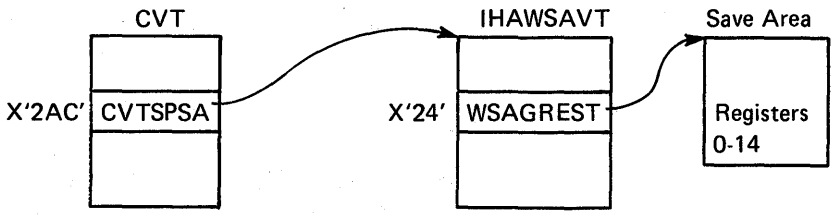
As long as enough system queue area storage is available, SLIP definitions may be established at any time. The recovery termination manager (RTM) compares the SLIP definitions with the dynamic system conditions at the time of the error. If RTM detects a match, the requested action is taken.

The ACTION keyword has the following options:

- ACTION=SVCD indicates that an SVC dump will be scheduled for the current ASID. This is the default option if ACTION is not specified. SDUMP parameters in this case are: SUM, SQA, RGN, TRT, LPA, CSA, and NUC.

One of the advantages of this 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 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. The biggest advantage is in situations where no dump was occurring.

- ACTION=WAIT indicates that the system will be placed in a 01B wait state. At this time, the operator can find the save area where the stop/restart routine (IEESTPRS) saves the caller's (IEAVTSLP) registers. Register 2 contains the address of the RTM work area for the error. This is either IHAFRRS (RTM1) or IHARTM2A (RTM2). Register 4 contains the address of the SLIP control element (SCE), which contains the id for this trap.



- ACTION=NODUMP indicates that SLIP is to set a flag in the RTM work area which is checked by the dump programs ABEND/SNAP 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 or its subroutines (FRR and ESTAE) are suppressed. Should the error involve recursive entry into RTM, the bit setting is propagated to the next RTM work area.

  This action is useful for preventing dumps that may not be needed (X22, 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 indicates that the system will not do any further SLIP processing, and that normal system recovery will continue. This option is normally chosen for known errors. For example, if trapping 0C4 completion codes and SLIP SET,COMP=0C4,ACTION=IGNORE,LPAMOD=MODX,END is entered after SLIP SET,COMP=0C4,A=SVCD,END had been issued, it results in dumps for all 0C4 errors except those in module MODx. The ACTION=IGNORE command must be issued after the original command because trap conditions are checked LIFO.

It is also possible to display information about SLIP definitions by using the DISPLAY command at the operator's console. For details concerning operand usage and entering the SLIP and DISPLAY commands, see *Operator's Library: OS/VS2 MVS System Commands.* The following is provided to demonstrate a typical application of the SLIP command:

Obtaining a Dump with Queue Control Blocks and Elements

An error in the DEQ SVC routine is suspected because whenever program DVTRTN executes, it abnormally terminates even though its parameter list is correct. The resulting abend dump does not include queue control blocks and queue elements. To get a dump that does include this information, issue the following SLIP command:

```
SLIP SET,ID=QELS,COMP=X30,ERRTYPE=ABEND,JSPGM=DVTRTN,END
```

ID identifies this SLIP definition as "QELS"; COMP specifies the applicable system completion code; ERRTYPE specifies that an abend condition must exist for this error interception; JSPGM identifies "DVTRTN" as the job step program that must be executing for this error interception; END denotes the end of this SLIP command.

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.

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. The following chart lists several SLIP keywords and indicates the data area fields that SLIP compares them with.

It should be understood that 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 routines, 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.

SLIP Keywords and Corresponding Data Areas

Note: There may be several RTM2 work areas pointed to by the TCB if several abends occurred. The oldest one (last on the queue) is probably the best one to use.

ERRTYP

(RTM1)

In RT1TENPT of the RTM1 work area is the number indicating the reason for entry into RTM1:

1=PROG	5=MACH
2=REST	10=PGIO
3=SVCERR	15=MEMTERM
4=DAT	

(RTM2)

The reason for entry into RTM2 is indicated by flags in the RTM2 work area as follows:

RTM2MCHK=MACH	RTM2SVCE=SVCERR
RTM2PCHK=PROG	RTM2TEXC=DAT
RTM2RKEY=RESTART	RTM2PGIO=PGIO
RTM2SVCD=ABEND	RTM2EOM=MEMTERM
RTM2ABTM=ABEND	

MODE

System mode at error time 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)

(RTM1)

The MODEBYTE value is contained in RT1WMODE. The PSW from SDWAEC1 is used for PP, Super, SKey, and PKey states. The SDWASTAF bit is used for RECV.

(RTM2)

In the ESAMODE field (SVRB + X'8B') of the SVRB pointed to by RTM2VRBC, are bits mapped by MODEBYTE as indicated above. For the PSW values, SLIP uses the RBOPSW field of the RB preceding the SVRB.

The RTM2RECR bit must be on for RECV, and in the previous RTM2 work area the RTM2XIP bit plus the SCBINUSE bit of the SCB pointed to by RTM2NSCBN must be on.

COMP

(RTM1)

In the SDWA, field SDWACMPC contains the original value.

(RTM2)

The RTM2CC field contains the original value for each work area.

JOBNAME

(RTM1 and RTM2)

In the ASCB, fields ASCBJBNI or ASCBJBNS point to the job name for either initiated or started jobs.

JSPGM

This keyword does not apply to errors which enter RTM1, so if it is specified, the trap is limited to RTM2 type errors only.

PVTMOD (RTM2 only)
LPAMOD
ADDRESS

The address used for these keywords is obtained from the same PSW used when checking values for the MODE keyword. Additionally, PVTMOD applies to RTM2 type errors only and restricts the trap accordingly. The module name for PVTMOD is compared with those in the CDE list for the jobstep TCB of the current address space.

ASID

(RTM1)

Both SDWAFMID and SDWAASID are checked.

(RTM2)

Both RTM2FMID and ASCBASID are checked.

Enabling The PER Hardware To Monitor Storage Locations

A convenient place to hook the system is in the MVS trace table's common prologue code in IEAVTRCE. All interrupts and dispatcher entries enter this code. Therefore a modification here will enter this trap after every interrupt and before the dispatcher dispatches or redispatches any TCB or SRB. The trap in the examples below was inserted in module IEAVTRCE three instructions after the label STR in place of the code that normally stores the timer value in the trace table.

This trap does not stop the system but traces in the MVS trace the PSWs that alter a specified storage location. To stop the system, a branch from the program check FLIH can be made to a patch area, and a test can be made for the interrupt code of X'80' with a branch equal to a trap to stop the system. In the system dump, the instruction that performs the modification is pointed to by X'98' in the PSA.

Care should be used with this diagnostic aid since degradation occurs in proportion to the number of interrupts taken. Only use it to monitor a section of storage which is never modified or only infrequently modified. Once the trap is in, there is no need to re-IPL to remove it. Manually storing a word of zeros in control register 9 prevents further interrupts.

Following is an example of the PER hardware trap to be applied by superzap.

```
NAME IEANUC01 IEAVTRCE
VER  03A0 B2058000,4780B02C,D70380028002,D203C01C8002,947FC014
REP  03A0 47F00608,07000700,07000700,07000700,07000700,07000700

NAME IEANUC01 IEAVTRTS
VER  0796 82001078
REP  0796 47F00600

NAME IEANUC01 IEAVFX00
VER  0600 00000000,00000000,00000000,00000000,00000000,00000000,00000000
VER  061C 00000000,00000000,00000000,00000000,00000000,00000000,00000000
REP  0600 96401078,82001078      TURN ON PER BEFORE ENTERING FRR
REP  0608 96400300               ALWAYS TURN ON PER FOR DISPATCHER
REP  060C 4700B032,92F0060D      BUT SET THE FIRST TIME SWITCH FOR THE REST
REP  0614 96400058,96400060,96400068,96400070,96400078  SET THE NEW PSW.
REP  0628 B79B0630               LOAD FUNCTION CODE, LOW AND HIGH RANGE
REP  062C 47F0B032               RETURN TO MAINLINE
REP  0630 XX000000               FUNCTION CODE IN HIGH ORDER
REP  0634 XXXXXXXX               LOW RANGE  } *
REP  0638 XXXXXXXX               HIGH RANGE }
```

**Note:* To check a word in storage starting at 9F41C for example,
Low range address = 9F41C
High range address = 9F41F.
To check a byte, use the same address in low and high.

Because the switch is in the PSA, the control registers and NEW PSWs are initialized on both processors in an MP environment. However, they are set only once and not each time through the routine.

The example in Figure 2-18 shows trace entries using the storage alteration mask (function code X'20'). The interrupt address is the address of the instruction that modified the monitored storage.

```
PGM  OLD PSW   470C3080 A0009A0A   R15/R0 00009970 00DF467A   R1  00FF0B08   IDS 00400002   TCB 00000000   TME 9BD80401
PGM  OLD PSW   470C3080 E0034126   R15/R0 00014C20 00DF467A   R1  00FF837C   IDS 00400002   TCB 00000000   TME 9BD80439
PGM  OLD PSW   470C3080 A000BEF8   R15/R0 0000BEB8 00DF467A   R1  00FF837C   IDS 00400002   TCB 00000000   TME 9BD8053F
```

Figure 2-18. Trace Example of PER Hardware Monitoring

On occasion it might be necessary to monitor when only one address space is active. One way of doing this is to change the previous superzap example at address 060E from B032 to 0640 and include the following superzap. This superzap turns PER on only if the specified address space is active.

```
NAME IEANUC01 IEAVFX00
VER  0640 00000000,00000000,00000000,00000000,00000000,00000000,00000000
REP  0640 58D00224               GET CURRENT ASCB
REP  0644 48D0D024               GET CURRENT ASID
REP  0648 49D00664               IS THIS MY ASID?
REP  064C 47800658               YES – GO TURN PER ON
REP  0650 B7990660               TURN PER OFF
REP  0654 47F0B032               RETURN TO MAINLINE
REP  0658 B7990630               TURN PER ON
REP  065C 47F0B032               RETURN TO MAINLINE
REP  0660 00000000               THIS WORD IN CR9 TURNS OFF PER
REP  0664 xxxx                   ASID TO BE MONITORED
```

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.

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'756'. 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 36F'00'
REP 0700 ACFC074E          DISABLE
REP 0704 900F0758          SAVE REGISTERS
REP 0708 58F00010          GET CVT POINTER
REP 070C 58E0F294          GET CSD POINTER
REP 0710 91C0E008          TEST IF MP
REP 0714 47E00744          NO JUST LOAD WAIT PSW
REP 0718 41200000          SET REG 2 TO CPU 0
REP 071C 41300001          SET REG 3 TO CPU 1
REP 0720 48400204          GET CPU ADDRESS
REP 0724 1244              TEST FOR CPU 0
REP 0726 4770073C          NO, STOP CPU 0 FIRST
REP 072A AE030009          YES, STOP CPU 1 FIRST
REP 072E 4760072A          SPIN TIL CC=0
REP 0732 D20700000750      MOVE THE WAIT PSW TO ZERO
REP 0738 82000000          LOAD WAIT STATE ON CPU 0
REP 073C AE020009          SIGP STOP CPU 0
REP 0740 4760073C          SPIN TIL CC=0
REP 0744 D20700000750      MOVE THE WAIT PSW TO ZERO
REP 074A 82000000,0000     LOAD WAIT STATE ON CPU 1
REP 0750 000E0000,0000DEAD WAIT PSW
REP 0758 00000000          SAVE AREA
```

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.

Using The MVS Trace To Monitor Storage

The MVS trace code in module IEAVTRCE is an excellent place to hook the system to monitor system operation and branch to a trap routine. Three instructions past label STR in IEAVTRCE is the code which stores the timer values in the trace table. All trace entries pass through this code. Overlaying this code allows you to monitor any place in the system as it runs disabled, key zero and supervisor state. It must be understood that this code is physically disabled and therefore the trap must not page fault. Also no reference can be made to private area addresses since the trap can receive control in any address space. For larger patches a branch from this code to a patch area in the PSA is possible. At entry to this code, register 12 (C) points to the trace entry. This code normally stores the timer value located at X'1C' into the trace table. Storing a word at register 12 (C) + X'1C' would allow dynamic monitoring of that word in storage if addressability is obtained. The other seven words of the trace table are built within the trace entry code for each trace type. Monitoring for more than one word entails changing all entries.

To eliminate certain trace entry types, it is only necessary to put a branch instruction 07FB at the entry point for that entry.

Caution: Location X'10' cannot be monitored with this trap because the PCFLIH refreshes location X'10' before it branches to the trace routine. 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

Use the following zap to force trace on during NIP processing.

```
NAME IEEVWAIT IEEVWAIT
VER 0194 4710
REP 0194 47F0
```

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 2EC0 0190
REP 2EC0 XXXX          WHERE X IS THE NEW VALUE DESIRED.
```

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.

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

SEC-
TION 3

3.1.2 OS/VS2 System Programming Library: MVS Diagnostic Techniques

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 he observes that a given segment of the system (such as VTAM) is not functioning (other areas appear okay – jobs are executing, SYSIN/SYSOUT is appearing, etc.), he may decide to take a stand-alone dump. In this case, the current state of the system is probably not important. He would not be interested in current PSW, registers, etc.; he 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 the analyst 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 (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 address 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.

The SVC dump is taken asynchronously and the global data areas (PSA, LCCA, PCCA, etc.) usually contain no relevant data except in cases where overlays, machine checks, channel checks, etc., have occurred.

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 program changes needed to print the summary dump, and multiple address-space output from SVC dump, see *OS/VS2 System Programming Library: 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 enhanced SVC Dump
	...1	user-specified ASIDLST=
	.... 1...	user-specified SUMLIST=
	others	reserved
2		SDATA options
	1...	ALLPSA
	.1..	PSA
	..1.	NUC
	...1	SQA
	.... 1...	LSQA
	.... .1..	RGN
	.... ..1.	LPA
	1	TRT (MVS trace table)

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
C		address of header record
10		address of ECB
14		caller's ASID
16		target ASID of scheduled dump
18		address of ASID list
1C		address of summary dump storage list
20		address of SYSMDUMP 4K SQA area
24		address of SYSMDUMP CSA work area
28		length of header record (less than 100)
29		header record (will appear as title)

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 *OS/VS2 System Programming Library: 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 *OS/VS Message Library: VS2 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 listing (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 intercepted 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

4.1.2 OS/VS2 System Programming Library: MVS Diagnostic Techniques

Wait states may be either enabled or disabled. The characteristics of each type are described below.

haracteristics 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.

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 *OS/VS Message Library: VS2 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 or hardware are overlaid and subsequently fetched for use in an LPSW. This often occurs when a PSA overlay condition exists, that is, the low storage PSWs fetched by the hardware 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. Bad LPSWs are difficult to track down. The most common MVS uses of the LPSW in instructions are:

- hardware loading from low storage for an interruption-processing sequence
- dispatcher loading from X'300' into the PSA
- RTM (IEAVTRTS) passing control to FRRs.
- the system termination routine
- SVC FLIH and I/O FLIH LPSWs.

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 "Standalone 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 code in *OS/VS Message Library: VS2 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.

If, for any reason, the current PSW is not formatted in the dump, the last PSW shown in the trace table, location X'300' (used by the dispatcher), or low storage should be examined as possible sources of the last PSW.

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 back-tracking.

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).

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 bottle-neck. 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 *vast* 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.

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) or there should be a considerable recurrence of the no-work wait in the OS 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:

   - There should be no bit set in the PSASUPER field (PSA+X'228'). If there is, some supervisor routine should be 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 (PSAHLHI), 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 PSAHLHI 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 conrol (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 the local lock. If in task mode at the time of suspension, the resume PSW instruction address (saved at IHSA + X'10') should be checked. When the instruction address = RBRTRAN (–C offset), it indicates the task is suspended while it waits for a page fault resolution. 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 = 00000000 indicates no locks are held. The RB structure can reveal the same situation as described above for RBOPSW instruction address = RBRTRAN or RBXWAIT=0 *and,* in addition, an RB wait count = 1. If you find several tasks in this state, check the dump for the page represented by RBRTRAN. 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 believe paging is a potential problem, check the PVTAFC (available frame count). A "low" value may indicate a frame shortage. While "low" is difficult to define, the value should certainly be above the PVTAFCOK value (PVT+X'6'). Beyond this, "low" is influenced by sizes of working sets of the address spaces in your system. The working set size for each address space is contained in the associated SRM-user control block (OUCB). This count (plus an SRM constant of 10) is the number of frames required to swap-in the corresponding address space. If enough frames are not available, the address space will remain swapped-out.

   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 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 facility of AMDPRDMP can be most helpful. An illustration follows:

   Investigation of QCBTRACE 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 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 GSMQ, LSMQ, GSPL, and each LSPL 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. In short, 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 a count of ready TCBs in the memory
- 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 locks (local/CMS) 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 CMS/local suspend queues may indicate a problem. The technique you adopt to resolve the conflicts is exactly the same as the ENQ interlock or log-jam 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 and TCAM chapters in Section 5 contain techniques for VTAM and TCAM 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 to 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 are defined as disabled or enabled, depending upon their external appearances. A *disabled* loop can be recognized externally by a solid system light 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 two common loop situations:

1. Two 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 pushed the STOP button on one processor and not the other to investigate a suspected problem.
   - The receiving processor 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, the other processor's 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 system, this usually indicates that the other processor is holding the lock and not releasing it. But it may indicate an overlaid lockword; if not, the problem is definitely on the other processor. In either case, register 11 contains the pointer to the requested lockword and register 14 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.

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.

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'40' 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, pre-empted 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 $\neq 0$. This alone does not prove there is a loop, but the information may help your analysis of the problem later.

Loops (continued)

- 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 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. SRM uses a timer interrupt approximately every .4 seconds 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 (PSA+X'300'). 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. 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 code 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 both 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.

4.2.6 OS/VS2 System Programming Library: MVS Diagnostic Techniques

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
- Trace Output Under Normal Conditions

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 a TCAM application to a TCAM terminal. The path might go from the application program buffer to a TCAM queue data set, to a TCAM buffer while 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 check-pointed 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.

TP Problems (continued)

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 *TCAM Debugging Guide* should be referenced for problem isolation in 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 EP and NCP mode traces and their relationships to each other. For more information on these traces refer to:

- *IBM 3704 and 3705 Communication Controllers Emulation Program Generation and Utilities Guide and Reference Manual*
- *IBM 3704 and 3705 Communication Controllers Network Control Program/VS Generation and Utilities Guide and Reference Manual*
- *OS/VS2 MVS VTAM Debugging Guide*
- *IBM 3704 and 3705 Program Reference Handbooks*
- *OS/VS2 TCAM System Programmer's Guide, Level 10*
- *OS/VS TCAM Debugging Guide, Level 10*

Figure 4-1 shows a summary of EP and NCP mode traces.

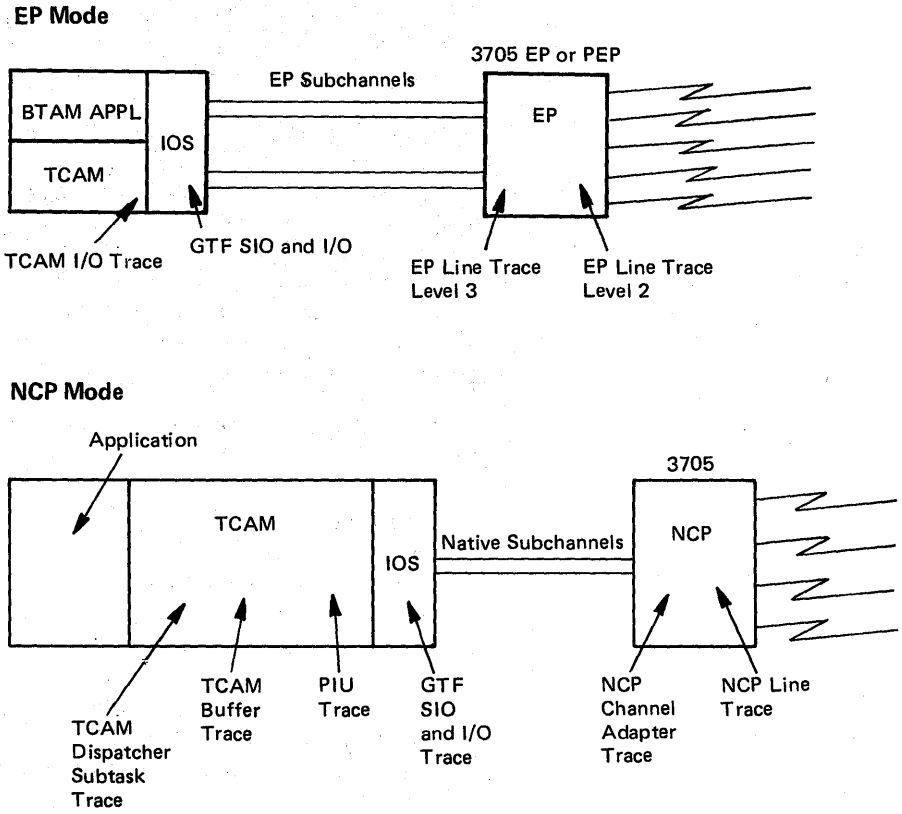


Figure 4-1. Summary of EP and NCP Mode Traces

EP Mode Traces

The following describes the six EP mode traces:

1. ***3705 Emulator Program line trace*** – Any or all emulator subchannels can be traced in the 3705. Each character over a line (level 2 interrupt) and/or each interaction with the channel (level 3 interrupt) for data or status transfer can be traced. The trace is activated via the 3705 panel or via the EP (emulator program) service aid, Dynadump. Trace data is retrieved from a 3705 dump or via Dynadump. All of the 3705 storage above the relatively small emulator program itself is used for a trace table (this can be a significant amount in a large 3705).

   If a problem can be isolated to one or two lines and can be detected quickly, an in-storage trace (in the 3705) is usually sufficient. The trace can be stopped before critical information is lost because of wrapping. If all lines must be traced, if there are a good number of high-speed lines with constant activity such as polling, or if the problem is not externally detectable, then Dynadump should be used to dump trace data as it is created.

2. ***PEP Emulator line trace*** – This traces EP lines in a PEP (partioned emulator program). The size of the trace table is set at PEP generation time and is fixed. The size of the trace table compared to the amount of storage used for tracing in an EP makes wrapping a much more serious problem. Dynadump is extremely useful when tracing EP lines in a PEP.

3. ***GTF I/O and SIO trace*** – All emulator subchannels can be traced. SIOs, I/O interrupts, CAWs, CSWs, and SIO condition codes are traced. No CCWs or data are traced. Data can be traced to an external file and selectively reduced by time and subchannel address. SIOs and I/O interrupts are directly correlated with channel activity seen in a 3705 EP line trace. Refer to *OS/VS2 System Programming Library: Service Aids* for details on activating and printing this trace.

4. ***TCAM EP mode line I/O interrupt trace table*** – This trace I/O interrupts on EP or 2701/2/3 lines. The TCAM INTRO macro or the operator specifies the response at TCAM start up.

5. ***TCAM dispatcher subtask trace*** – This records the flow of all TCAM-dispatched elements. It is specified, as is the TCAM I/O trace, by the INTRO macro or by operator responses. It is useful in determining the flow of activity within TCAM. It contains a history of TCAM buffers, LCBs, QCBs, etc., that are processed by the various TCAM subtasks.

6. ***TCAM buffer trace*** – This traces buffers for specified lines as they are processed by a message handler (MH). The lines are the same lines as those specified by the I/O interrupt trace or TPIO trace.

NCP Mode Traces

The following describes the NCP mode traces:

1. *NCP line trace* – This traces all characters on a given TP line (level 2 interrupt) in the 3705. Only one line can be traced at a time. There is one four-byte entry for each character interrupt in the 3705, including a one-byte time field. The time field wraps after 25.5 seconds, but it is useful for seeing delays and when time-out periods expire. This trace is controlled through access method operator commands. Data is shipped to VTAM from the NCP and traced via GTF, so GTF must be active for the USR trace when line trace data is required.

2. *NCP channel adapter trace* – This traces the NCP's interaction with the channel for status and data transfer. There is one 32-byte entry for each Level 3 NCP channel adapter interrupt. The data stays in a static trace table within the 3705 and is retrieved via a 3705 dump. The trace option is included in an NCP by altering the TRACE= operand in the SYSCG006 macro (in SYS1.MAC3705). Refer to the section "Channel Adapter Trace" in *IBM 3704 and 3705 Communications Controller Network Control Program/VS Logic*, and to the *3704/3705 Program Reference Handbook.*

   If the NCP uses a Type 1 channel adapter, the trace includes all data transferred over the channel. If the NCP uses a Type 2 or Type 3 channel adapter (CA), the trace does not include data. The trace is most useful for problems involving NCP ABENDs where the last activity on the channel is of interest. Because the trace wraps very fast and cannot be written out dynamically, it is not useful for finding what happened over a period of time.

3. *GTF SIO and I/O trace* – This traces SIO and I/O interrupts for a 3705. Its main value is in showing what SIOs and I/O interrupts take place in relation to the RNIO trace (which is discussed next). The RNIO trace shows data (PIUs) in and out. The SIO and I/O traces can then show attention interrupts and how many PIUs were transferred at a time and if there are problems in timing or in "coat-tailing." Coat-tailing is the ability to bring PIUs into the system from the 3705 on the same I/O operation used to write other data out to the 3705.

4. *GTF RNIO trace, or VTAM I/O trace* – This shows the header and first few bytes of data of each PIU coming into or going out of VTAM. It is sent to GTF under the RNIO trace option. Each entry is time-stamped. The tracing is done in channel-end appendages as soon as the I/O operation that transfers the data is complete.

TP Problems (continued)

5. ***VTAM buffer trace*** – This is a VTAM trace sent to GTF under the USR trace option. The trace is performed at two points in VTAM.

   a) *TPIOS.* Traces are labeled TPIOS IN/OUT REMOTE. Application jobname, destination and origin node names, feedback data block (FDB), feedback status block (FSB) for input operations, PIU header and text are traced.

      For TPIOS OUT REMOTE entries, the transmission header is not exactly as it will appear when I/O is finally performed. Sequence number and length fields, plus some other bits in the TH (transmission header) may be filled in after the TPIOS trace. Use the RNIO trace to see exactly how the PIU header was sent to the 3705.

   b) *Control Layer.* Traces are labeled C/L IN and C/L OUT. Application jobname, destination and origin node names, and text are traced approximately at the time of the transfer of a message from the application's buffer to VTAM's buffer, and vice-versa.

      Note: VTAM traces such as I/O trace (RNIO) and buffer trace (USR) must be started and stopped by a VTAM operator command for each terminal or node in the network that is to be traced. There is no higher level operation available.

6. ***TCAM dispatcher subtask trace*** – This trace is described in item 5 under "EP Mode Traces" earlier in this chapter.

7. ***TCAM buffer trace*** – This trace is similar to the TCAM buffer trace for EP lines described at the end of the section "EP Mode Traces" earlier in this chapter.

8. ***TCAM PIU trace*** – This traces path information units (PIUs) for a line, a line group, or an NCP.

Trace Output Under Normal Conditions

The following sections illustrate how some normal situations are seen in traces. Understanding normal processing cannot be over-emphasized, for it is often a comparison between traces such as these and a trace of an error that reveals the key to a problem.

Example 1: VTAM I/O Trace

The first example (Figure 4-2) shows only a VTAM I/O trace (RNIO) for:

1. the activation of a PEP (it is incidental that it's a PEP),
2. the activation and connection of a 3600 logical unit,
3. data exchange between an application and the LU, then:
4. disconnection and deactivation of the LU and the PEP.

Trace Entries:

- 1 - 9 – show the PEP activation and initialization
- 10 - 15 – are line-activates for start-stop lines
- 16 - 21 – are line-activates for some BSC lines.
- 24 - 32 – show the activation of the link, controller, and LU. Operator VARY commands are used to activate a 3600 cluster controller and logical unit. Unlike old-device lines, the SDLC link is not activated until a device on the link needs to be activated.

  An application program is then started that opens (via OPNDST) the LU.
- 33 - 37 – show both the connection and first message to the LU.
- 38 - 42 – show another link activation and controller contact; the controller and its LUs are not being traced, however, so none of the subsequent activation is shown. (Every node to be traced must have a VTAM command issued.) The PEP itself is being RNIO traced, which is the reason its activation is shown.
- 43 - 52 – show data exchange between the LU and the application.
- 53 - 56 – show the disconnection from issuing a CLSDST.
- 57 - end– show the PEP deactivation of the SNA devices, the S/S and BSC lines, and the PEP itself.

Note entries 8 - 9 and 22 - 23 in Figure 4-2: this trace was made with a level 3.0 PEP, which does not support VTAM's attempt to alter the channel attention delay; therefore, error responses are received. VTAM, through the NCP or PEP, immediately activates all old-device lines (S/S and BSC) in NCP mode. The SDLC devices are system generated with ISTATUS = INACTIVE and therefore, no SDLC links are activated. (The VTAM messages IST . . . . .I PEP736A ACTIVE would have appeared on the console after trace entry 23.)

```
                                                    EXTERNAL TRACE - DD TAPE

   *** DATE   DAY 168    YEAR 1975            TIME 16.30.32.113950

 1 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM          IN 1D000000 20000000 00092B00 00500900 400698
             TIME    41690.744573
 2 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM         OUT 1F002000 10000001 00066B80 00110103
             TIME    41690.745676
 3 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM          IN 1F001000 20000001 000DFB80 001101D7 C5D7F7F3
             TIME    41691.003599
 4 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM         OUT 1F002000 10000002 00046B80 00A0
             TIME    41691.226997
 5 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM          IN 1F001000 20000002 0004FB80 00A0
             TIME    41691.528884
 6 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM         OUT 1E002000 10000001 001D0B80 00010211 200001F0
             TIME    41691.549546
 7 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM          IN 1E001000 20000001 00069B80 00010211
             TIME    41691.799592
 8 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM         OUT 1E002000 10000002 000B0B80 00010211 20000500
             TIME    41691.818728
 9 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM          IN 1E001000 20000002 000A9F90 00100200 00010211
             TIME    41692.099679
10 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM         OUT 1E002000 10000003 00080B80 0001020A 2001
             TIME    41692.117212
11 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM          IN 1E001000 20000003 00069B80 0001020A
             TIME    41692.399922
12 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM         OUT 1E002000 10000004 00080B80 0001020A 2003
             TIME    41692.417538
13 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM          IN 1E001000 20000004 00069B80 0001020A
             TIME    41692.700031
14 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM         OUT 1E002000 10000005 00080B80 0001020A 2005
             TIME    41692.723716
15 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM          IN 1E001000 20000005 00069B80 0001020A
             TIME    41693.000187

16 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM         OUT 1E002000 10000026 00080B80 0001020A 206B
             TIME    41704.523269
17 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM          IN 1E001000 20000026 00069B80 0001020A
             TIME    41704.805988
18 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM         OUT 1E002000 10000027 00080B80 0001020A 2079
             TIME    41704.863258
19 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM          IN 1E001000 20000027 00069B80 0001020A
             TIME    41705.106058
20 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM         OUT 1E002000 10000028 00080B80 0001020A 2087
             TIME    41705.185170
21 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM          IN 1E001000 20000028 00069B80 0001020A
             TIME    41705.406283
22 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM         OUT 1E002000 10000029 000B0B80 00010211 20000500
             TIME    41705.758939
23 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM          IN 1E001000 20000029 000A9F90 00100200 00010211
             TIME    41706.006557
24 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM         OUT 1E002000 1000002A 00080B80 0001020A 211B
             TIME    41746.543962
```

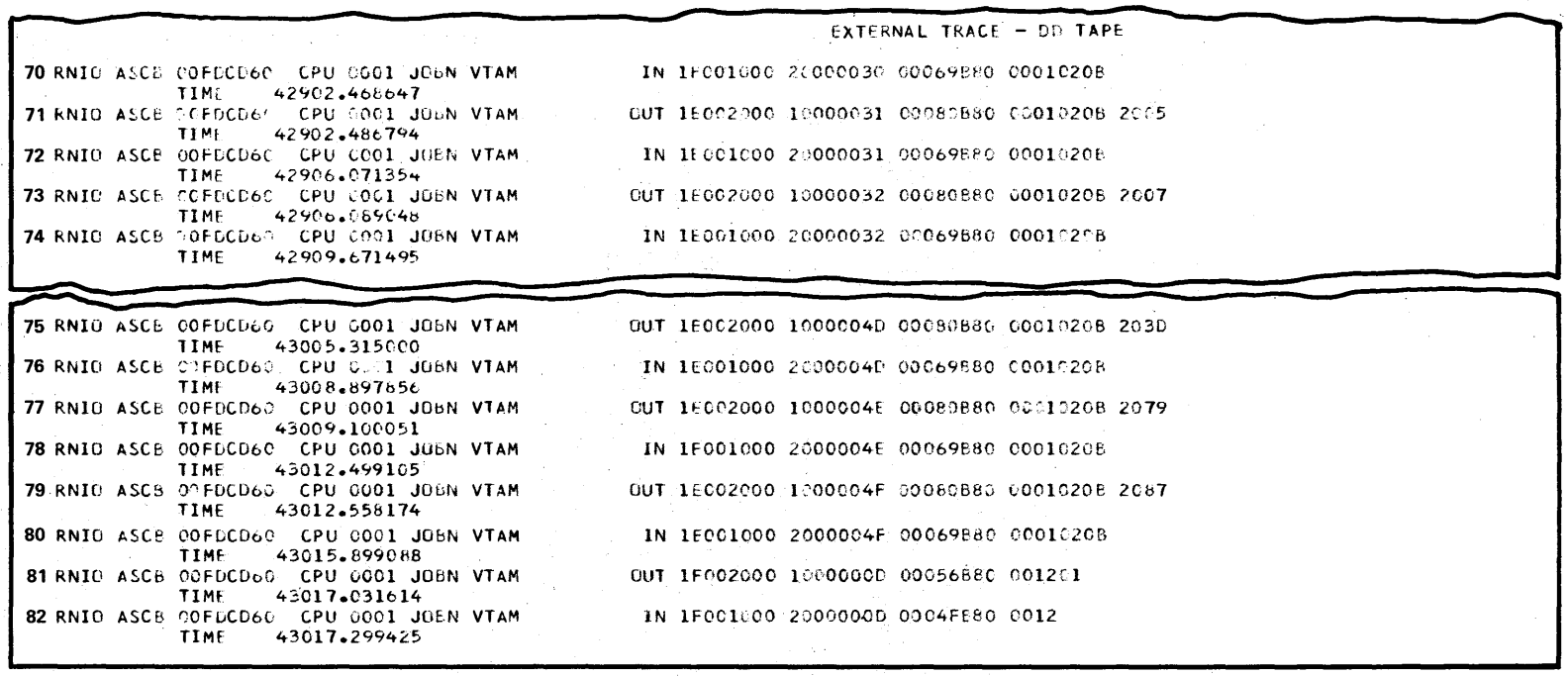
Figure 4-2. VTAM I/O Trace Example (Part 1 of 4)

```
                                                                EXTERNAL TRACE - DD TAPE

25 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM          IN 1E001000 2000002A 00069B80 0001020A
           TIME    41746.823375
26 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM         OUT 1E002000 1000002B 00080B80 00010201 211C
           TIME    41746.842293
27 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM          IN 1E001000 2000002B 00069B80 00010201
           TIME    41747.123477
28 RNIO ASCB 00FDCD6C  CPU 0001 JOBN VTAM          IN 1C001000 20000001 00090B00 00010280 211C01
           TIME    41749.324865
29 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM         OUT 1F00211C 10000001 00066B80 00110101
           TIME    41749.345127
30 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM          IN 1F001000 211C0001 00CDEB80 001101F3 F6F0F0F0
           TIME    41750.025422
31 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM         OUT 1F00211D 10000001 00066B80 000D0101
           TIME    41758.282656
32 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM          IN 1F001000 211D0001 0005EB80 000D01
           TIME    41758.828781
33 RNIO ASCB 00FE4390  CPU 0001 JOBN TRAC3600      OUT 1F00211D 10010001 00276B80 00310103 02FF9100
           TIME    41787.394270
34 RNIO ASCB 00FE4390  CPU 0001 JOBN TRAC3600       IN 1F001001 211D0001 0004EB80 0031
           TIME    41788.042000


   *** DATE   DAY 168    YEAR 1975            TIME 16.36.27.900691

35 RNIO ASCB 00FE4390  CPU 0001 JOBN TRAC3600      OUT 1F00211D 10010002 00046B80 00A0
           TIME    41788.097764
36 RNIO ASCB 00FE4390  CPU 0001 JOBN TRAC3600       IN 1F001001 211D0002 0004EB80 00A0
           TIME    41788.642254
37 RNIO ASCB 00FE4390  CPU 0001 JOBN TRAC3600      OUT 1E00211D 10010001 00230390 00D9C5C1 C4E84B40
           TIME    41789.441206
38 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM          OUT 1E002000 1000002C 00080B80 0001020A 212E
           TIME    41872.831413
39 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM          IN 1E001000 2000002C 00069B80 0001020A
           TIME    41873.071784
40 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM          OUT 1E002000 1000002D 00080B80 00010201 212F
           TIME    41873.266604
41 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM          IN 1E001000 2000002D 00069B80 0001C201
           TIME    41873.571967
42 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM          IN 1C001000 20000002 00090B00 00010280 212F01
           TIME    41875.573113

43 RNIO ASCB 00FE4390  CPU 0001 JOBN TRAC3600      IN 1C001001 211D0001 00060390 004DF0F1
           TIME    41968.800126
44 RNIO ASCB 00FE4390  CPU 0001 JOBN TRAC3600     OUT 1E00211D 10010002 00100390 00C9D5D8 F3F6F0F0
           TIME    41968.876992
45 RNIO ASCB 00FE4390  CPU 0001 JOBN TRAC3600      IN 1C001001 211D0002 00180390 00C9D5D8 E4C9D9E8
           TIME    41972.101915
46 RNIO ASCB 00FE4390  CPU 0001 JOBN TRAC3600     OUT 1E00211D 10010003 005B0390 00C4C1E3 C140C2C1
           TIME    41972.125414
47 RNIO ASCB 00FE4390  CPU 0001 JOBN TRAC3600      IN 1C001001 211D0003 00180390 00C9D5D8 E4C9D9E8
           TIME    41991.309495
```

Figure 4-2. VTAM I/O Trace Example (Part 2 of 4)

```
                                                    EXTERNAL TRACE - DD TAPE
48 RNIO ASCB 00FE4390  CPU 0001 JOBN TRAC3600      OUT 1E00211D 10010004 005B0390 00C4C1E3 C140C2C1
           TIME     41991.332963
49 RNIO ASCB 00FE4390  CPU 0001 JOBN TRAC3600       IN 1C001001 211D0004 00180390 00C9D5D8 E4C9D9E8
           TIME     41994.510949

50 RNIO ASCB 00FE4390  CPU 0001 JOBN TRAC3600       IN 1C001C01 211D0017 00180390 00C9D5D8 E4C9D9E8
           TIME     42224.350661
51 RNIO ASCB 00FE4390  CPU 0001 JOBN TRAC3600      OUT 1E00211D 10010018 00580390 00C4C1E3 C140C2C1
           TIME     42224.375301
52 RNIO ASCB 00FE4390  CPU 0001 JOBN TRAC3600       IN 1C001001 211D0018 00180390 00C9D5D8 E4C9D9E8
           TIME     42237.552094
53 RNIO ASCB 00FE4390  CPU 0001 JOBN TRAC3600       IN 1F001001 211D0003 0004EB80 00A1
           TIME     42238.652850
54 RNIO ASCB 00FE4390  CPU 0001 JOBN TRAC3600       IN 1F001001 211D0001 0004EB80 0032
           TIME     42239.251781
55 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM          OUT 1F00211D 10000002 00046B80 000E
           TIME     42253.499154
56 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM           IN 1F001000 211D0002 0004EB80 000E
           TIME     42254.054066


  *** DATE   DAY 168    YEAR 1975            TIME 16.44.23.800573

57 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM          OUT 1F00211C 10000002 00056B80 001202
           TIME     42263.952790
58 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM           IN 1F001000 211C0002 0004EB80 0012
           TIME     42264.555801
59 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM          OUT 1E002000 10000034 00080B80 00010202 211C
           TIME     42264.586587
60 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM           IN 1E001000 20000034 00069B80 00010202
           TIME     42264.955744
61 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM          OUT 1E002000 10000035 00080B80 0001020B 211B
           TIME     42265.202027
62 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM           IN 1E001000 20000035 00069B80 0001020B
           TIME     42270.355864
63 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM          OUT 1E002000 10000036 00080B80 00010202 212F
           TIME     42275.385671
64 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM           IN 1E001000 20000036 00069B80 00010202
           TIME     42275.756677
65 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM          OUT 1E002000 10000037 00080B80 0001020B 212E
           TIME     42275.797796
66 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM           IN 1E001000 20000037 00069B80 0001020B
           TIME     42281.157516

67 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM          OUT 1E002000 1000002F 00080B80 0001020B 2001
           TIME     42895.353348
68 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM           IN 1E001000 2000002F 00069B80 0001020B
           TIME     42898.866996
69 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM          OUT 1E002000 10000030 00080B80 0001020B 2003
           TIME     42898.884574
```

Figure 4-2. VTAM I/O Trace Example (Part 3 of 4)



```
                                                    EXTERNAL TRACE - DD TAPE

70 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM          IN 1F001000 20000030 00069B80 0001020B
            TIME     42902.468647
71 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM          OUT 1E002000 10000031 00080B80 0001020B 2005
            TIME     42902.486794
72 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM          IN 1E001000 20000031 00069B80 0001020B
            TIME     42906.071354
73 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM          OUT 1E002000 10000032 00080B80 0001020B 2007
            TIME     42906.069648
74 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM          IN 1E001000 20000032 00069B80 0001020B
            TIME     42909.671495

75 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM          OUT 1E002000 1000004D 00080B80 0001020B 203D
            TIME     43005.315000
76 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM          IN 1E001000 2000004D 00069B80 0001020B
            TIME     43008.897856
77 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM          OUT 1E002000 1000004E 00080B80 0001020B 2079
            TIME     43009.100051
78 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM          IN 1F001000 2000004E 00069B80 0001020B
            TIME     43012.499105
79 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM          OUT 1E002000 1000004F 00080B80 0001020B 2087
            TIME     43012.558174
80 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM          IN 1E001000 2000004F 00069B80 0001020B
            TIME     43015.899088
81 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM          OUT 1F002000 1000000D 00056B8C 001201
            TIME     43017.031614
82 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM          IN 1F001000 2000000D 0004FE80 0012
            TIME     43017.299425
```

Figure 4-2. VTAM I/O Trace Example (Part 4 of 4)

Example 2: VTAM and GTF Traces

The second example (Figure 4-3) shows all of the VTAM-GTF traces for parts of the process shown in the previous example. The TPIOS buffer trace, control layer (C/L) buffer trace, RNIO trace, and NCP line trace are illustrated in the order they occur.

Trace Entries:

- 1 - 17 – show the PEP activation, etc.
- 18 - 29 – are activations for start-stop lines.
- 30 - 55 – show the SDLC link activation; they also show controller and LU activation and connection.
- 56 – is the first C/L record, the first data received from the application.
- 56 - 76 – show the message flow between the application and the LU.
- 77 - 93 – show the NCP line trace in relation to the data in the PIUs. The exact placement of line trace records relative to RNIO and buffer trace records cannot be depended upon; in general, most or all of the line trace that shows receipt of a message in the NCP will precede the inbound host traces for that message. The RNIO records are omitted from this section of the trace: there should be RNIO records for each PIU, plus one for each line trace entry showing a buffer of line trace data coming into VTAM from the NCP.
- 84 - 122 – show the last data exchanges, the disconnection and deactivation of the 3600, and deactivation of the SDLC link.

The RNIO trace entries in this example can be matched exactly with the entries of the previous example. This example shows how messages can be followed through the network. The GTF I/O and SIO traces are not shown in this example. Running a single terminal at a low message rate, as in this example, would cause almost every RNIO trace to be preceded by several I/O – SIO entries. The sequences are usually as follows:

– Outbound:

GTF	SIO
GTF	I/O – CE, DE
GTF	RNIO OUT

– Inbound:

GTF	I/O – ATTN
GTF	SIO
GTF	I/O – CE, DE, UE
GTF	RNIO IN

All SIO and I/O entries are for the native subchannel address of the 3705. With more message traffic, different sequences may be seen. Very few problems have occurred in the area of the VTAM-NCP channel interface, but a brief explanation is included to avoid confusion.

The following GTF I/O, SIO, and RNIO sequences are possible:

1. Coat-tailing – data is returned from the NCP on the same I/O operation used to send data out, as follows:

   GTF SIO (WRITE CCWs with READ CCW appended)
   GTF I/O CE, DE, UE (UE because some but not all READ CCWs were used)
   RNIO IN
   RNIO IN
   RNIO OUT

   The number of PIUs transferred out and in can vary. If the maximum number of PIUs is sent in (see the MAXBFRU operand of the HOST macro in NCP generation), then the I/O interruption has a status of just CE, DE.

2. More than the maximum number of PIUs were in the NCP ready to be sent to VTAM (assumes the maximum is 3):

   GTF I/O ATTN
   GTF SIO (READ CCW string)
   GTF I/O CE, DE, ATTN
   RNIO IN
   RNIO IN
   RNIO IN
   GTF SIO (read CCW string)
   GTF I/O CE, DE, UE
   RNIO IN

   The first read operation ends normally with an attention, indicating more data in the NCP. This avoids an extra interrupt just to present an attention.

```
                                                    EXTERNAL TRACE - DD TAPE

  *** DATE   DAY 168    YEAR 1975            TIME 16.30.32.113950

1 USRFD FEF ASCB 00FDCD60  JOBN VTAM
             TPIOS OUT ANODE VTAM       FDB  00000000 00B67028 00100000   RSVD 0000   LNG2 00A4   RSVD  00000000 00000000
               REMOTE  DNODE PEP736A    THRH 1C002000 10000000 00006B80 00
                                        TEXT 110103                                    *...                *
             TIME    41690.735400
2 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM         IN 1D000000 20000000 00092800 00500900 400698
             TIME    41690.744573
3 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM        OUT 1F002000 10000001 00066B80 00110103
             TIME    41690.745676
4 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM         IN 1F001000 20000001 000DFB80 00110107 C5D7F7F3
             TIME    41691.003599
5 USRFD FEF ASCB 00FDCD60  JOBN VTAM
             TPIOS IN  ANODE VTAM       FDB  00000000 00B67369 000A0000   RSVD 0812   LNG2 00C0
               REMOTE  DNODE PEP736A    FSB  022C0000 00000000 10002000 00010000 00000000 00000000 00000000 000D0000
                                        THRH 1F001000 20000001 000DFB80 00
                                        TEXT 1101D7C5 D7F7F3F6 C140                    *..PEP736A          *
             TIME    41691.074251
6 USRFD FEF ASCB 00FDCD60  JOBN VTAM
             TPIOS OUT ANODE VTAM       FDB  00000000 00B67348 000E0000   RSVD 0000   LNG2 00A4   RSVD  00000000 00000000
               REMOTE  DNODE PEP736A    THRH 1C002000 10000000 00006B80 00
                                        TEXT A0                                        *.                  *
             TIME    41691.217088
7 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM        OUT 1F002000 10000002 00046B80 00A0
             TIME    41691.226997
8 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM         IN 1F001000 20000002 0004FB80 00A0
             TIME    41691.528884
9 USRFD FEF ASCB 00FDCD60  JOBN VTAM
             TPIOS IN  ANODE VTAM       FDB  00000000 00B67431 00010000   RSVD 0812   LNG2 00C0
               REMOTE  DNODE PEP736A    FSB  022C0000 00000000 10002000 00020000 00000000 00000000 00000000 00040000
                                        THRH 1F001000 20000002 0004FB80 00
                                        TEXT A0                                        *.                  *
             TIME    41691.532498
10 USRFD FEF ASCB 00FDCD60  JOBN VTAM
             TPIOS OUT ANODE VTAM       FDB  00000000 00B67410 00270000   RSVD 0000   LNG2 00A4   RSVD  00000000 00000000
               REMOTE  DNODE PEP736A    THRH 1C002000 10000000 00000B80 00
                                        TEXT 01021120 0001F0F6 61F1F761 F7F54BF1 F6F8F1F6  *.....06/17/75.16816*
                                             4BF3F44B F5F2                             *.34.52             *
             TIME    41691.543192
11 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM        OUT 1F002000 10000001 001D0B80 00010211 200001F0
             TIME    41691.549546
12 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM         IN 1E001000 20000001 00069B80 00010211
             TIME    41691.799592
13 USRFD FEF ASCB 00FDCD60  JOBN VTAM
             TPIOS IN  ANODE VTAM       FDB  00000000 00B67369 00030000   RSVD 0812   LNG2 00C0
               REMOTE  DNODE PEP736A    FSB  022C0000 00000000 10002000 00010000 00000000 00000000 00000000 00060000
                                        THRH 1E001000 20000001 00069B80 00
                                        TEXT 010211                                    *...                *
             TIME    41691.803107
14 USRFD FEF ASCB 00FDCD60  JOBN VTAM
             TPIOS OUT ANODE VTAM       FDB  00000000 00B67348 00150000   RSVD 0000   LNG2 00A4   RSVD  00000000 00000000
               REMOTE  DNODE PEP736A    THRH 1C002000 10000000 00006B80 00
                                        TEXT 01021120 00050000                         *.........          *
```

Figure 4-3. VTAM and GTF Traces (Part 1 of 13)

```
                                                    EXTERNAL TRACE - DD TAPE

          TIME     41691.814362
15 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM          OUT 1E002000 10000002 00080B80 00010211 20000500
          TIME     41691.818728
16 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM           IN 1E001000 20000002 000A9F90 00100200 00010211
          TIME     41692.699679
17 USRFD FEF ASCB 00FDCD60  JOBN VTAM
           TPIOS IN  ANODE VTAM        FDB  00000000 00B67431 00070000   RSVD 0812   LNG2 00C0
             REMOTE  DNODE PEP736A     FSB  022C0000 00000000 10002000 00020000 00000000 00000000 00000000 00CA0000
                                       THRH 1E001000 20000002 000A9F90 00
                                       TEXT 10020000 010211                                  *.......             *
          TIME     41692.103190
18 USRFD FEF ASCB 00FDCD60  JOBN VTAM
           TPIOS OUT ANODE VTAM        FDB  00000000 00B67410 00120000   RSVD 0000   LNG2 00A4   RSVD  00000000 00000000
             REMOTE  DNODE PEP736A     THRH 1C002000 10000000 00000B80 00
                                       TEXT 01020A20 01                                      *......             *
          TIME     41692.112675
19 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM          OUT 1E002000 10000003 00080B80 0001020A 2001
          TIME     41692.117212
20 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM           IN 1E001000 20000003 00069B80 0001020A
          TIME     41692.399922
21 USRFD FEF ASCB 00FDCD60  JOBN VTAM
           TPIOS IN  ANODE VTAM        FDB  00000000 00B67369 00030000   RSVD 0812   LNG2 00C0
             REMOTE  DNODE PEP736A     FSB  022C0000 00000000 10002000 00030000 00000000 00000000 00000000 00060000
                                       THRH 1E001000 20000003 00069B80 00
                                       TEXT 01020A                                           *...                 *
          TIME     41692.403432
22 USRFD FEF ASCB 00FDCD60  JOBN VTAM
           TPIOS OUT ANODE VTAM        FDB  00000000 00B67348 00120000   RSVD 0000   LNG2 00A4   RSVD  00000000 00000000
             REMOTE  DNODE PEP736A     THRH 1C002000 10000000 00000B80 00
                                       TEXT 01020A20 03                                      *......             *
          TIME     41692.413215
23 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM          OUT 1E002000 10000004 00080B80 0001020A 2003
          TIME     41692.417538
24 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM           IN 1E001000 20000004 00069B80 0001020A
          TIME     41692.700031
25 USRFD FEF ASCB 00FDCD60  JOBN VTAM
           TPIOS IN  ANODE VTAM        FDB  00000000 00B67431 00030000   RSVD 0812   LNG2 00C0
             REMOTE  DNODE PEP736A     FSB  022C0000 00000000 10002000 00040000 00000000 00000000 00000000 00060000
                                       THRH 1E001000 20000004 00069B80 00
                                       TEXT 01020A                                           *...                 *
          TIME     41692.703547
26 USRFD FEF ASCB 00FDCD60  JOBN VTAM
           TPIOS OUT ANODE VTAM        FDB  00000000 00B67410 00120000   RSVD 0000   LNG2 00A4   RSVD  00000000 00000000
             REMOTE  DNODE PEP736A     THRH 1C002000 10000000 00000B80 00
                                       TEXT 01020A20 05                                      *......             *
          TIME     41692.713339
27 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM          OUT 1E002000 10000005 00080B80 0001020A 2005
          TIME     41692.723718
28 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM           IN 1E001000 20000005 00069B80 0001020A
          TIME     41693.000187
```

Figure 4-3. VTAM and GTF Traces (Part 2 of 13)

```
                                                    EXTERNAL TRACE - DD TAPE

29 USRFD FEF ASCB 00FDCD60  JOBN VTAM
              TPIOS IN  ANODE VTAM        FDB  00000000 00B67369 00030000   RSVD 0812   LNG2 00C0
                REMOTE  DNODE PEP736A     FSB  022C0000 00000000 10002000 00050000 00000000 00000000 00000000 00060000
                                          THRH 1E001000 20000005 00069B80 00
                                          TEXT 01020A                                          *...                *
              TIME     41693.003693

30 USRFD FEF ASCB 00FDCD60  JOBN VTAM
              TPIOS OUT ANODE VTAM        FDB  00000000 00B67410 00120000   RSVD 0000   LNG2 00A4   RSVD  00000000 00000000
                REMOTE  DNODE PEP736A     THRH 1C002000 10000000 00000B80 00
                                          TEXT 01020A21 1B                                     *.....              *
              TIME     41746.536984
31 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM         OUT 1E002000 1000002A 00080B80 0001020A 211B
              TIME     41746.543962
32 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM          IN 1E001000 2000002A 00069B80 0001020A
              TIME     41746.823375
33 USRFD FEF ASCB 00FDCD60  JOBN VTAM
              TPIOS IN  ANODE VTAM        FDB  00000000 00B67369 00030000   RSVD 0812   LNG2 00C0
                REMOTE  DNODE PEP736A     FSB  022C0000 00000000 10002000 002A0000 00000000 00000000 00060000
                                          THRH 1E001000 2000002A 00069B80 00
                                          TEXT 01020A                                          *...                *
              TIME     41746.826948
34 USRFD FEF ASCB 00FDCD60  JOBN VTAM
              TPIOS OUT ANODE VTAM        FDB  00000000 00B67348 00120000   RSVD 0000   LNG2 00A4   RSVD  00000000 00000000
                REMOTE  DNODE PEP736A     THRH 1C002000 10000000 00000B80 00
                                          TEXT 01020121 1C                                     *.....              *
              TIME     41746.837796
35 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM         OUT 1E002000 1000002B 00080B80 00010201 211C
              TIME     41746.842293
36 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM          IN 1E001000 2000002B 00069B80 00010201
              TIME     41747.123477
37 USRFD FEF ASCB 00FDCD60  JOBN VTAM
              TPIOS IN  ANODE VTAM        FDB  00000000 00B67431 00030000   RSVD 0812   LNG2 00C0
                REMOTE  DNODE PEP736A     FSB  022C0000 00000000 10002000 002B0000 00000000 00000000 00060000
                                          THRH 1E001000 2000002B 00069B80 00
                                          TEXT 010201                                          *...                *
              TIME     41747.127060
38 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM          IN 1C001000 20000001 00090B00 00010280 211C01
              TIME     41749.324865
39 USRFD FEF ASCB 00FDCD60  JOBN VTAM
              TPIOS IN  ANODE VTAM        FDB  00000000 00B672A1 00060000   RSVD 0812   LNG2 00C0
                REMOTE  DNODE PEP736A     FSB  022C0000 00000000 10002000 00010000 00000000 00000000 00090000
                                          THRH 1C001000 20000001 00090B00 00
                                          TEXT 01028021 1C01                                   *......            *
              TIME     41749.328496
40 USRFD FEF ASCB 00FDCD60  JOBN VTAM
              TPIOS OUT ANODE VTAM        FDB  00000000 00B67280 00100000   RSVD 0000   LNG2 00A4   RSVD  00000000 00000000
                REMOTE  DNODE CL1LOE7     THRH 1C0C211C 10000005 00006B8C 00
                                          TEXT 110101                                          *...                *
              TIME     41749.340640
41 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM         OUT 1FC0211C 10000001 00066B80 00110101
              TIME     41749.345127
```

Figure 4-3. VTAM and GTF Traces (Part 3 of 13)

```
                                                    EXTERNAL TRACE - DD TAPE

42 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM          IN 1F001000 211C0001 000DEB80 0011C1F3 F6F0F0F0
           TIME    41750.025422
43 USRFD FEF ASCB 00FDCD60  JOBN VTAM
            TPIOS IN  ANODE VTAM       FDB  00000000 00B67369 000A0000   RSVD 0812   LNG2 00C0
              REMOTE  DNODE CL1LOE7    FSB  022C0000 00000000 1000211C 00010000 00000000 00000000 00000000 00000000
                                       THRH 1F001000 211C0001 000DEB80 00
                                       TEXT 1101F3F6 F0F0F0F0 F0F0                        *..36000000          *
           TIME    41750.028979
44 USRFD FEF ASCB 00FDCD60  JOBN VTAM
            TPIOS OUT ANODE VTAM       FDB  00000000 00B67348 00100000   RSVD 0000   LNG2 00A4   RSVD  000 0000 00000000
              REMOTE  DNODE U1C1LOE7   THRH 1C00211D 10000000 00006B80 00
                                       TEXT 0D0101                                        *...                 *
           TIME    41756.278253
45 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM        OUT 1F00211D 10000001 00066B80 000D0101
           TIME    41758.282656
46 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM         IN 1F001000 211D0001 0005EB80 000D01
           TIME    41758.628781
47 USRFD FEF ASCB 00FDCD60  JOBN VTAM
            TPIOS IN  ANODE VTAM       FDB  00000000 00B672A1 00020000   RSVD 0812   LNG2 00C0
              REMOTE  DNODE U1C1LOE7   FSB  022C0000 00000000 1000211D 00010000 00000000 00000000 00000000 00050000
                                       THRH 1F001000 211D0001 0005EB80 00
                                       TEXT 0D01                                          *..                  *
           TIME    41758.632307
48 USRFD FEF ASCB 00FE4390  JOBN TRAC3600
            TPIOS OUT ANODE P1VT3600   FDB  00000000 00B67280 00310000   RSVD 0000   LNG2 00A4   RSVD  0000C000 00000000
              REMOTE  DNODE U1C1LOE7   THRH 1C00211D 10010000 00006B80 00
                                       TEXT 31010302 FF910000 D9C5C3D6 D9C44040 00000040  *..... ..RECORD  ... *
                                            40404040 40404008 D7F1E5E3 F3F6F0F0           *       .P1VT3600    *
           TIME    41787.389730
49 RNIO ASCB 00FE4390  CPU 0001 JOBN TRAC3600     OUT 1F00211D 10010001 00276B80 00310103 02FF910C
           TIME    41787.394270
50 RNIO ASCB 00FE4390  CPU 0001 JOBN TRAC3600      IN 1F001001 211D0001 0004EB80 0031
           TIME    41788.042000


   *** DATE   DAY 168     YEAR 1975            TIME 16.36.27.900691                                                  ***

51 USRFD FEF ASCB 00FE4390  JOBN TRAC3600
            TPIOS IN  ANODE P1VT3600   FDB  00000000 00B67369 00010000   RSVD 0812   LNG2 00C0
              REMOTE  DNODE U1C1LOE7   FSB  022C0000 00000000 1001211D 00010000 00000000 00000000 00000000 00040000
                                       THRH 1F001001 211D0001 0004EB80 00
                                       TEXT 31                                            *.                   *
           TIME    41788.044994
52 USRFD FEF ASCB 00FE4390  JOBN TRAC3600
            TPIOS OUT ANODE P1VT3600   FDB  00000000 00B67348 00FE0000   RSVD 0000   LNG2 00A4   RSVD  00000000 00000000
              REMOTE  DNODE U1C1LOE7   THRH 1C00211D 10010000 00006B80 00
                                       TEXT A0                                            *.                   *
           TIME    41788.093294
53 RNIO ASCB 00FE4390  CPU 0001 JOBN TRAC3600     OUT 1F00211D 1C010002 00046B8C 00A0
           TIME    41788.097764
```

Figure 4-3. VTAM and GTF Traces (Part 4 of 13)

```
                                                        EXTERNAL TRACE - DD TAPE

54 RNIO ASCB 00FE4390  CPU 0001 JOBN TRAC3600        IN 1F001001 211D0002 0004EB80 00AC
           TIME    41788.642254
55 USRFD FEF ASCB 00FE4390  JOBN TRAC3600
            TPIOS IN   ANODE P1VT3600   FDB  00000000 00E672A1 0001000C   RSVD C812   LNG2 00C0
              REMOTE   DNODE U1C1L0E7   FSE  022C0000 00000000 1001211D 00020000 000.0000 00000000 000 000 00040.00
                                        THRH 1F001001 211D0002 0004EB80 00
                                        TEXT A0                                          *.              *
           TIME    41788.645266
56 USRFD FF1 ASCB 00FE4390  JOBN TRAC3600
           C/L  OUT   ANODE P1VT3600    TEXT D9C5C1C4 E84B40E3 E8D7C540 7DD7D9D6 D4D7E37D  *READY. TYPE 'PROMPT'*
                      DNODE U1C1L0E7         40C6D6D9 40C8C5D3 D74B4040                   * FOR HELP.         *
           TIME    41789.259277
57 USRFD FEF ASCB 00FE4390  JOBN TRAC3600
            TPIOS OUT ANODE P1VT3600    FDB  00000000 00E6728C 002D0000   RSVD 0000   LNG2 00A4   RSVD  00000000 00000000
              REMOTE  DNODE U1C1L0E7    THRH 1C00211D 10010000 00000390 00
                                        TEXT D9C5C1C4 E84B40E3 E8D7C540 7DD7D9D6 D4D7E37D  *READY. TYPE 'PROMPT'*
                                             40C6D6D9 40C8C5D3 D74B4040                   * FOR HELP.         *
           TIME    41789.435172
58 RNIO ASCB 00FE4390  CPU 0001 JOBN TRAC3600       OUT 1E00211D 10010001 00230390 00D9C5C1 C4E84B40
           TIME    41789.441206
```

```
59 RNIO ASCB 00FE4390  CPU 0001 JOBN TRAC3600        IN 1C001001 211D0001 00060390 004DF0F1
           TIME    41968.800126
60 USRFD FEF ASCB 00FE4390  JOBN TRAC3600
            TPIOS IN   ANODE P1VT3600   FDB  00000000 00E67111 00030000   RSVD C812   LNG2 00C0
              REMOTE   DNODE U1C1L0E7   FSB  022C0000 000.000 1001211D 00010000 00000000 00000000 00000000 00060000
                                        THRH 1C001001 211D0001 00060390 00
                                        TEXT 4DF0F1                                      *(01            *
           TIME    41968.805427
61 USRFD FF1 ASCB 00FE4390  JOBN TRAC3600
            C/L  IN    ANODE P1VT3600   TEXT 4DF0F1                                      *(01            *
                       DNODE U1C1L0E7
           TIME    41968.857165
62 USRFD FF1 ASCB 00FE4390  JOBN TRAC3600
            C/L  OUT   ANODE P1VT3600   TEXT C9D5D8F3 F6F0F040 D9C5C1C4 E8                *INQ3600 READY     *
                       DNODE U1C1L0E7
           TIME    41968.867686
63 USRFD FEF ASCB 00FE4390  JOBN TRAC3600
            TPIOS OUT ANODE P1VT3600    FDB  00000000 00E670FC 001A0000   RSVD 0000   LNG2 00A4   RSVD  00000000 00000000
              REMOTE  DNODE U1C1L0E7    THRH 1C00211D 10010000 00000390 00
                                        TEXT C9D5D8F3 F6F0F040 D9C5C1C4 E8                *INQ3600 READY     *
           TIME    41968.871201
64 RNIO ASCB 00FE4390  CPU 0001 JOBN TRAC3600       OUT 1E00211D 10010002 00100390 00C9D5D8 F3F6F0F0
           TIME    41968.876992
65 RNIO ASCB 00FE4390  CPU 0001 JOBN TRAC3600        IN 1C001001 211D0002 00180390 00C9D5D8 E4C9D9E8
           TIME    41972.101915
```

Figure 4-3. VTAM and GTF Traces (Part 5 of 13)

```
                                                    EXTERNAL TRACE - DD TAPE
66 USRFD FEF ASCB 00FE4390  JOBN TRAC3600
              TPIOS IN  ANODE P1VT3600    FDB  00000000 00B671D9 00150000   RSVD 0812   LNG2 00C0
                REMOTE  DNODE U1C1L0E7    FSB  022C0000 00000000 1001211D 00020000 00000000 00000000 00000000 00180000
                                          THRH 1C001001 211D0002 00180390 00
                                          TEXT C9D5D8E4 C9D9E840 F3F6F0F0 40F0F0F1 40E4F1C3  *INQUIRY 3600 001 U1C*
                                               F1                                            *1                    *
              TIME    41972.104925
67 USRFD FF1 ASCB 00FE4390  JOBN TRAC3600
              C/L  IN   ANODE P1VT3600    TEXT C9D5D8E4 C9D9E840 F3F6F0F0 40F0F0F1 40E4F1C3  *INQUIRY 3600 001 U1C*
                        DNODE U1C1L0E7         F1                                            *1                    *
              TIME    41972.108208
68 USRFD FF1 ASCB 00FE4390  JOBN TRAC3600
              C/L  OUT  ANODE P1VT3600    TEXT C4C1E3C1 40C2C1E2 C540C1D7 D7D3C9C3 C1E3C9D6  *DATA BASE APPLICATIO*
                        DNODE U1C1L0E7         D540D9C5 E2D7D6D5 E2C540D4 C5E2E2C1 C7C54040  *N RESPONSE MESSAGE  *
                                               40404040 40404040 40404040 40404040 40404040  *                    *
                                               40404040 40404040 40404040 40404040 40404040  *                    *
                                               40404040 40404040                             *                    *
              TIME    41972.116250
69 USRFD FEF ASCB 00FE4390  JOBN TRAC3600
              TPIOS OUT ANODE P1VT3600    FDB  00000000 00B671B8 00650000   RSVD 0000   LNG2 00A4   RSVD 00000000 00000000
                REMOTE  DNODE U1C1L0E7    THRH 1C00211D 10010000 00000390 00
                                          TEXT C4C1E3C1 40C2C1E2 C540C1D7 D7D3C9C3 C1E3C9D6  *DATA BASE APPLICATIO*
                                               D540D9C5 E2D7D6D5 E2C540D4 C5E2E2C1 C7C54040  *N RESPONSE MESSAGE  *
                                               40404040 40404040 40404040 40404040 40404040  *                    *
                                               40404040 40404040 40404040 40404040 40404040  *                    *
                                               40404040 40404040                             *                    *
              TIME    41972.119800
70 RNIO ASCB 00FE4390  CPU 0001 JOBN TRAC3600     OUT 1E00211D 10010003 005B0390 00C4C1E3 C14CC2C1
              TIME    41972.125414
71 RNIO ASCB 00FE4390  CPU 0001 JOBN TRAC3600      IN 1C001001 211D0003 00180390 00C9D5D8 E4C9D9E8
              TIME    41991.309495
72 USRFD FEF ASCB 00FE4390  JOBN TRAC3600
              TPIOS IN  ANODE P1VT3600    FDB  00000000 00B67111 00150000   RSVD 0812   LNG2 00C0
                REMOTE  DNODE U1C1L0E7    FSB  022C0000 00000000 1001211D 00030000 00000000 00000000 00000000 00180000
                                          THRH 1C001001 211D0003 00180390 00
                                          TEXT C9D5D8E4 C9D9E840 F3F6F0F2 40F0F0F2 40E4F1C3  *INQUIRY 3600 002 U1C*
                                               F1                                            *1                    *
              TIME    41991.312497
73 USRFD FF1 ASCB 00FE4390  JOBN TRAC3600
              C/L  IN   ANODE P1VT3600    TEXT C9D5D8E4 C9D9E840 F3F6F0F0 40F0F0F2 40E4F1C3  *INQUIRY 3600 002 U1C*
                        DNODE U1C1L0E7         F1                                            *1                    *
              TIME    41991.315783
74 USRFD FF1 ASCB 00FE4390  JOBN TRAC3600
              C/L  OUT  ANODE P1VT3600    TEXT C4C1E3C1 40C2C1E2 C540C1D7 D7D3C9C3 C1E3C9D6  *DATA BASE APPLICATIO*
                        DNODE U1C1L0E7         D540D9C5 E2D7D6D5 E2C540D4 C5E2E2C1 C7C54040  *N RESPONSE MESSAGE  *
                                               40404040 40404040 40404040 40404040 40404040  *                    *
                                               40404040 40404040 40404040 40404040 40404040  *                    *
                                               40404040 40404040                             *                    *
              TIME    41991.323826
```

Figure 4-3. VTAM and GTF Traces (Part 6 of 13)

```
                                                    EXTERNAL TRACE - DD TAPE

75 USRFD FEF ASCB 00FE439C  JOBN TRAC3600
          TPIOS OUT ANODE P1VT3600   FDB  00000000 00B670F0 00650000   RSVD 0000   LNG2 00A4   RSVD  00000000 00000000
            REMOTE  DNODE U1C1LCE7   THRH 1C00211D 1001[illegible]0 [illegible]0003490 C0
                                     TEXT C4C1E3C1 40C2C1E2 C540C1D7 D7D3C9C3 C1E3C9D6  *DATA BASE APPLICATIO*
                                          D54CD9C5 E2D7D6D5 E2C540D4 C5E2E2C1 C7C54040  *N RESPONSE MESSAGE  *
                                          40404040 40404040 40404040 40404040 40404040  *                    *
                                          40404040 40404040 40404040 40404040 40404040  *                    *
                                          40404040 40404040                           *                    *
          TIME     41991.327379
76 RNIO ASCB 00FE439C  CPU 0001 JOBN TRAC3600     OUT 1E00211D 10010004 0058C390 00C4C1E3 C140C2C1
          TIME     41991.332963

77                                LCD 9  PCF 9  TIME 07  SCF 45  PDF 7E      LCD 9  PCF 9  TIME 07  SCF 45  PDF 7E
                                  LCD 9  PCF 9  TIME 07  SCF 40  PDF C1      LCD 9  PCF 9  TIME 07  SCF 40  PDF F1
                                  LCD 9  PCF 9  TIME 07  SCF 40  PDF 33      LCD 9  PCF 9  TIME 07  SCF 40  PDF 3A
                                  LCD 9  PCF 9  TIME 07  SCF 45  PDF 7E      LCD 9  PCF 9  TIME 07  SCF 45  PDF FF
                                  LCD 9  PCF 5  TIME 07  SCF 45  PDF FF
          TIME     42085.939817
78 USRFD FF2 ASCB 00FDCD60  JOBN VTAM
           LINE       DNODE P#P736A       EP 00       TIME 08
                                  LCD 9  PCF 6  TIME 09  SCF 0D  PDF FF      LCD 9  PCF 7  TIME 09  SCF 49  PDF C1
                                  LCD 9  PCF 7  TIME 09  SCF 49  PDF 5E      LCD 9  PCF 7  TIME 09  SCF 49  PDF 2C
                                  LCD 9  PCF 7  TIME 09  SCF 49  PDF 00      LCD 9  PCF 7  TIME 09  SCF 49  PDF 01
                                  LCD 9  PCF 7  TIME 09  SCF 49  PDF 01      LCD 9  PCF 7  TIME 09  SCF 49  PDF 00
                                  LCD 9  PCF 7  TIME 09  SCF 49  PDF 0C      LCD 9  PCF 7  TIME 09  SCF 49  PDF 03
                                  LCD 9  PCF 7  TIME 09  SCF 49  PDF 90      LCD 9  PCF 7  TIME 09  SCF 49  PDF 00
                                  LCD 9  PCF 7  TIME 09  SCF 49  PDF C9      LCD 9  PCF 7  TIME 09  SCF 49  PDF D5
                                  LCD 9  PCF 7  TIME 09  SCF 49  PDF D8      LCD 9  PCF 7  TIME 09  SCF 49  PDF E4
                                  LCD 9  PCF 7  TIME 09  SCF 49  PDF C9      LCD 9  PCF 7  TIME 09  SCF 49  PDF D9
                                  LCD 9  PCF 7  TIME 09  SCF 49  PDF E8
          TIME     42085.942128
79 USRFD FF2 ASCB 00FDCD60  JOBN VTAM
           LINE       DNODE P#P736A       EP 00       TIME 08
                                  LCD 9  PCF 7  TIME 09  SCF 49  PDF 40      LCD 9  PCF 7  TIME 09  SCF 49  PDF F3
                                  LCD 9  PCF 7  TIME 09  SCF 49  PDF F6      LCD 9  PCF 7  TIME 09  SCF 49  PDF F0
                                  LCD 9  PCF 7  TIME 09  SCF 49  PDF F0      LCD 9  PCF 7  TIME 0A  SCF 49  PDF 40
                                  LCD 9  PCF 7  TIME 0A  SCF 49  PDF F0      LCD 9  PCF 7  TIME 0A  SCF 49  PDF F1
                                  LCD 9  PCF 7  TIME 0A  SCF 49  PDF F1      LCD 9  PCF 7  TIME 0A  SCF 49  PDF 40
                                  LCD 9  PCF 7  TIME 0A  SCF 49  PDF E4      LCD 9  PCF 7  TIME 0A  SCF 49  PDF F1
                                  LCD 9  PCF 7  TIME 0A  SCF 49  PDF C3      LCD 9  PCF 7  TIME 0A  SCF 49  PDF F1
                                  LCD 9  PCF 7  TIME 0A  SCF 49  PDF CB      LCD 9  PCF 7  TIME 0A  SCF 49  PDF 85
                                  LCD 9  PCF 6  TIME 0A  SCF 0D  PDF 85      LCD 9  PCF 9  TIME 0A  SCF 45  PDF 00
                                  LCD 9  PCF 9  TIME 0A  SCF 45  PDF 7E
          TIME     42085.942805
80 USRFD FF2 ASCB 00FDCD60  JOBN VTAM
           LINE       DNODE P#P736A       EP 00       TIME 08
                                  LCD 9  PCF 9  TIME 0A  SCF 45  PDF 7E      LCD 9  PCF 9  TIME 0A  SCF 45  PDF 7E
                                  LCD 9  PCF 9  TIME 0A  SCF 40  PDF C1      LCD 9  PCF 9  TIME 0A  SCF 40  PDF 11
                                  LCD 9  PCF 9  TIME 0A  SCF 40  PDF 3D      LCD 9  PCF 9  TIME 0A  SCF 40  PDF 0D
                                  LCD 9  PCF 9  TIME 0A  SCF 45  PDF 7E      LCD 9  PCF 9  TIME 0A  SCF 45  PDF FF
                                  LCD 9  PCF 5  TIME 0A  SCF 45  PDF FF
          TIME     42085.943443
```

Figure 4-3. VTAM and GTF Traces (Part 7 of 13)

```
                                                    EXTERNAL TRACE - DD TAPE

81 USRFD FEF ASCB 00FE4390  JOBN TRAC3600
             TPIOS IN   ANODE P1VT3600    FDB  00000000 00867369 00150000   RSVD 0812   LNG2 00C0
               REMOTE   DNODE U1C1L0E7    FSB  022C0000 00000000 1001211D 000C0000 00000000 00000000 00000000 00180000
                                          THRH 1C001001 211D000C 00180390 0C
                                          TEXT C9D5D8E4 C9D9E840 F3F6F0F0 40F0F1F1 40E4F1C3  *INQUIRY 3600 011 U1C*
                                               F1                                            *1                    *
             TIME     42085.948765
   USRFD FF1 ASCB 00FE4390  JOBN TRAC3600
             C/L   IN   ANODE P1VT3600    TEXT C9D5D8E4 C9D9E840 F3F6F0F0 40F0F1F1 40E4F1C3  *INQUIRY 3600 011 U1C*
                        DNODE U1C1L0E7         F1                                            *1                    *
             TIME     42085.956648
   USRFD FF1 ASCB 00FE4390  JOBN TRAC3600
             C/L  OUT   ANODE P1VT3600    TEXT C4C1E3C1 40C2C1E2 C540C1D7 D7D3C9C3 C1E3C9D6  *DATA BASE APPLICATIO*
                        DNODE U1C1L0E7         D540D9C5 E2D7D6D5 E2C540D4 C5E2E2C1 C7C54040  *N RESPONSE MESSAGE  *
                                               40404040 40404040 40404040 40404040 40404040  *                    *
                                               40404040 40404040 40404040 40404040 40404040  *                    *

82 USRFD FEF ASCB 00FE4390  JOBN TRAC3600
             TPIOS OUT  ANODE P1VT3600    FDB  00000000 00867348 00650000   RSVD 0000   LNG2 00A4   RSVD  00000000 00000000
               REMOTE   DNODE U1C1L0E7    THRH 1C00211D 10010000 00000390 C0
                                          TEXT C4C1E3C1 40C2C1E2 C540C1D7 D7D3C9C3 C1E3C9D6  *DATA BASE APPLICATIO*
                                               D540D9C5 E2D7D6D5 E2C540D4 C5E2E2C1 C7C54040  *N RESPONSE MESSAGE  *
                                               40404040 40404040 40404040 40404040 40404040  *                    *
                                               40404040 40404040 40404040 40404040 40404040  *                    *
                                               40404040 40404040                             *                    *
             TIME     42085.970723
83 USRFD FF2 ASCB 00FDCD60  JOBN VTAM
             LINE       DNODE PEP736A       EP C0       TIME 0E
                              LCD 9  PCF 6  TIME 0C  SCF 0D  PDF FF      LCD 9  PCF 7  TIME 0C  SCF 49  PDF C1
                              LCD 9  PCF 7  TIME 0C  SCF 49  PDF 51      LCD 9  PCF 7  TIME 0C  SCF 49  PDF 39
                              LCD 9  PCF 7  TIME 0C  SCF 49  PDF 9F      LCD 9  PCF 6  TIME 0C  SCF 0D  PDF 9F
                              LCD 9  PCF 9  TIME 0C  SCF 45  PDF 00      LCD 9  PCF 9  TIME 0C  SCF 45  PDF 7E
                              LCD 9  PCF 9  TIME 0C  SCF 45  PDF 7E      LCD 9  PCF 9  TIME 0C  SCF 45  PDF 7E
                              LCD 9  PCF 9  TIME 0C  SCF 40  PDF C1      LCD 9  PCF 9  TIME 0C  SCF 40  PDF 04
                              LCD 9  PCF 9  TIME 0C  SCF 40  PDF 2A      LCD 9  PCF 9  TIME 0C  SCF 40  PDF 00
                              LCD 9  PCF 9  TIME 0C  SCF 40  PDF 01      LCD 9  PCF 9  TIME 0C  SCF 40  PDF C1
                              LCD 9  PCF 9  TIME 0C  SCF 40  PDF 00      LCD 9  PCF 9  TIME 0C  SCF 40  PDF 0D
                              LCD 9  PCF 9  TIME 0C  SCF 40  PDF 03
             TIME     42086.537157
84 USRFD FF2 ASCB 00FDCD60  JOBN VTAM
             LINE       DNODE PEP736A       EP 00       TIME 0E
                              LCD 9  PCF 9  TIME 0C  SCF 40  PDF 90      LCD 9  PCF 9  TIME 0C  SCF 40  PDF 00
                              LCD 9  PCF 9  TIME 0C  SCF 40  PDF C4      LCD 9  PCF 9  TIME 0C  SCF 40  PDF C1
                              LCD 9  PCF 9  TIME 0C  SCF 40  PDF E3      LCD 9  PCF 9  TIME 0C  SCF 40  PDF C1
                              LCD 9  PCF 9  TIME 0D  SCF 40  PDF 40      LCD 9  PCF 9  TIME 0D  SCF 40  PDF C2
                              LCD 9  PCF 9  TIME 0D  SCF 40  PDF C1      LCD 9  PCF 9  TIME 0D  SCF 40  PDF E2
                              LCD 9  PCF 9  TIME 0D  SCF 40  PDF C5      LCD 9  PCF 9  TIME 0D  SCF 40  PDF 40
                              LCD 9  PCF 9  TIME 0D  SCF 40  PDF C1      LCD 9  PCF 9  TIME 0D  SCF 40  PDF D7
                              LCD 9  PCF 9  TIME 0D  SCF 40  PDF D7      LCD 9  PCF 9  TIME 0D  SCF 40  PDF D3
                              LCD 9  PCF 9  TIME 0D  SCF 40  PDF C9      LCD 9  PCF 9  TIME 0D  SCF 40  PDF C3
                              LCD 9  PCF 9  TIME 0D  SCF 40  PDF C1
             TIME     42086.537843
```

Figure 4-3. VTAM and GTF Traces (Part 8 of 13)

```
                                                                        EXTERNAL TRACE - DD TAPE
85 USRFD FF2 ASCB 00FDCD60  JOBN VTAM
             LINE       DNODE PEP736A        EP 00        TIME 0E
                                LCD 9  PCF 9  TIME 0D  SCF 40  PDF E3      LCD 9  PCF 9  TIME 0D  SCF 40  PDF C9
                                LCD 9  PCF 9  TIME 0D  SCF 40  PDF D6      LCD 9  PCF 9  TIME 0D  SCF 40  PDF D5
                                LCD 9  PCF 9  TIME 0D  SCF 40  PDF 40      LCD 9  PCF 9  TIME 0D  SCF 40  PDF D9
                                LCD 9  PCF 9  TIME 0D  SCF 40  PDF C5      LCD 9  PCF 9  TIME 0D  SCF 40  PDF E2
                                LCD 9  PCF 9  TIME 0D  SCF 40  PDF D7      LCD 9  PCF 9  TIME 0D  SCF 40  PDF D6
                                LCD 9  PCF 9  TIME 0D  SCF 40  PDF D5      LCD 9  PCF 9  TIME 0D  SCF 40  PDF E2
                                LCD 9  PCF 9  TIME 0D  SCF 40  PDF C5      LCD 9  PCF 9  TIME 0D  SCF 40  PDF 40
                                LCD 9  PCF 9  TIME 0D  SCF 40  PDF D4      LCD 9  PCF 9  TIME 0D  SCF 40  PDF C5
                                LCD 9  PCF 9  TIME 0D  SCF 40  PDF E2
            TIME    42086.538504
86 USRFD FF2 ASCB 00FDCD60  JOBN VTAM
             LINE       DNODE PEP736A        EP 00        TIME 11
                                LCD 9  PCF 9  TIME 0E  SCF 40  PDF E2      LCD 9  PCF 9  TIME 0E  SCF 40  PDF C1
                                LCD 9  PCF 9  TIME 0E  SCF 40  PDF C7      LCD 9  PCF 9  TIME 0E  SCF 40  PDF 42
                                LCD 9  PCF 9  TIME 0E  SCF 40  PDF 01      LCD 9  PCF 9  TIME 0E  SCF 45  PDF 7E
                                LCD 9  PCF 9  TIME 0E  SCF 45  PDF 7E      LCD 9  PCF 9  TIME 0E  SCF 45  PDF 7E
                                LCD 9  PCF 9  TIME 0E  SCF 45  PDF 7E      LCD 9  PCF 9  TIME 0E  SCF 40  PDF C1
                                LCD 9  PCF 9  TIME 0E  SCF 40  PDF 11      LCD 9  PCF 9  TIME 0E  SCF 40  PDF 3D
                                LCD 9  PCF 9  TIME 0E  SCF 40  PDF DD      LCD 9  PCF 9  TIME 0E  SCF 45  PDF 7E
                                LCD 9  PCF 9  TIME 0E  SCF 45  PDF FF      LCD 9  PCF 5  TIME 0E  SCF 45  PDF FF
                                LCD 9  PCF 6  TIME 10  SCF 0D  PDF FF      LCD 9  PCF 7  TIME 10  SCF 49  PDF C1
                                LCD 9  PCF 7  TIME 10  SCF 49  PDF 71
            TIME    42086.540840
87 USRFD FF2 ASCB 00FDCD60  JOBN VTAM
             LINE       DNODE PEP736A        EP 00        TIME 11
                                LCD 9  PCF 7  TIME 10  SCF 49  PDF 3B      LCD 9  PCF 7  TIME 10  SCF 49  PDF BE
                                LCD 9  PCF 6  TIME 10  SCF 0D  PDF BE      LCD 9  PCF 9  TIME 10  SCF 45  PDF C0
                                LCD 9  PCF 9  TIME 10  SCF 45  PDF 7E      LCD 9  PCF 9  TIME 10  SCF 45  PDF 7E
                                LCD 9  PCF 9  TIME 10  SCF 45  PDF 7E      LCD 9  PCF 9  TIME 10  SCF 40  PDF C1
                                LCD 9  PCF 9  TIME 10  SCF 40  PDF 06      LCD 9  PCF 9  TIME 10  SCF 40  PDF 26
                                LCD 9  PCF 9  TIME 10  SCF 40  PDF 00      LCD 9  PCF 9  TIME 10  SCF 40  PDF 01
                                LCD 9  PCF 9  TIME 10  SCF 40  PDF 01      LCD 9  PCF 9  TIME 10  SCF 40  PDF 00
                                LCD 9  PCF 9  TIME 10  SCF 40  PDF 00      LCD 9  PCF 9  TIME 10  SCF 40  PDF C5
                                LCD 9  PCF 9  TIME 10  SCF 40  PDF 40      LCD 9  PCF 9  TIME 10  SCF 40  PDF 40
                                LCD 9  PCF 9  TIME 10  SCF 40  PDF 40
            TIME    42086.541493
88 USRFD FF2 ASCB 00FDCD60  JOBN VTAM
             LINE       DNODE PEP736A        EP 00        TIME 11
                                LCD 9  PCF 9  TIME 10  SCF 40  PDF 40      LCD 9  PCF 9  TIME 10  SCF 40  PDF 40
                                LCD 9  PCF 9  TIME 10  SCF 40  PDF 40
            TIME    42086.542112
89 USRFD FF2 ASCB 00FDCD60  JOBN VTAM
             LINE       DNODE PEP736A        EP 00        TIME 13
                                LCD 9  PCF 9  TIME 11  SCF 40  PDF 40      LCD 9  PCF 9  TIME 11  SCF 40  PDF 40
                                LCD 9  PCF 9  TIME 11  SCF 40  PDF 40      LCD 9  PCF 9  TIME 11  SCF 40  PDF 40
                                LCD 9  PCF 9  TIME 11  SCF 40  PDF 40      LCD 9  PCF 9  TIME 11  SCF 40  PDF 40
                                LCD 9  PCF 9  TIME 11  SCF 40  PDF 40      LCD 9  PCF 9  TIME 11  SCF 40  PDF 40
                                LCD 9  PCF 9  TIME 11  SCF 40  PDF 40      LCD 9  PCF 9  TIME 11  SCF 40  PDF 40
                                LCD 9  PCF 9  TIME 11  SCF 40  PDF 40      LCD 9  PCF 9  TIME 11  SCF 40  PDF 40
                                LCD 9  PCF 9  TIME 11  SCF 40  PDF 40      LCD 9  PCF 9  TIME 11  SCF 40  PDF 40
```

Figure 4-3. VTAM and GTF Traces (Part 9 of 13)

```
                                                          EXTERNAL TRACE - DD TAPE
                                LCD 9   PCF 9   TIME 11   SCF 40   PDF 40       LCD 9   PCF 9   TIME 11   SCF 40   PDF 40
                                LCD 9   PCF 9   TIME 11   SCF 40   PDF 40       LCD 9   PCF 9   TIME 11   SCF 40   PDF 40
                                LCD 9   PCF 9   TIME 11   SCF 40   PDF 40
           TIME     42087.037982
90 USRFD FF2 ASCB 00FDCD60  JOBN VTAM
            LINE       DNODE PEP736A        EP 00        TIME 13
                                LCD 9   PCF 9   TIME 11   SCF 40   PDF 40       LCD 9   PCF 9   TIME 11   SCF 40   PDF 40
                                LCD 9   PCF 9   TIME 11   SCF 40   PDF 40       LCD 9   PCF 9   TIME 11   SCF 40   PDF 40
                                LCD 9   PCF 9   TIME 11   SCF 40   PDF 40       LCD 9   PCF 9   TIME 11   SCF 40   PDF 40
                                LCD 9   PCF 9   TIME 11   SCF 40   PDF 40       LCD 9   PCF 9   TIME 11   SCF 40   PDF 40
                                LCD 9   PCF 9   TIME 11   SCF 40   PDF 40       LCD 9   PCF 9   TIME 11   SCF 40   PDF 40
                                LCD 9   PCF 9   TIME 11   SCF 40   PDF 40       LCD 9   PCF 9   TIME 12   SCF 40   PDF 40
                                LCD 9   PCF 9   TIME 12   SCF 40   PDF 40       LCD 9   PCF 9   TIME 12   SCF 40   PDF 40
                                LCD 9   PCF 9   TIME 12   SCF 40   PDF 40       LCD 9   PCF 9   TIME 12   SCF 40   PDF 40
                                LCD 9   PCF 9   TIME 12   SCF 40   PDF 40       LCD 9   PCF 9   TIME 12   SCF 40   PDF 40
                                LCD 9   PCF 9   TIME 12   SCF 40   PDF 40
           TIME     42087.038683
91 USRFD FF2 ASCB 00FDCD60  JOBN VTAM
            LINE       DNODE PEP736A        EP 00        TIME 13
                                LCD 9   PCF 9   TIME 12   SCF 40   PDF 40       LCD 9   PCF 9   TIME 12   SCF 40   PDF 40
                                LCD 9   PCF 9   TIME 12   SCF 40   PDF 40       LCD 9   PCF 9   TIME 12   SCF 40   PDF 40
                                LCD 9   PCF 9   TIME 12   SCF 40   PDF 40       LCD 9   PCF 9   TIME 12   SCF 40   PDF 40
                                LCD 9   PCF 9   TIME 12   SCF 40   PDF 9F       LCD 9   PCF 9   TIME 12   SCF 40   PDF 02
                                LCD 9   PCF 9   TIME 12   SCF 45   PDF 7E       LCD 9   PCF 9   TIME 12   SCF 45   PDF 7E
                                LCD 9   PCF 9   TIME 12   SCF 45   PDF 7E       LCD 9   PCF 9   TIME 12   SCF 45   PDF 7E
                                LCD 9   PCF 9   TIME 12   SCF 40   PDF C1       LCD 9   PCF 9   TIME 12   SCF 40   PDF 11
                                LCD 9   PCF 9   TIME 12   SCF 40   PDF 3D       LCD 9   PCF 9   TIME 12   SCF 40   PDF DD
                                LCD 9   PCF 9   TIME 12   SCF 45   PDF 7E       LCD 9   PCF 9   TIME 12   SCF 45   PDF FF
                                LCD 9   PCF 5   TIME 12   SCF 45   PDF FF
           TIME     42087.039365
92 USRFD FF2 ASCB 00FDCD60  JOBN VTAM
            LINE       DNODE PEP736A        EP 00        TIME 16
                                LCD 9   PCF 6   TIME 14   SCF 0D   PDF FF       LCD 9   PCF 7   TIME 14   SCF 49   PDF C1
                                LCD 9   PCF 7   TIME 14   SCF 49   PDF 91       LCD 9   PCF 7   TIME 14   SCF 49   PDF 35
                                LCD 9   PCF 7   TIME 14   SCF 49   PDF 59       LCD 9   PCF 6   TIME 14   SCF 0D   PDF 59
                                LCD 9   PCF 9   TIME 14   SCF 45   PDF 00       LCD 9   PCF 9   TIME 14   SCF 45   PDF 7E
                                LCD 9   PCF 9   TIME 14   SCF 45   PDF 7E       LCD 9   PCF 9   TIME 14   SCF 45   PDF 7E
                                LCD 9   PCF 9   TIME 14   SCF 40   PDF C1       LCD 9   PCF 9   TIME 14   SCF 40   PDF 11
                                LCD 9   PCF 9   TIME 14   SCF 40   PDF 3D       LCD 9   PCF 9   TIME 14   SCF 40   PDF DD
                                LCD 9   PCF 9   TIME 14   SCF 45   PDF 7E       LCD 9   PCF 9   TIME 14   SCF 45   PDF FF
                                LCD 9   PCF 5   TIME 14   SCF 4D   PDF FF       LCD 9   PCF 6   TIME 14   SCF 0D   PDF FF
                                LCD 9   PCF 7   TIME 14   SCF 49   PDF C1
           TIME     42087.041704
93 USRFD FF2 ASCB 00FDCD60  JOBN VTAM
            LINE       DNODE PEP736A        EP 00        TIME 16
                                LCD 9   PCF 7   TIME 14   SCF 49   PDF 91       LCD 9   PCF 7   TIME 14   SCF 49   PDF 35
                                LCD 9   PCF 7   TIME 14   SCF 49   PDF 59       LCD 9   PCF 6   TIME 14   SCF 0D   PDF 59
                                LCD 9   PCF 9   TIME 14   SCF 45   PDF 00       LCD 9   PCF 9   TIME 15   SCF 45   PDF 7E
                                LCD 9   PCF 9   TIME 15   SCF 45   PDF 7E       LCD 9   PCF 9   TIME 15   SCF 45   PDF 7E
                                LCD 9   PCF 9   TIME 15   SCF 40   PDF C1       LCD 9   PCF 9   TIME 15   SCF 40   PDF 11
                                LCD 9   PCF 9   TIME 15   SCF 40   PDF 3D       LCD 9   PCF 9   TIME 15   SCF 40   PDF DD
```

Figure 4-3. VTAM and GTF Traces (Part 10 of 13)

```
                                                    EXTERNAL TRACE - DD TAPE

                          LCD 9  PCF 9  TIME 15  SCF 45  PDF 7E      LCD 9  PCF 9  TIME 15  SCF 45  PDF FF
                          LCD 9  PCF 5  TIME 15  SCF 4D  PDF FF      LCD 9  PCF 6  TIME 15  SCF 0D  PDF FF
                          LCD 9  PCF 7  TIME 15  SCF 49  PDF C1      LCD 9  PCF 7  TIME 15  SCF 49  PDF 91
                          LCD 9  PCF 7  TIME 15  SCF 49  PDF 35
          TIME     42087.042370

 94 RNIO ASCB 00FE4390  CPU 0001 JOBN TRAC3600      IN 1C001001 211D0017 00180390 00C9D5D8 E4C9D9E8
          TIME     42224.350661
 95 USRFD FEF ASCB 00FE4390  JOBN TRAC3600
           TPIOS IN   ANODE P1VT3600   FDB  00000000 00B67111 00150000   RSVD 0812   LNG2 00C0
             REMOTE   DNODE U1C1L0E7   FSB  022C0000 00000000 1001211D 00170000 00000000 00000000 00000000 00180000
                                       THRH 1C001001 211D0017 00180390 00
                                       TEXT C9D5D8E4 C9D9E840 F3F6F0F0 40F0F2F2 40E4F1C3  *INQUIRY 3600 022 U1C*
                                            F1                                            *1                    *
          TIME     42224.353665
 96 USRFD FF1 ASCB 00FE4390  JOBN TRAC3600
           C/L  IN    ANODE P1VT3600   TEXT C9D5D8E4 C9D9E840 F3F6F0F0 40F0F2F2 40E4F1C3  *INQUIRY 3600 022 U1C*
                      DNODE U1C1L0E7        F1                                            *1                    *
          TIME     42224.357028
 97 USRFD FF1 ASCB 00FE4390  JOBN TRAC3600
           C/L  OUT   ANODE P1VT3600   TEXT C4C1E3C1 40C2C1E2 C540C1D7 D7D3C9C3 C1E3C9D6  *DATA BASE APPLICATIO*
                      DNODE U1C1L0E7        D540D9C5 E2D7D6D5 E2C540D4 C5E2E2C1 C7C54040  *N RESPONSE MESSAGE   *
                                            40404040 40404040 40404040 40404040 40404040  *                     *
                                            40404040 40404040 40404040 40404040 40404040  *                     *
                                            40404040 40404040                             *                     *
          TIME     42224.365078
 98 USRFD FEF ASCB 00FE4390  JOBN TRAC3600
           TPIOS OUT ANODE P1VT3600   FDB  00000000 00B670F0 00650000   RSVD 0000   LNG2 00A4   RSVD  00000000 00000000
             REMOTE   DNODE U1C1L0E7   THRH 1C00211D 10010000 00000390 00
                                       TEXT C4C1E3C1 40C2C1E2 C540C1D7 D7D3C9C3 C1E3C9D6  *DATA BASE APPLICATIO*
                                            D540D9C5 E2D7D6D5 E2C540D4 C5E2E2C1 C7C54040  *N RESPONSE MESSAGE   *
                                            40404040 40404040 40404040 40404040 40404040  *                     *
                                            40404040 40404040 40404040 40404040 40404040  *                     *
                                            40404040 40404040                             *                     *
          TIME     42224.368637
 99 RNIO ASCB 00FE4390  CPU 0001 JOBN TRAC3600      OUT 1E00211D 10010018 005B0390 00C4C1E3 C140C2C1
          TIME     42224.375301
100 RNIO ASCB 00FE4390  CPU 0001 JOBN TRAC3600      IN 1C001001 211D0018 00180390 00C9D5D8 E4C9D9E8
          TIME     42237.552094
101 USRFD FEF ASCB 00FE4390  JOBN TRAC3600
           TPIOS OUT ANODE P1VT3600   FDB  00000000 00B67280 000E0000   RSVD 0000   LNG2 00A4   RSVD  00000000 00000000
             REMOTE   DNODE U1C1L0E7   THRH 1C00211D 10010000 00006B80 00
                                       TEXT A1                                            *.                    *
          TIME     42238.034392
102 RNIO ASCB 00FE4390  CPU 0001 JOBN TRAC3600      IN 1F001001 211D0003 0004EB80 00A1
          TIME     42238.652850
103 USRFD FEF ASCB 00FE4390  JOBN TRAC3600
           TPIOS IN   ANODE P1VT3600   FDB  00000000 00B67111 00010000   RSVD 0812   LNG2 00C0
             REMOTE   DNODE U1C1L0E7   FSB  022C0000 00000000 1001211D 0003C000 000C0000 00000000 00000000 00040000
                                       THRH 1F001001 211D0003 0004EB80 00
                                       TEXT A1                                            *.                    *
          TIME     42238.691521
```

Figure 4-3. VTAM and GTF Traces (Part 11 of 13)

```
                                                    EXTERNAL TRACE - DD TAPE
104 USRFD FEF ASCB 00FE4390  JOBN TRAC3600
              TPIOS OUT ANODE P1VT3600    FDB  00000000 00B67028 000F0000   RSVD 0000   LNG2 00A4   RSVD  00000000 00000000
                REMOTE  DNODE U1C1L0E7    THRH 1C00211D 10010000 00006B80 00
                                          TEXT 3201                                    *..                  *
              TIME    42238.717074
105 RNIO ASCB 00FE4390  CPU 0001 JOBN TRAC3600       IN 1F001001 211D0001 0004EB80 0032
              TIME    42239.251781
106 USRFD FEF ASCB 00FE4390  JOBN TRAC3600
              TPIOS IN  ANODE P1VT3600    FDB  00000000 00B675C1 00010000   RSVD 0812   LNG2 00C0
                REMOTE  DNODE U1C1L0E7    FSB  022C0000 00000000 1001211D 00010000 00000000 00000000 00000000 00040000
                                          THRH 1F001001 211D0001 0004EB80 00
                                          TEXT 32                                      *.                   *
              TIME    42239.254610
107 USRFD FEF ASCB 00FDCD60  JOBN VTAM
              TPIOS OUT ANODE VTAM        FDB  00000000 00B675A0 000F0000   RSVD 0000   LNG2 00A4   RSVD  00000000 00000000
                REMOTE  DNODE U1C1L0E7    THRH 1C00211D 10000000 00006B80 00
                                          TEXT 0E                                      *.                   *
              TIME    42253.494622
108 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM         OUT 1F00211D 10000002 00046B80 000E
              TIME    42253.499154
109 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM          IN 1F001000 211D0002 0004EB80 000E
              TIME    42254.054066
110 USRFD FEF ASCB 00FDCD60  JOBN VTAM
              TPIOS IN  ANODE VTAM        FDB  00000000 00B672A1 00010000   RSVD 0812   LNG2 00C0
                REMOTE  DNODE U1C1L0E7    FSB  022C0000 00000000 1000211D 00020000 00000000 00000000 00000000 00040000
                                          THRH 1F001000 211D0002 0004EB80 00
                                          TEXT 0E                                      *.                   *
              TIME    42254.121367


   *** DATE   DAY 168    YEAR 1975            TIME 16.44.23.800573                                                ***

111 USRFD FEF ASCB 00FDCD60  JOBN VTAM
              TPIOS OUT ANODE VTAM        FDB  00000000 00B67280 000F0000   RSVD 0000   LNG2 00A4   RSVD  00000000 00000000
                REMOTE  DNODE CL1L0E7     THRH 1C00211C 10000000 00006B80 00
                                          TEXT 1202                                    *..                  *
              TIME    42263.944876
112 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM         OUT 1F00211C 10000002 00056B80 001202
              TIME    42263.952790
113 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM          IN 1F001000 211C0002 0004EB80 0012
              TIME    42264.555801
114 USRFD FEF ASCB 00FDCD60  JOBN VTAM
              TPIOS IN  ANODE VTAM        FDB  00000000 00B675C1 00010000   RSVD 0812   LNG2 00C0
                REMOTE  DNODE CL1L0E7     FSB  022C0000 00000000 1000211C 00020000 00000000 00000000 00000000 00040000
                                          THRH 1F001000 211C0002 0004EB80 00
                                          TEXT 12                                      *.                   *
              TIME    42264.559330
115 USRFD FEF ASCB 00FDCD60  JOBN VTAM
              TPIOS OUT ANODE VTAM        FDB  00000000 00B675A0 00120000   RSVD 0000   LNG2 00A4   RSVD  00000000 00000000
                REMOTE  DNODE PEP736A     THRH 1C002000 10000000 00000B80 00
                                          TEXT 01020221 1C                             *.....               *
              TIME    42264.582179
116 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM         OUT 1F002000 10000034 00080B80 00010202 211C
              TIME    42264.586587
```

Figure 4-3. VTAM and GTF Traces (Part 12 of 13)

```
                                                EXTERNAL TRACE - DD TAPE

117 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM          IN 1E001000 20000034 00069B80 00010202
             TIME    42264.955744
118 USRFD FEF ASCB 00FDCD60  JOBN VTAM
              TPIOS IN  ANODE VTAM       FDB  00000000 00B672A1 00030000   RSVD 0812   LNG2 00C0
                REMOTE  DNODE PEP736A    FSB  022C0000 00000000 16002000 00340000 00000000 00000000 00000000 00060000
                                         THRH 1E001000 20000034 00069B80 00
                                         TEXT 010202                                  *...                 *
             TIME    42264.959263
119 USRFD FEF ASCB 00FDCD60  JOBN VTAM
              TPIOS OUT ANODE VTAM       FDB  00000000 00E67280 00120000   RSVD 0000   LNG2 00A4   RSVD  00000000 00000000
                REMOTE  DNODE PEP736A    THRH 1C002000 10000000 00000B80 00
                                         TEXT 01020B21 1B                             *.....               *
             TIME    42265.197036
120 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM         OUT 1E002000 10000035 0008CB80 0001020B 211E
             TIME    42265.202027
121 RNIO ASCB 00FDCD60  CPU 0001 JOBN VTAM          IN 1E001000 20000035 00069B80 0001020B
             TIME    42270.355804
122 USRFD FEF ASCB 00FDCD60  JOBN VTAM
              TPIOS IN  ANODE VTAM       FDB  00000000 00B675C1 00030000   RSVD 0812   LNG2 00C0
                REMOTE  DNODE PEP736A    FSB  022C0000 00000000 16002000 00350000 00000000 00000000 00000000 00060000
                                         THRH 1E001000 20000035 00069B80 00
                                         TEXT 010206                                  *...                 *
             TIME    42270.359397
```

Figure 4-3. VTAM and GTF Traces (Part 13 of 13)

Notes on Examples 1 and 2

1. Mappings of the data in the various trace entries are not included. Blocks such as FDB and FSB are described in VTAM manuals, *OS/VS2 VTAM Logic* and *OS/VS2 VTAM Data Areas.*

   PIU formats can be found in the *3704/3705 Program Reference Handbook.* Those PIUs that accomplish a function other than transfer of data between an application and a terminal have a network command in the RU (as shown in entry 3 of the examples). Most are detailed in the "Network Commands" section of the *3704/3705 Program Reference Handbook.* Network commands that the NCP must process are shown in more detail in the "Network Commands" appendix in *IBM 3704 and 3705 Communications Controller Network Control Program/VS Logic.*

   For a full understanding of the line trace entries, refer to *3704/3705 Communications Controller Principles of Operation* for ICW field definitions. SDLC commands and N(R) – N(S) processing can be seen in the trace examples (at entries 83-93). The *IBM 3704 and 3705 Program Reference Handbook* section "SDLC Commands and Responses", and *IBM Synchronous Data Link Control General Information* may be useful.

2. Data flow between the application and the LU is on an exception-response-only basis. Other PIUs request positive FME acknowledgement, and the FMEs can be seen in the trace examples (at entry 8).

3. No pacing is used; it was not included in the NCP definition.

4. Note that the line trace shows the outbound PIU (see entry 82) changed from the FID1 TH (transmission header) format that was transferred to the NCP, into a FID2 format TH that is sent to a cluster controller (see entry 83). Also note that the NCB segmented the PIU during transmission to the controller. The length of the PIU was greater than the MAXDATA operand for the controller in the NCP generation (66 in this NCP), so it was broken into two segments. The second segment is sent (starting in entry 87).

5. At the beginning of each PIU transmission, there are three or four flags set (X'7E') because a temporary "superzap" was made in the NCP at the time the trace was made. Normally only one flag would be sent.

Summary

When symptoms are intermittent or confusing, the debugger should be aware that there is some variable present that he has not recognized. In such situations, assumptions are dangerous, yet it is very easy and common to focus immediately on the wrong process as the location of a problem and assume that other steps must be correct. Using the traces in the TP components should help the debugger to make fewer assumptions.

A less obvious benefit of these traces is their educational value. NCP line traces, for example, illustrate from actual examples the workings of TP line protocols and their use with different devices. These protocols are important when "what if" situations need to be projected and when line errors or terminal errors can be translated into some of the none-too-obvious external symptoms that sometimes result. Then, symptoms may be seen later in terms of possible component errors and traces or traps can be used to confirm suspicions.

As discussed earlier, an operator command turns on (or off) a trace for one node. Sometimes, many or all terminals of a certain class or on a set of lines need to be traced. Depending on when an error is occurring, or on the connection design of a network, tracing sometimes should start as soon as the NCP is activated. A trace can be started when the NCP is activated (see *Operator's Library: VTAM Network Operating Procedures* or, *Operator's Library: OS/VS TCAM Level 10*), buf if several must be started, the following technique has proven useful.

In the NCP definition, code INITEST=YES on the PCCU macro, but do not put an INITEST DD card in the VTAM procedure. Upon activation of the NCP, VTAM asks the operator if he wants to bypass initial test. At this point the network is defined inside VTAM so traces can be started by operator command, but the NCP has not yet been loaded. Start as many traces as desired and reply to bypass initial test. After your reply, the NCP will activate. This technique was used to trace the initialization sequences shown in the first trace example.

VTAM Buffer Trace Modification

Many operator commands are required if all nodes are to be traced. However, VTAM modules can be superzapped to trace unconditionally. The examples that follow are intended only to illustrate a technique for gathering information. They may be applied differently to suit individual situations.

Various techniques can be used for a VTAM buffer trace. Buffer trace for a node is indicated by bits in the RDTE (resource definition table entry) and in the FMCB (function management control block) that describe a session. Module ISTOCCFB creates the FMCB and, by temporarily modifying it to create all FMCBs with trace bits on, causes buffer trace to always be done.

NAME	ISTOCCRT	ISTOCCFB	
ver	08BA	9110A015	test RDTE trace bit
ver	08BE	47E0B8AE	
ver	08C2	9604E020	turn on in FMCB
rep	08BE	4700	cause fall-thru

Alternately, each traced path can be "superzapped" in the logic that checks a trace bit. For buffer trace, four "zaps" are required: for C/L and TPIOS buffer trace, and each IN and OUT. By zapping at these locations, selectivity is possible, and alternate control can be introduced by changing the logic from checking the trace bit in FMCB to checking some other global indicator, such as a flag in the PSA. This technique is illustrated in the following discussion of I/O trace.

VTAM I/O Trace (RNIO) Modification

The same alternatives apply to the I/O trace that applied to the VTAM buffer trace. To create every NCB (node control block) with the RNIO trace indicator on (so that everything is traced), the ZAP is:

NAME	ISTOCCRT	ISTOCCFB	
ver	06BA	91087015	check flag in RDTE
ver	06BE	47E0B6AE	
ver	06C2	9201A01D	turn on flag in NCB
rep	06BE	4700	cause fall-thru

Using the second approach of altering the checking at the time of the tracing, two zaps are required. The following zap will cause tracing of all inbound PIUs if a byte in the PSA(s) (location X'xxx') is set non-zero:

NAME		ISTZCEAB	ISTZFM1B
ver	68	9500301D	check NCB flag
rep	68	95000xxx	check low-core flag instead
ver	7A	9500401D	check NCB flag
rep	7A	95000xxx	check low-core

Two paths exist within ISTZFM1B, so two locations must be changed.

For outbound PIUs, the logic in VTAM is slightly more complex. An indication of whether or not to perform RNIO trace is transferred from the NCB to the PIUs buffer area before the buffer is queued for output. However, the trace is not performed until the I/O is complete, so although the indicator is used at that time, VTAM *also* checks to see if the GTF RNIO trace is still active. Therefore, in addition to the following "zap," the GTF RNIO trace must be active before VTAM attempts to make the trace entry:

NAME		ISTZCEAB	ISTZFFDB	
ver	D2	91406011	47E0yyyy	check trace flag
rep	D2	95000xxx	4780yyyy	check low-core instead

Other Tracing Methods

If there is a single point to be made in this section, it is to minimize assumptions about what or how data is travelling through a network. There are other sources of information besides standard traces that can provide snapshots of messages at various points enroute. An application program may log every message received and sent. It may be important to know exactly at what point in the flow the logging occurs, but in general the log can be used as another trace point when access method traces have been followed as far toward the application as they go.

Access method or component buffers may sometimes be used to see if a message got as far as a buffer, or to see what form a message had when it was put into a buffer by a component. Dumps of buffer areas and dumps of TCAM queues, from disk or main storage, would be used in place of traces. The limitation here is buffer or queue reuse, which often creates confusion when half of the message to be examined is found but critical information has been lost because of reuse. Nevertheless, these sources can be valuable. In the NCP, for example, because only one line can be traced, buffers usually provide the last snapshot of a message as it appeared before going to a terminal. Status indicators in buffer headers often can be used to tell how a message was processed; if the buffer is still in use, then backtracking to find a work element or process that refers to the buffer can provide the key to understanding why a message is stuck. The following example shows such a case.

TP Problems (continued)

Assume a heavy-running 3600 network in which a few logical units do not receive a response message after input is entered. The problem is intermittent and strikes any LU any time over a period of several hours. The GTF VTAM traces are run for all LUs during a typical run and when one LU fails to receive a response message, traces are stopped and all network components dumped.

It is not possible (without writing a user exit) to print the GTF trace for selected terminals, so the entire trace is printed for a short period surrounding the time of failure. Activity on the problem LU shows the last input message came into VTAM and the application and a response message was sent all the way out to the 3705. Line trace is not used because only one line can be traced and the problem line is not predictable.

Other variations are tried with the LU; it can be closed (via CLSDST) and reconnected and run, so a hardware problem is unlikely. No MDR records in SYS1.LOGREC indicate an error on the line. At this point the problem is isolated to the NCP or beyond in the network. Using other indicators (in this case the NCP's logical unit block, LUB), an analysis of the message path from the time the NCP receives it shows that the last outbound PIU did not go out over the TP line. (NCP keeps the sequence number of the last PIU sent and the number in the GTF trace is the next higher sequence number than the one in the LUB.) The NCP buffers are searched and the missing PIU is found intact in a buffer. The problem is isolated to the NCP; the buffer is still in use, and indicators in the buffer header reveal how much processing was done in the NCP; this leads eventually to the bug in the NCP.

4.3.32 OS/VS2 System Programming Library: MVS Diagnostic Techniques

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.

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 an address space, the nucleus, and request SQA. The PRDMP service aid (with the QCBTRACE 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 4-4.

$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 A command.

Figure 4-4. 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 4-5 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. The in-use blocks are anchored in the CVT at CVT+X'280' (CVTFQCB, first QCB element) and at CVT+X'284' (CVTLQCB, last QCB element). A queue of many QELs off a particular major or minor 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.

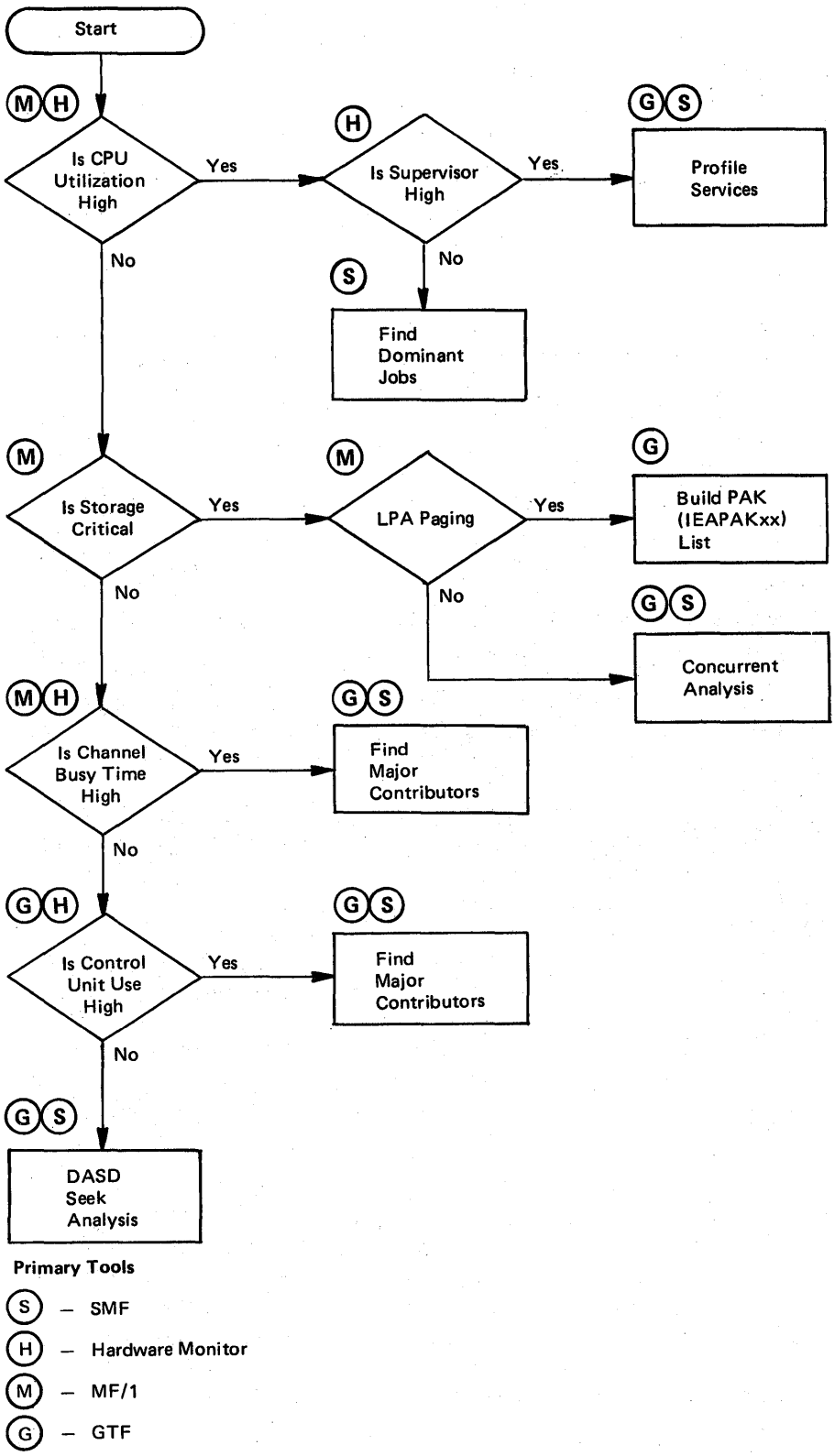


Figure 4-5. System Use of Hardware Components

Also check the free elements. The ENQ/DEQ global save area, mapped in IEAVENQ1, contains six addresses, each of which points to the first element of a free queue. The ENQ/DEQ global save area is found through: CVT+X'2AC' (CVTSPSA) which points to the global save area table; the global save area table +X'20', points to the ENQ/DEQ global save area. There are multiple queues, each containing blocks of all one size. These blocks occupy SQA. Merging the free elements with the in-use elements should provide an indication of ENQ SQA fragmentation. Because fixed storage is involved, the fragmentation may be reducing the number of frames available for paging.

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 IECVSHDR, 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.

Performance Degradation (continued)

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'1D0') and ASMIORQC (ASMVT+X'1D4') 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'74'), PAREFLGS (PART+X'8'), and the IOSB for paging requests (IOSCOD=X'41') 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. 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.

4.4.6 OS/VS2 System Programming Library: MVS Diagnostic Techniques

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 ?

Incorrect Output (continued)

- 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 15 system components and provides debugging techniques for determining the cause of an error that has been isolated to a component.

The components described in this section are contained in the following chapters:

- Dispatcher
- IOS
- Program Manager
- VSM
- RSM
- ASM
- SRM
- VTAM
- VSAM
- Catalog Management
- Allocation/Unallocation
- JES2
- SSI
- RTM
- Communications Task

SEC-
TION

5.1.2 OS/VS2 System Programming Library: MVS Diagnostic Techniques

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, no locks held.

This entry point is called by the following:

- Exit prologue (IEAVEEXP), when control is not returned to the issuer of an SVC.
- Lock manager (IEAVELK), when it is suspending a task that unconditionally requested a local lock that was unavailable.
- Program check FLIH (IEAVEPC), when a TCB or SRB was suspended because of a page fault that required I/O or because no frames were available.
- External FLIH.
- RTM (recovery termination manager).

IEAPDS7 – Entered disabled, key 0, supervisor state, no locks held.

This entry point is called by I/O FLIH and by SVC FLIH when the SVC requires a local lock that is not available.

IEAPDS6 – Entered disabled key 0, supervisor state, no locks held.

This entry point is called by RTM on an EOT (end of task) condition.

IEAPDS2 – Entered disabled, key 0, supervisor state, the dispatcher lock held.

This entry point is called by the lock manager (IEAVELK), when suspending an SRB that has requested the local or CMS lock, and when suspending an address space that has requested the CMS lock.

IEAPDSRT – Entered enabled or disabled, any key, supervisor state, no locks held.

This entry point is the termination return address for all SRBs.

DSJSTCSR – Job step timing subroutine. Calculates and accumulates job step timing.

This entry point is called by the following:

- Lock manager (IEAVELK), when common suspend routine of the lock manager is suspending an SRB or locally locked TCB because of a lock request or a page fault suspension.
- Dispatcher. The dispatcher calls this subroutine internally when it is saving the status of a previously executed unit of work.
- Timer SLIH. The timer SLIH calls this subroutine before it gives control to SRM.

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 with which they are dispatched.

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:

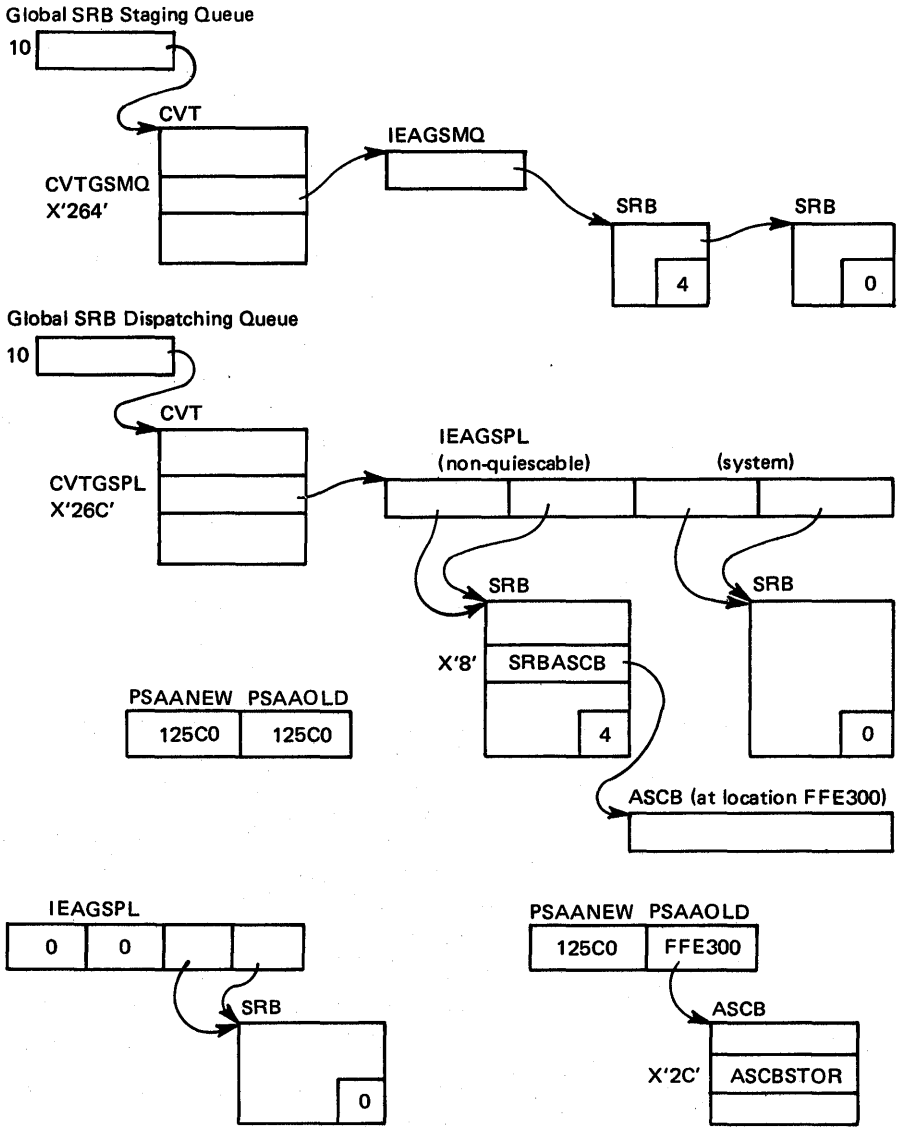
Bit	*Exit*
LCCAACR	ACR
LCCAVCPU	Vary CPU
LCCATIMR	Timer Recovery

   The dispatcher enters these exits via a branch.

2. Global SRBs

   IEAGSMQ is the header for the global SRB staging queue. If it is not zero, it points to the global SRB queue. (See Figure 5-1.) Requestors use the SCHEDULE macro to compare and swap global SRBs onto the queue. The dispatcher obtains the DISP lock and removes the SRBs from the queue with the compare and swap (CS) instruction. The dispatcher then calls CSECT IEAVESC0 at entry IEAVESC1 in order to move the SRBs to the appropriate priority level (0 or 4) on the GSPL (global system priority list) queue.

   IEAGSPL is the global SRB dispatching queue. The queue is divided into non-quiescable (priority level 4) and system level (priority level 0) SRBs. The dispatcher removes the SRB from the GSPL queue, updates the PSAAOLD with the SRBASCB address, loads its STOR value, and dispatches the SRB. PSAANEW is not updated.



NOTE: 0 and 4 in SRBs represent system priority level

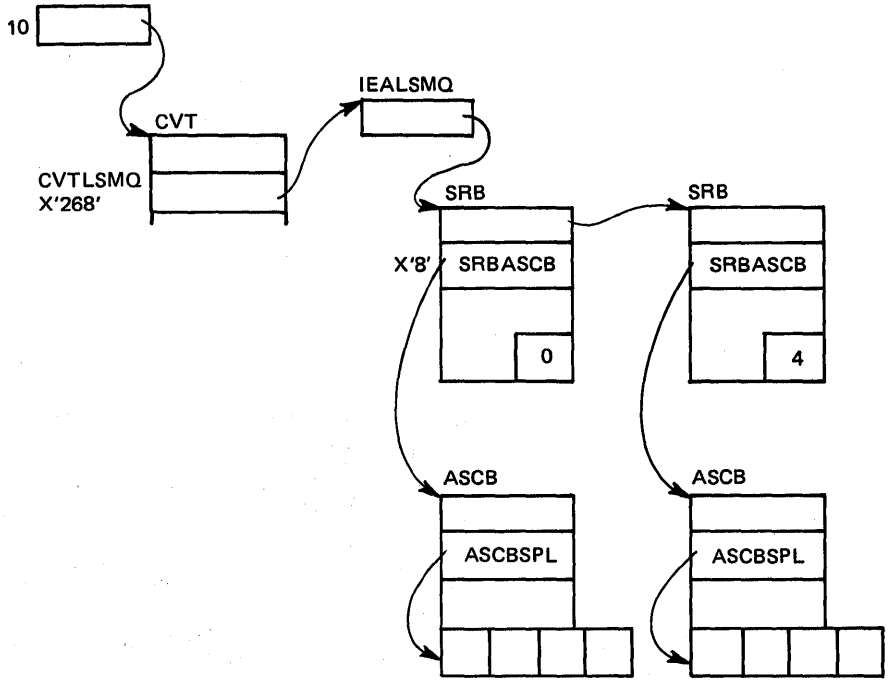
Figure 5-1. Global SRB Queue Structure and Control Block Relationships

3. Local SRBs

   IEALSMQ is the header for the local SRB staging queue. If it is not zero, it points to the local SRB queue. (See Figure 5-2.) Requestors use the SCHEDULE macro to compare and swap local SRBs onto the queue. The dispatcher tests this queue if it cannot find any special exits or global SRBs to dispatch. If this queue is *not* empty, the dispatcher obtains the DISP lock and removes the entire queue with compare and swap instructions. The dispatcher then calls CSECT IEAVESC0 at entry IEAVESC2 in order to move the SRBs to the appropriate priority level (0 or 4) on the LSPL. IEAVESC2 also notifies SRM via the SYSEVENT macro if the address space is swapped out. Memory switch is then invoked to direct the dispatcher to the highest priority work.

   (Note that no work is dispatched. The SRBs are simply moved to the appropriate dispatching queues (ASCBSPLs).)

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:

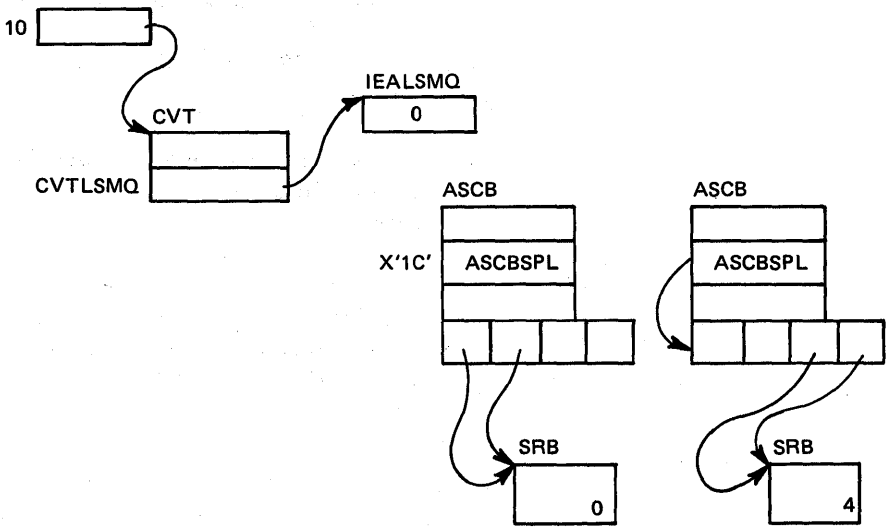


Figure 5-2. Local SRB Queue Structure and Control Block Relationships

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 (memory) 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 LSPL (pointed to by the ASCBSPL), the top SRB is dequeued and dispatched. If there are no SRBs on the local SPL queue, the local lock is tested for the 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 program is interrupted and control is switched away from it because there is higher priority work to handle.

   The dispatcher does a compare and swap to obtain the local lock:

   - If the local lock is available and the number of ready TCBs exceeds the number of processors active in the address space,

   - or if the ASCBS3S bit (ASCB + X'67') indicates that there is work for the Stage 3 Exit Effector to process.

   If the dispatcher is successful in obtaining the lock, it will go to the Stage 3 exit effector, if necessary, and then select the first dispatchable TCB that is not active on another processor.

   The dispatcher may dispatch the above units (SRBs, supervisor, TCBs) without going through the memory dispatcher if the address space was current when the dispatcher was entered and if there was no indication that a memory switch was required. (PSAANEW = PSAAOLD).

5. Wait Task

   The wait task 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.

   Figure 5-3 provides an overview of the processing sequence through the MVS dispatcher.

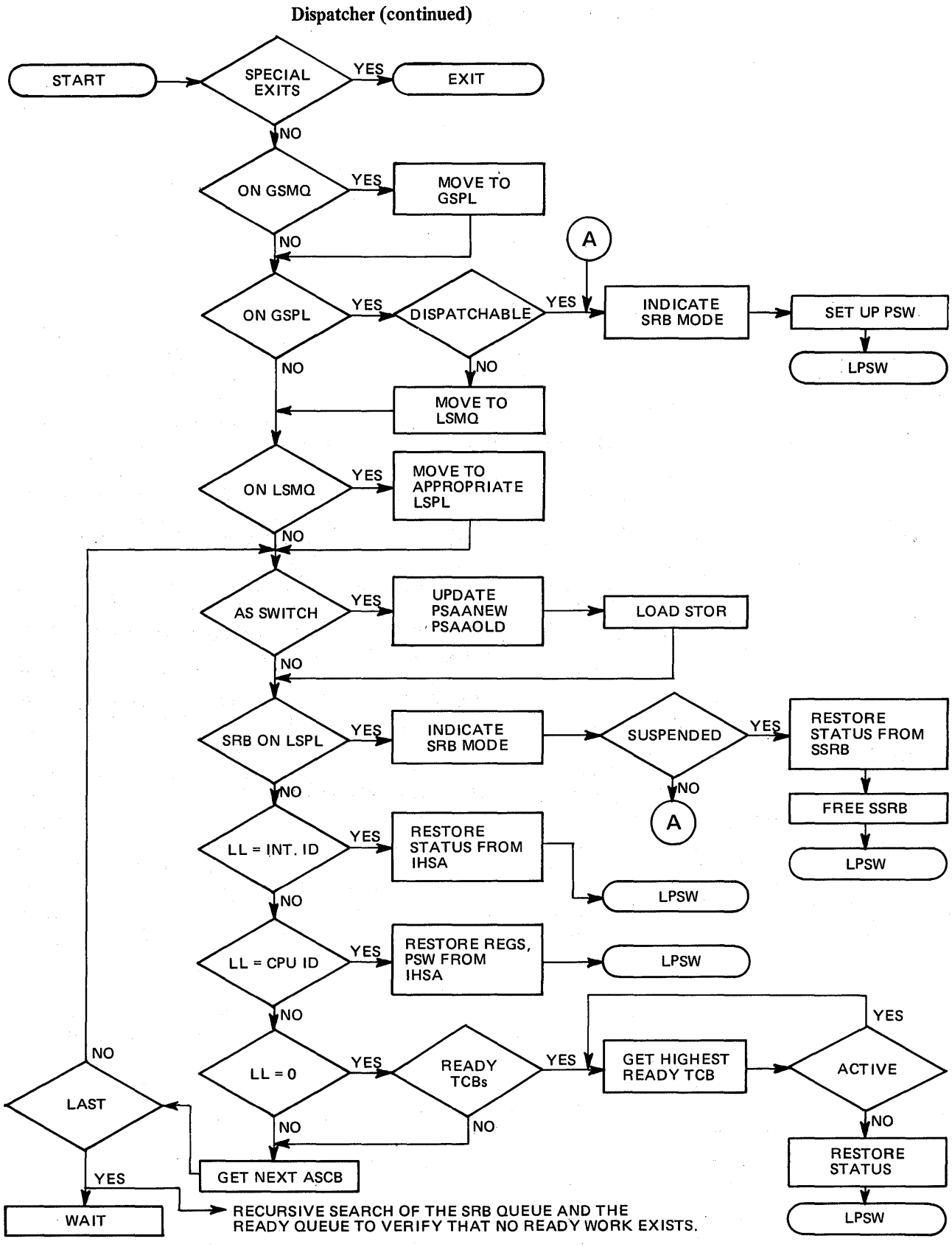


Figure 5-3. Dispatcher Processing Overview

Dispatchability Tests

The dispatcher conducts the following dispatchability checks:

SRB Tests

*Test **	*Condition*
1. ASCBRCTF//ASCBOUT	Address space swapped out.
2. CSDSCFL1//CSDSYSND	System non-dispatchable.
(a) ASCBFLG2//ASCBXMPT	(a) If the system is non-dispatchable, the SRB must have been scheduled to an exempt address space.
3. ASCBDSP1//(any bit on)	Address space non-dispatchable.
4. ASCBFLG2//ASCBSNQS	All SRBs stopped.
5. ASCBSSRB	System level SRBs stopped (This does not apply to NONQ SRBs.)
6. SRBCPAFF	Does SRB have affinity to this processor ? (PCCACAFM defines the current processor)
7. SSRBFLG1//SSRBLLH	If set, compare and swap (via CS instruction) CPUID into local lock word, ASCBLOCK, which has the suspend ID in it (X'7FFFFFFF').

*Format of test description is "field//bit within field."

Dispatcher (continued)

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//ASCBNOQ	The ASCB is not on the ready queue. The address space will not be dispatched.
2. ASCBDSP1//ASCBFAIL	The ASCB is in failure mode and in the process of being terminated. The address space will not be dispatched.
3. CSDSCFL1//CSDSYSND	System non-dispatchable.
(a) ASCBFLG2//ASCBXMPT	(a) If the system is non-dispatchable, the SRB must have been scheduled to an exempt address space.
4. *LOCAL LOCK//ASCBLOCK*	
Suspend ID (X'7FFFFFFF')	Cannot process the address space unless an SRB owned the local lock and was suspended and is now re-scheduled to be dispatched.
Interrupt ID (X'FFFFFFFF')	Compare and swap CPUID into local lock and restore the status (FPRs, GPRs, FRR stack, CPU timer value, PSATOLD PSATNEW, and resume PSW) from the IHSA.
Own CPUID	Restore GPRs (general purpose registers) and PSW from the IHSA.
Free (X'00000000')	If ready work is in the address space, compare and swap (via CS instruction) the CPUID into ASCBLOCK.
Other CPUID	Bypass the address space.
5. ASCBFLG1//ASCBS3S	Go to the task dispatcher and interface with Stage 3 exit effector.
6. ASCBTCBS > ASCBCPUS	There are more TCBs ready than there are processors currently executing in the address space; the address space can be dispatched.

After these six tests indicate that the dispatcher should dispatch in an address space, the following task indicators are tested.

Task Tests	*Condition*
1. RBWCF	RB must not be waiting.
2. TCBFLGS4	TCB primary non-dispatchability flags must not be set.
3. TCBFBYT1//TCBACTIV	If TCB is active, it must be a redispatch situation; otherwise, this TCB is active on the other processor (TCBCCPVI).
4. TCBAFFN	TCB affinity, if any, must match this processor's physical address (which is located in PCCACAFM).

Miscellaneous Notes about the Dispatcher

1. You can determine the last dispatch by examining the PSW at location X'300'. The TOD of the last dispatch is located at LCCADTOD (LCCA + X'258').
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

Variable Data:

SDWAURAL – Seven full words of data as follows:

PSAHLHI – Locks held at time of error.

ASCBLOCK – Value of local lockword for current address space at the time of error.

LCCASPLJ – SRB queue journal word. Contains the address of the top SRB on the staging queue when dequeued by the dispatcher and passed to IEAVESC0.

PSAAOLD – Current ASCB address at the time of error.

Control Register 1 – Value of STOR (CR1) at the time of error.

PSATOLD – Current TCB address at the time of error.

LCCADSF1 – Dispatcher flag bytes that were on at the time of error.

If the dispatcher lock was held at the time of error, the following recovery routines are called by the dispatcher recovery routine:

- IEAVESCR – Schedule 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 was held by the dispatcher, 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 SDWAURAL 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). 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

- The abend C0D is issued from CSECT IEAVESC0 when a local SRB is scheduled to an invalid ASCB.
- 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

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 IOS 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.

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 appendages). Figure 5-4 shows an overview of the IOS 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.

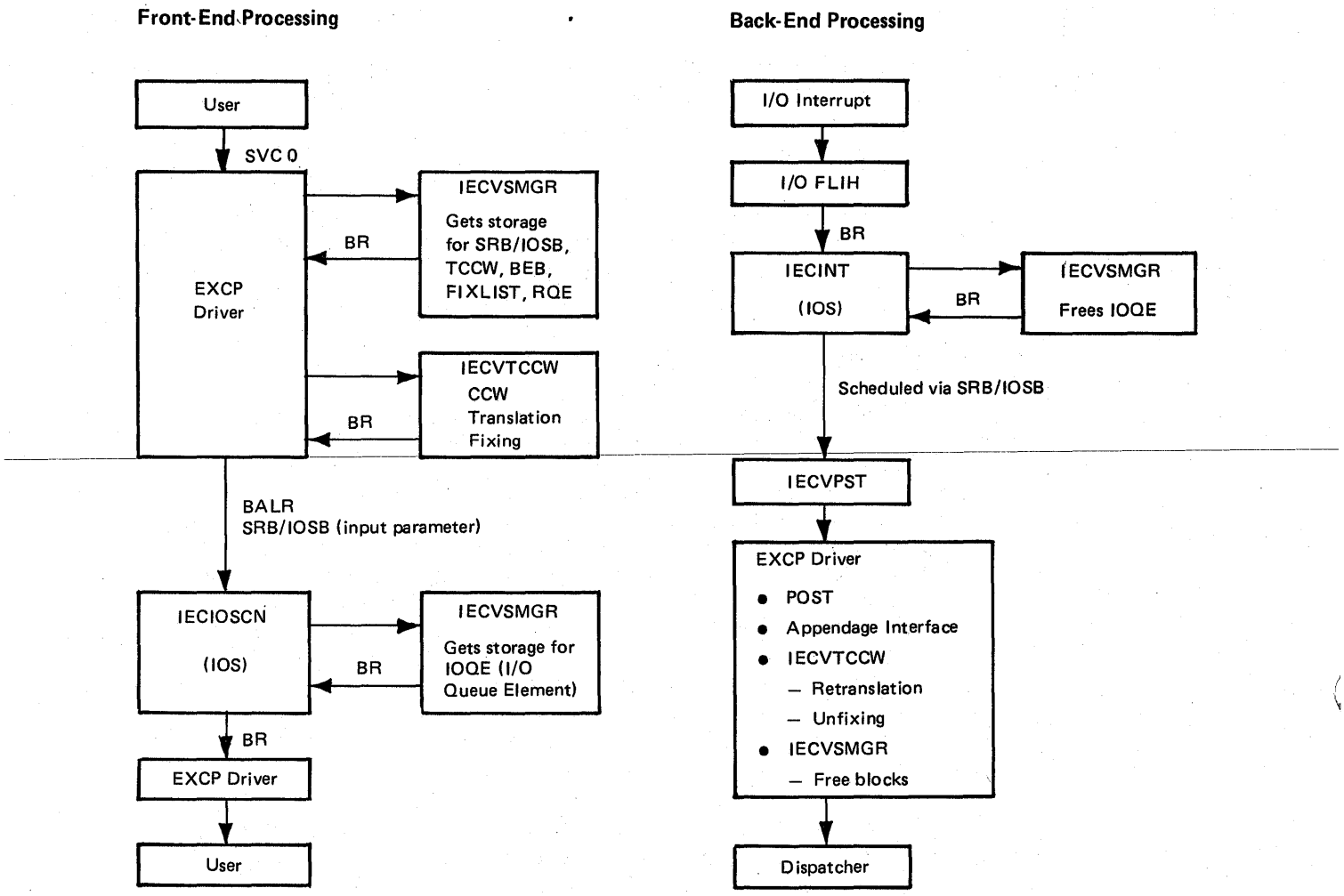


Figure 5-4. IOS Processing Overview

Major IOS Control Block Relationships

Major EXCP Control Block Relationships

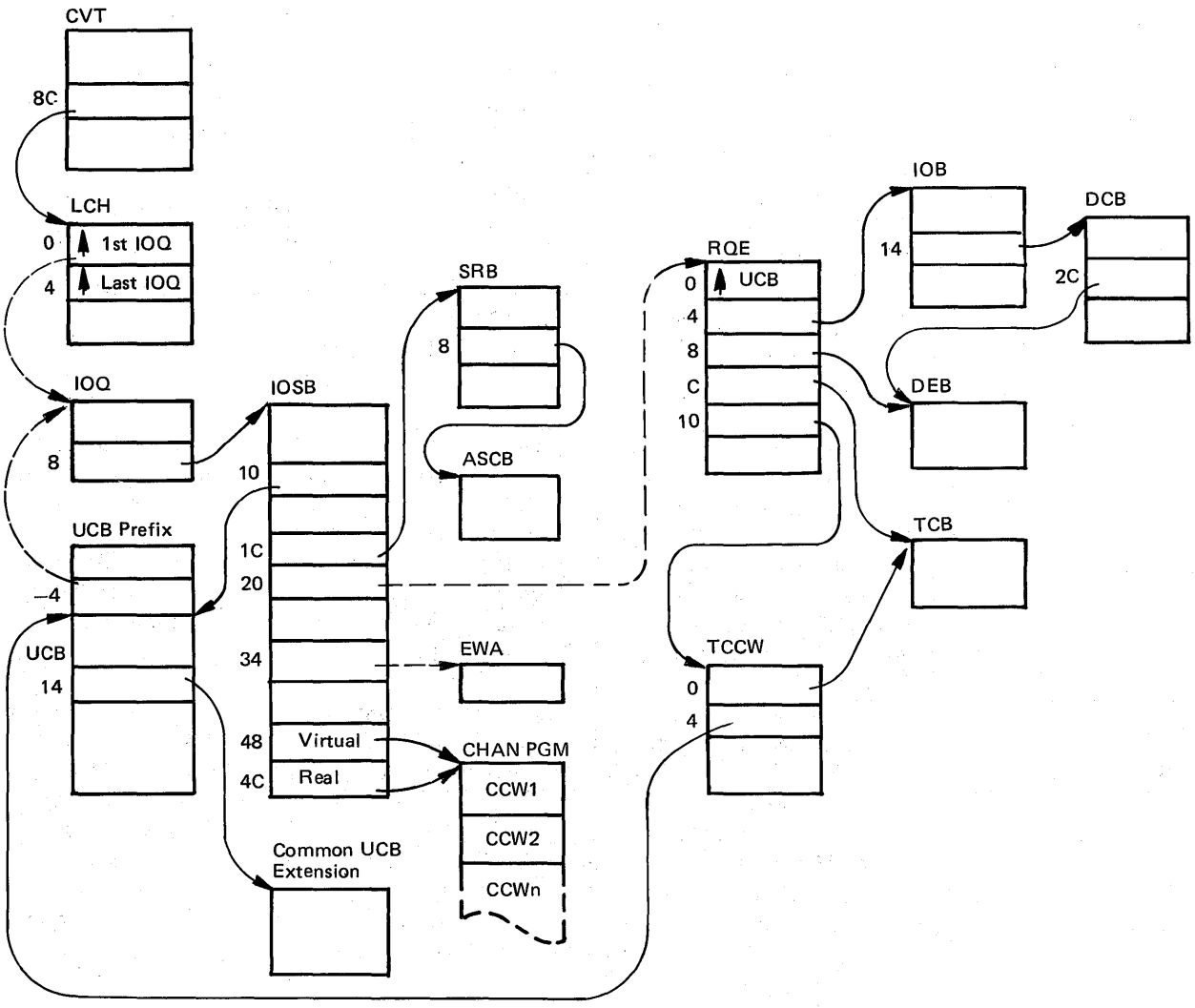


Figure 5-5. Major IOS and EXCP Control Block Relationships

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 *OS/VS Message Library: VS2 System Codes.* The EXCP problem determination area is documented in *OS/VS2 I/O Supervisor Logic.*

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 the storage manager (IECVSMGR) passing a bad address back to a requestor. Consequently, the requestor initializes the bad block and overlays or clobbers some valuable piece of storage. On occasion the bad address passed by IECVSMGR is 0. 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.

The main thing to keep in mind is that generally IECVSMGR is not the cause of the problem. For performance optimization reasons, the storage manager has minimal validity checking and thus trusts that the invoker is operating correctly. Historically the cause of this type of problem is that the same block is freed twice, which causes the storage manager's free queue to contain invalid pointers.

Often this 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. 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 IOSB pointed to by 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 a 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 compatability. 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 entered with IOSCOD=7F, IOSEX=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 buffers 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.

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, or channel data 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:
   - For DASD error recovery – issue a BALR to IECVDERP.
   - 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:
   - Retry/restart the channel program
   - Issue console message IEA000I or IEA000A
   - Log the error
   - Indicate that the error is permanent
   - 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 a 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 sub-routine 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		*Action*
IOSMSG	*IOSLOG*	
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'7E' 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.

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)

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 in 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.

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 functions.
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
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	CMS lock	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. Queue 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

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.

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.)

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 useable copy of the requested load module. An abend occurs if it cannot. If a useable copy is found, but not in virtual storage, module IEAVLK01 brings it in.

If a useable 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 reuseable module currently in use (indicated by the CDREN bit being off, the CDSER bit being on, and a non-zero CDRRBP field.

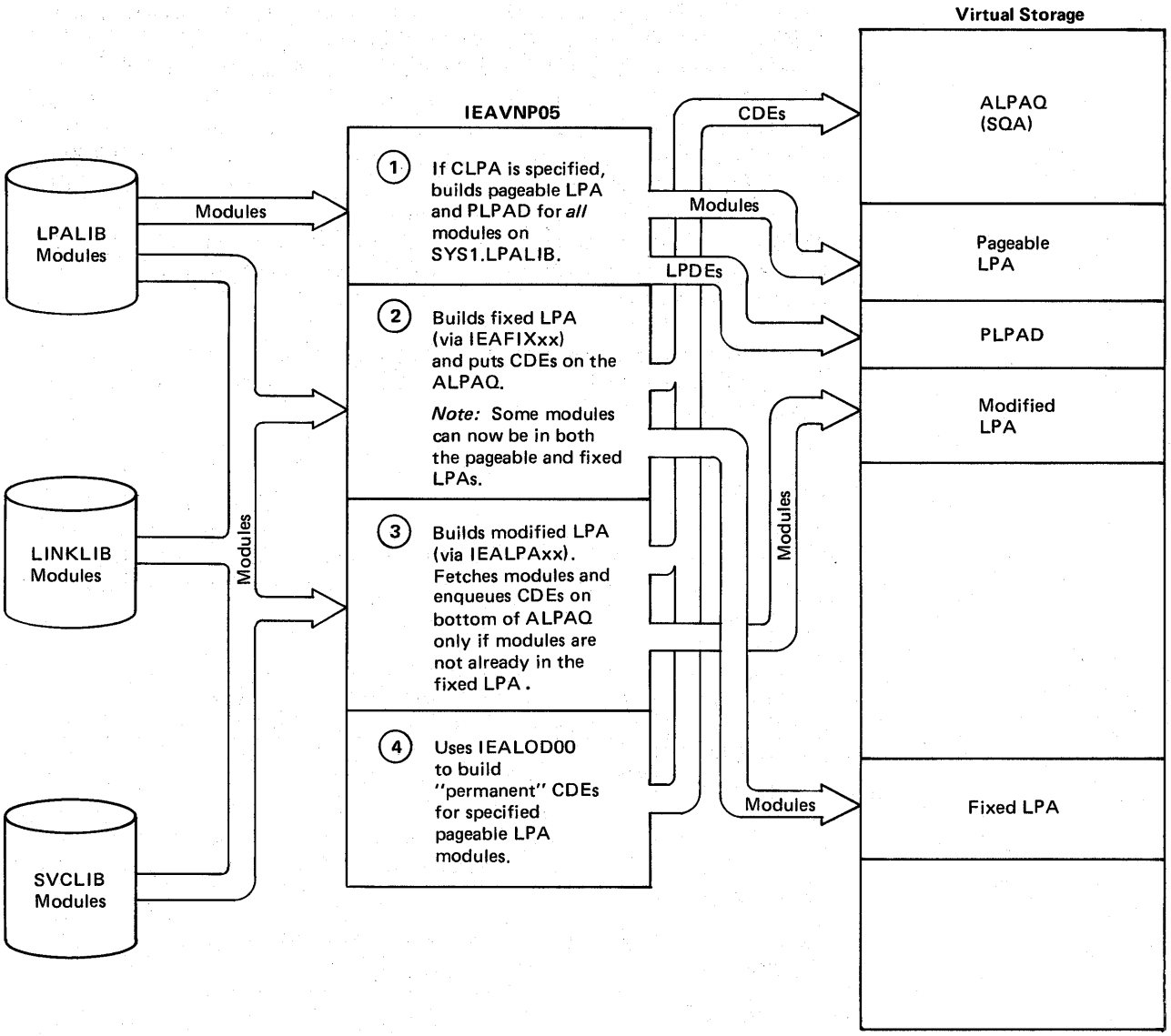


Figure 5-9. IEAVNP05 Initialization

If a useable 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 reuseable 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 useable, the use count is *not* increased.

- If an LPDE in the PLPAD is found useable, 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.

RB field	How initialized by IEAVLK00 for XCTL		
	Caller is a PRB	Caller is an 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

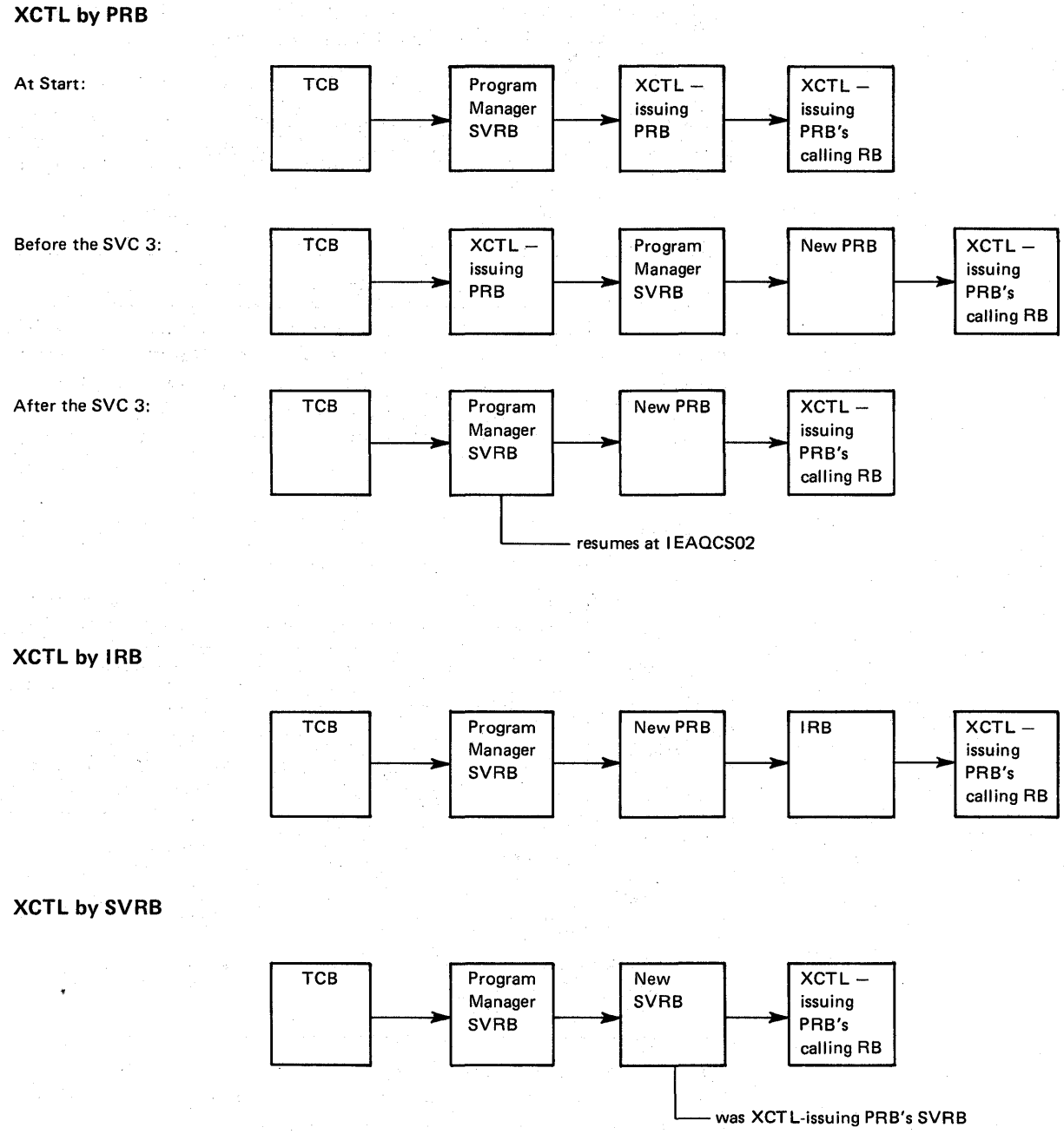


Figure 5-12. XCTL RB Manipulation

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 useable 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 useable 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 reuseable, 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 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 built 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.

CDE Field	Normal Request: Normal	Normal Request: 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 S VRB 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.

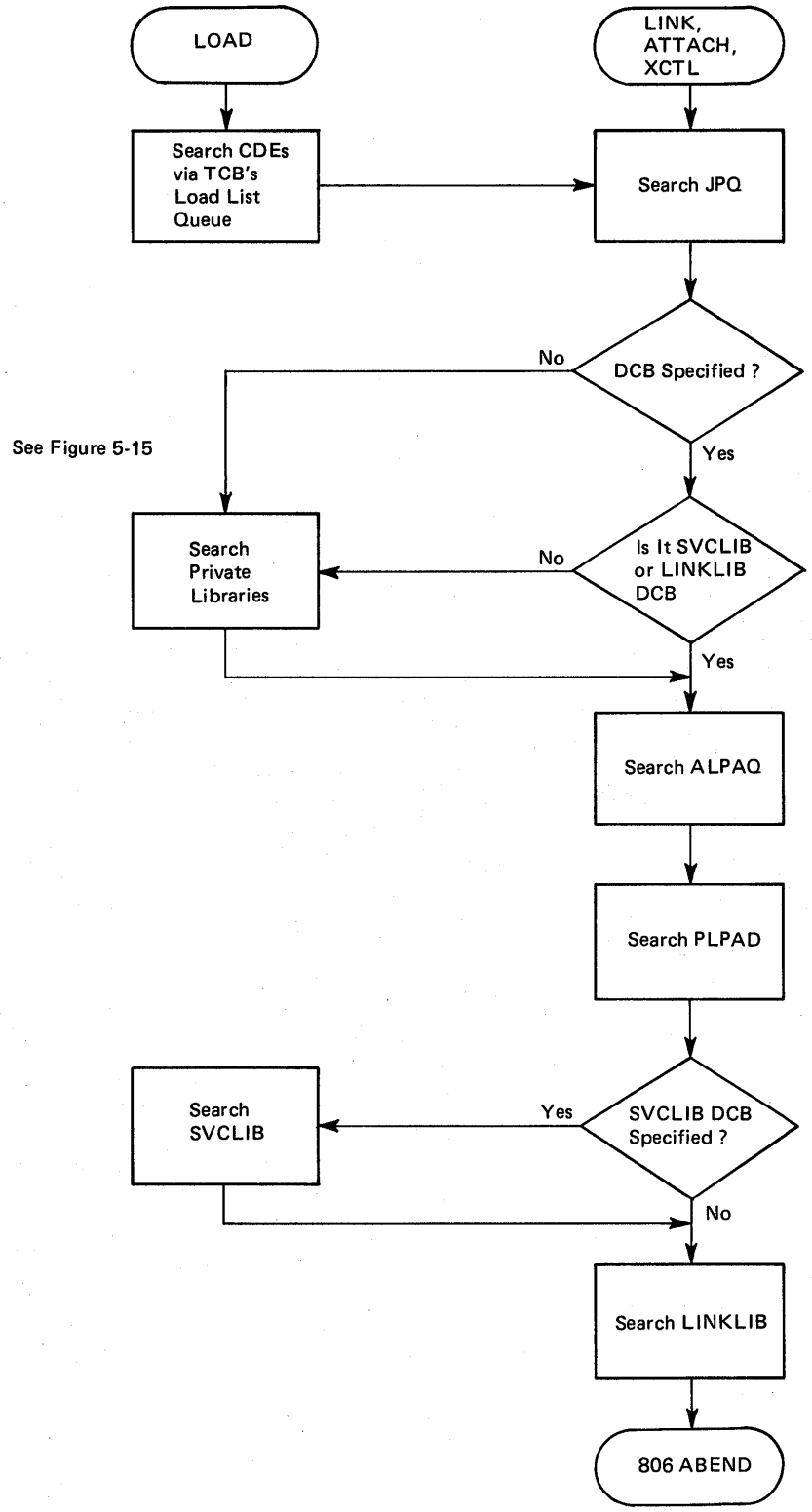
   The search sequence for LINK, ATTACH, XCTL, and LOAD (if the LLE scan is unsuccessful) is shown in Figures 5-14 and 5-15.

2. Useability 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 useability of the associated module. Figure 5-16 describes tests and actions taken by CDALLOC.

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.



Note: For XCTL, if caller is an SVRB, only the ALPAQ and PLPAD are searched

Order of CDEs on ALPAQ

First: Modules activated from the pageable LPA — newest modules first

Second: Modules listed 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

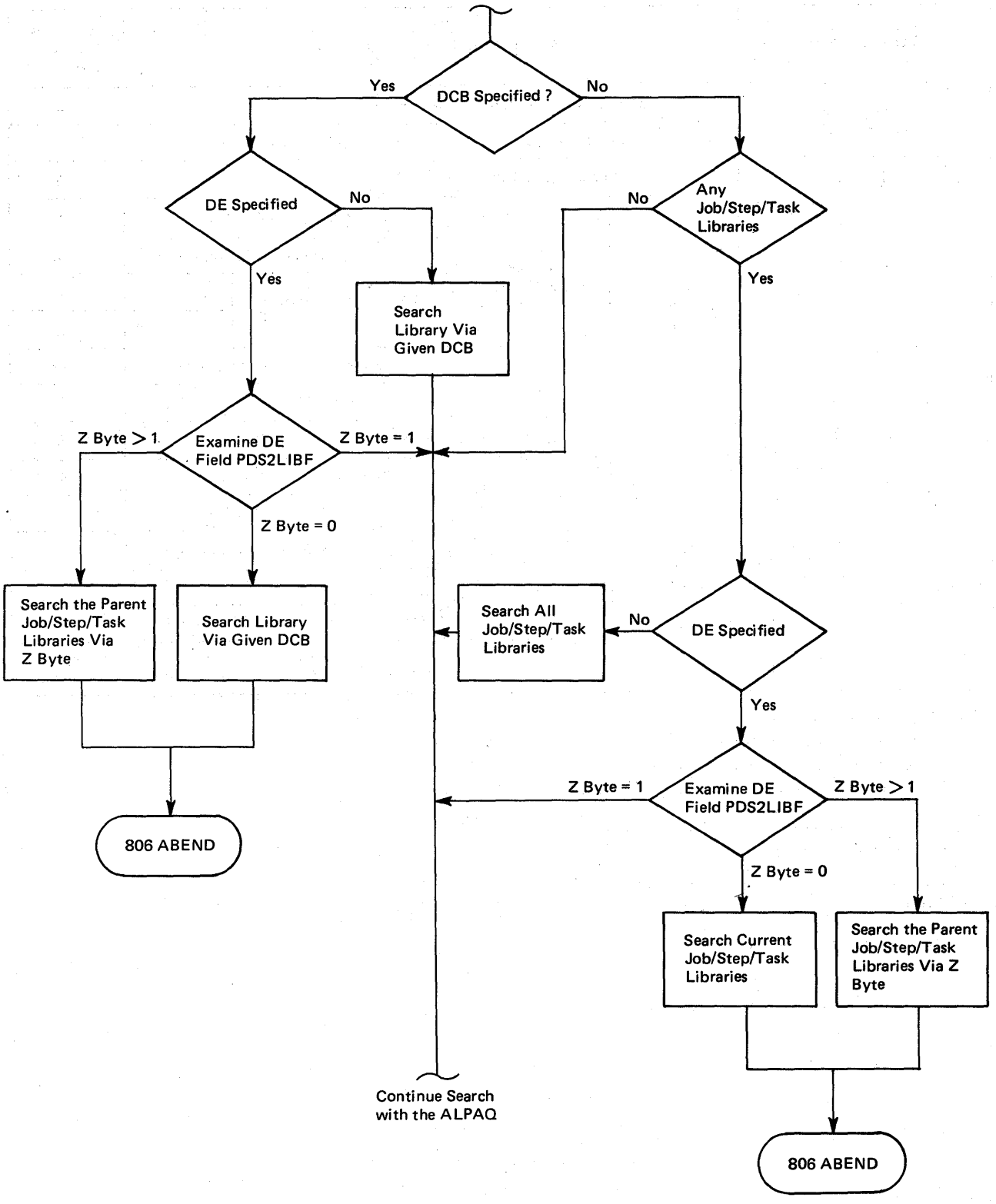


Figure 5-15. Module Search Sequence of Private Libraries

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

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 is 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 useable 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 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.

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 5-17.)

The following are the five basic functions that VSM performs:

Function	Module Performing Function	Comments
Nucelus initialization (NIP)	IEAVNP08	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	GETCELL (get cell)
	IEAVFRCL	FREECELL (free cell)
	IEAVBLDP	BLDCPOOL (build cell pool)
	IEAVDELP	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

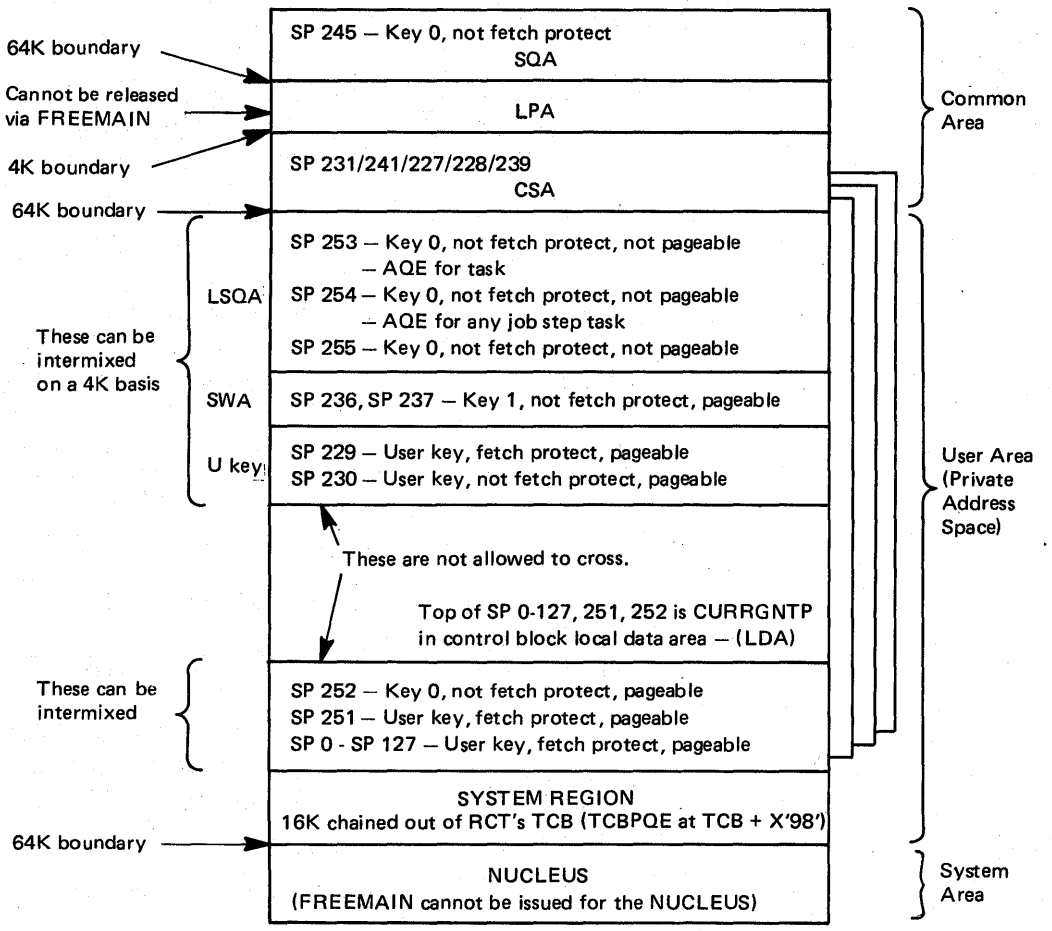


Figure 5-17. Virtual Storage Management's View of MVS Storage

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 5-18. 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 5-18). 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.

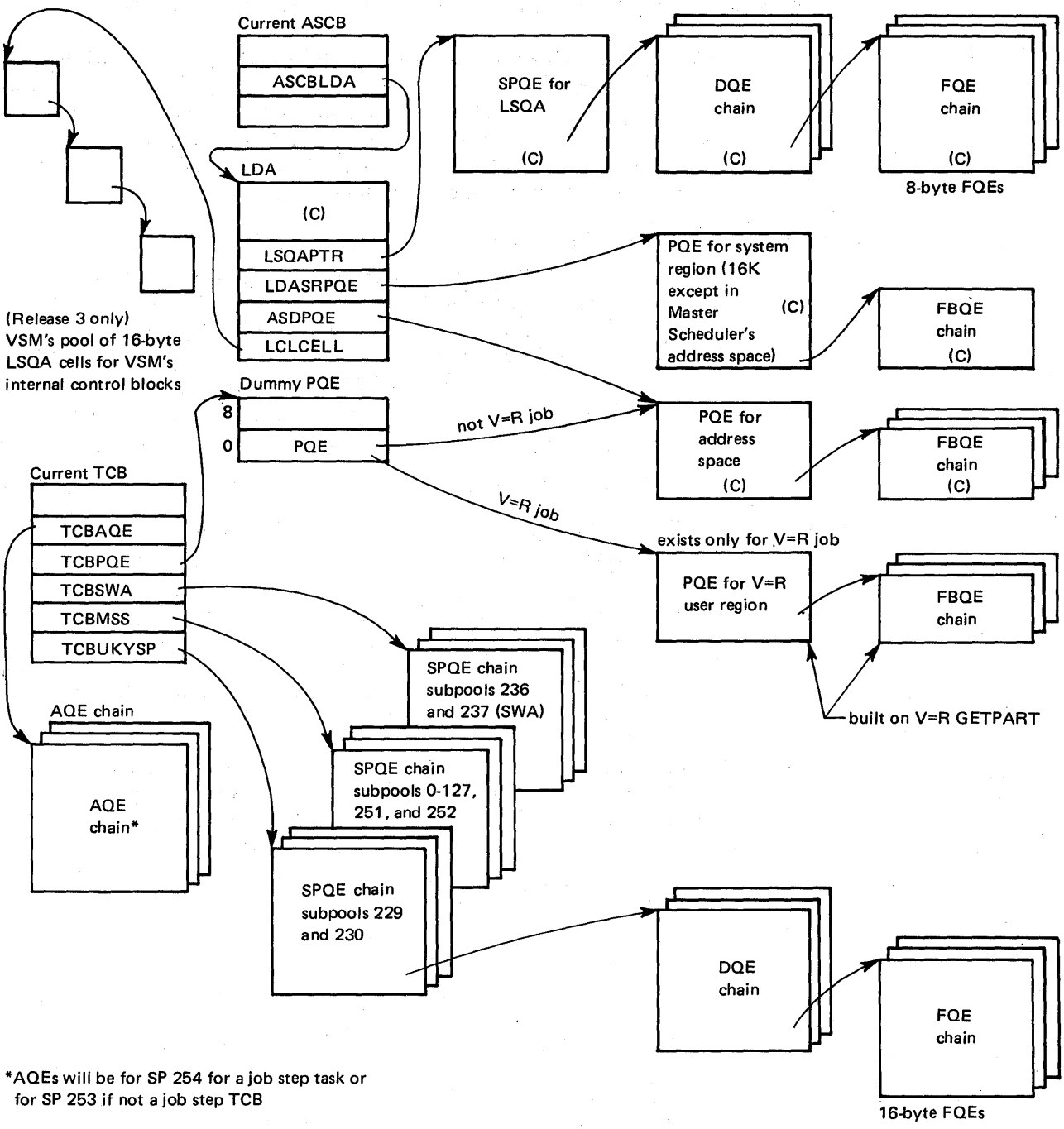
IEAVGCAS's functional recovery routine (FRR) is IEAVCARR. IEAVCARR is divided into the following three sections, corresponding to those of IEAVGCAS.

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 5-18. 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 TCBPQE

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 (IEAVGM00 – GETMAIN/FREEMAIN)

IEAVGM00 satisfies all GETMAIN requests for virtual storage. The control block structure it uses is shown in Figures 5-18 and 5-19. 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.

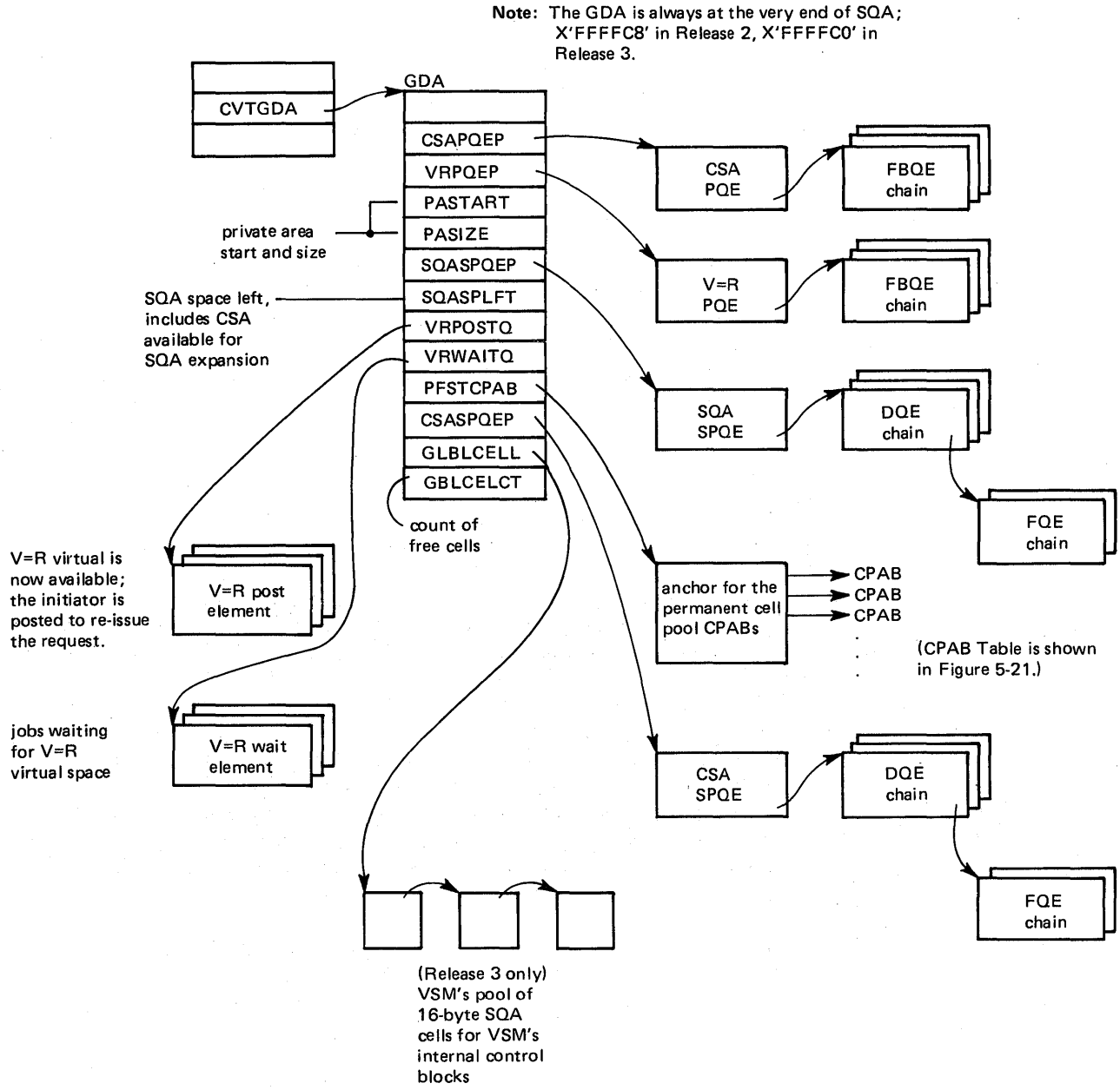
In debugging GETMAIN, the most important information is contained in the original request for virtual storage. This information can be obtained from the trace table, from a parameter list passed by the problem program code, or sometimes from the LDA (local data area).

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)



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 5-19. 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. 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 (ordered 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 5-20. Control blocks are checked to determine if they are in the correct subpool, for example, SQA or LSQA; queues are checked for validity.

Code	Data Length	Data Addresses 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 SQA 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 5-20. 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 5-21. Note that the permanent cell pools have IDs that are negative, for example, RSM01 (X'D9E2D401'), 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. You can catch callers who do not hold the local lock on entering GETMAIN within routine CSPCHK. To do this, test for all of the following conditions:

   - Not NIP (CVTNIP)
   - Not GLBRANCH entry (SALHELD flag offset X'1E' in LDA)
   - Not GETMB or FREEMB (offset X'10' in LDA)
   - Local lock not held (PSAHLHI)
   - Not in the address space creation process (ASXBTCBS not equal to 0)

3. A valid GETMAIN/FREEMAIN that is issued for zero bytes is treated as a no-op.

4. The routine CSPCHK is a good place for a GETMAIN/FREEMAIN trap to occur because CSPCHK is called for every request during the beginning of IEAVGM00 processing.

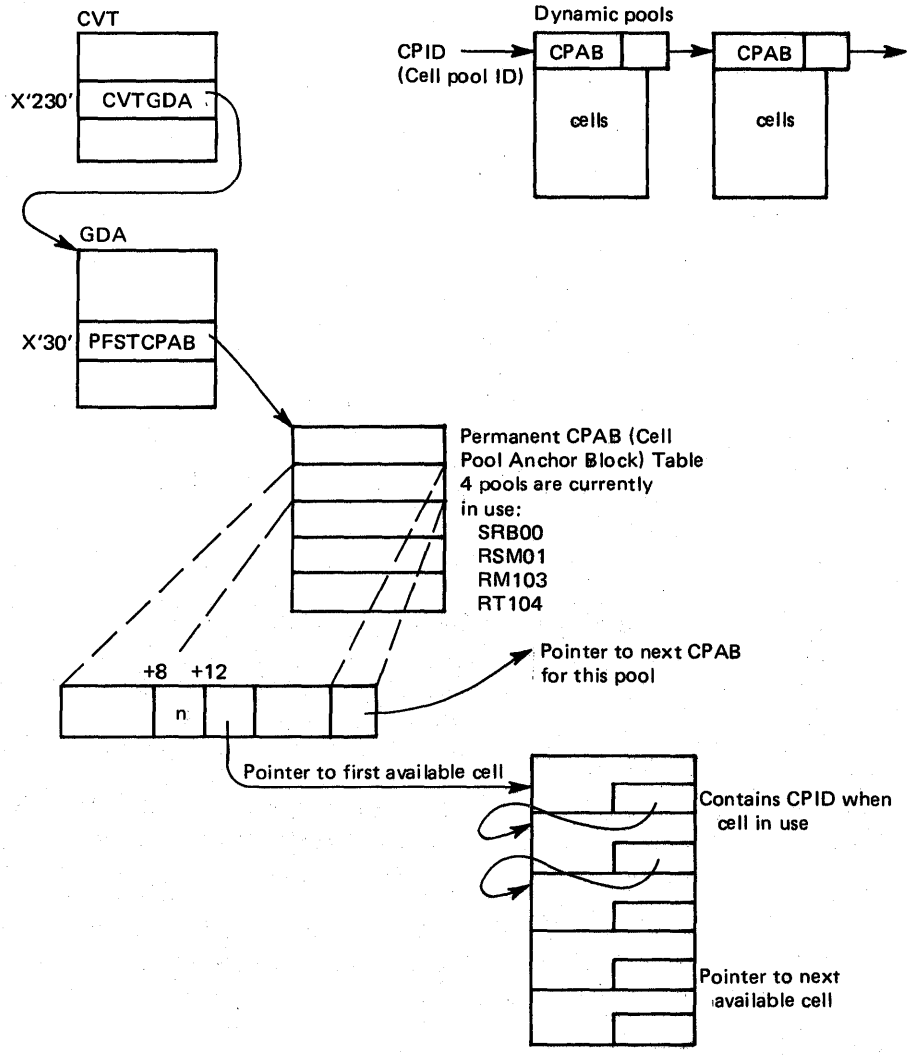


Figure 5-21. VSM Cell Pool Management

5. GETMAIN makes few queue manipulation errors. If IEAVGM00 rejects a request, it is usually because the caller made an error on the interface; the message IEA700I is issued.

6. 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

7. On branch entry to GETMAIN, IEAVGM00 saves registers 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.)

8. Since the GDA occupies the last few bytes of storage, a random store at –4 or –8 overlays the GDA.

9. 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

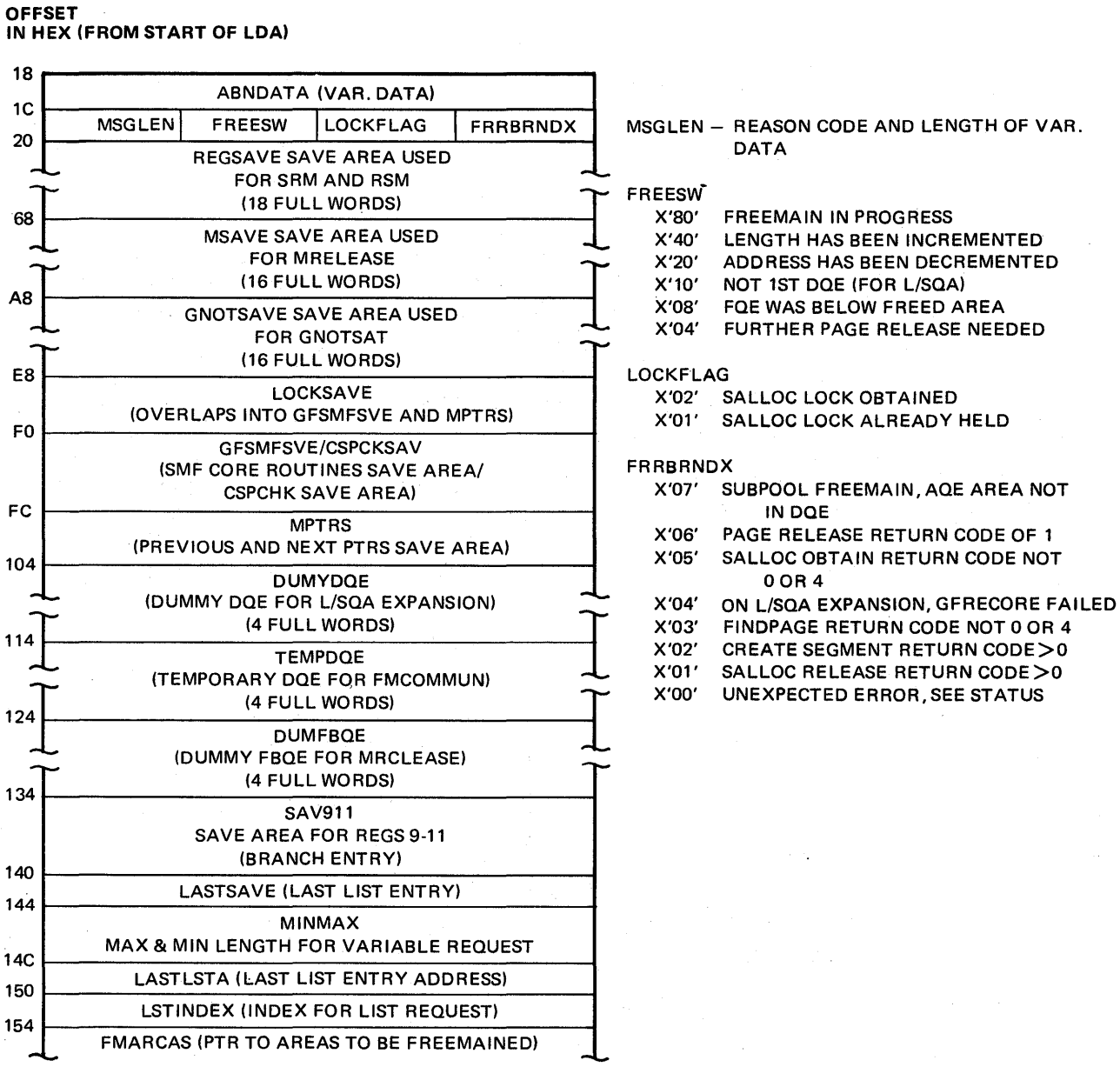
Byte 3: Following flag values:

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.

10. 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'18')



OFFSET IN HEX (FROM START OF LDA)

158	PARMLDA	FRRPARM (FRR PARM AREA ADD)		
15C	CLOPREV (PREVIOUS FQE TO CLOSEST)			
160	CLOSEST (CLOSEST INSIZE FQE)			
164	LARGESTA (LARGEST AVAILABLE)			
168	LARGEST (LARGEST AVAILABLE FBQE)			
16C	LENSAVE (SAVE AREA FOR LENGTH LIST PTR)			
170	SAVESIZE (SIZE OF MULTIPLE OF 4K CORE)			
174	ENDADD (END ADDRESS)			
178	STRTADD (START ADDRESS)			
17C	DIFF/SAVEPQE (DIFFERENCE/PQE PTR IN FBQESPCH)			
180	FIXSTART (STARTING ADDRESS TO CLEAR)			
184	FIXEND (ENDING ADDRESS TO CLEAR)			
188	NOTSATSV (LEN PTR USED BY GNOTSAT)			
18C	NOTSATSI (LDARQSTA SAVE AREA FOR GNOTSAT)			
190	SAVESEG (ADDRESS OF MULTIPLE OF 4K CORE)			
194	LARSOFAR (LARGEST AVAILABLE IN FBQE)			
198	RSTRTADD (ROUNDED START ADDRESS)			
19C	RENDADD (ROUNDED END ADDRESS)			
1A0	VPFP (FIND PAGE ADDRESS TO BE USED)			
1A4	DQESAVE SAVE DQE PTR AND PREVIOUS DQE PTR			
1AC	FMSAVE (SAVE RETURN REG FOR FREEMAIN)			
1B0	PREVFQSV (SAVE AREA FOR PREVIOUS FQE PTR)			
1B4	FQESAVE (SAVE AREA FOR FQE)			
1B8	SPQESAVE (SAVE AREA FOR SPQE)			
1BC	SVRLB (SAVE AREA FOR RLB)			
1C0	SVSIZE (SAVE AREA FOR ROUNDED SIZE)			
1C4	DQESAVE1 SAVE DQE INFO WHEN USING FMAINB			
1CC	FMSAVE1 (SAVE RETURN REG IN FMAINB)			
1D0	FQESAVE1 (SAVE FQE INFO IN FMAINB)			
1D4	PREVFQS1 (SAVE PREVIOUS FQE IN FMAINB)			
1DB	SPQSAVE1 (SAVE SPQE IN FMAINB)			
1DC	SVRLB1 (SAVE RLB FOR FMAINB)			
1E0	SVSIZE1 (SAVE ROUNDED SIZE FOR FMAINB)			
1E4	SAVSVTSV (SAVE LDARQSTA IN FMAINB)			
1E8	FQENXTSV (FQE NEXT SAVE AREA)			
1EC	OLDFQELN (OLD FQE LENGTH)			
1F0	NEWFQELN (NEW FQE LENGTH)			
1F4	SEGTEST (END SEG TEST AREA)		CODE1	CLEARSW
1F8	GMFMSW	FETCH	OUTSW	CODE2
1FC	SAVFRESW	SPID	LSPQEKEY	RQSTRKEY
200	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)

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) and ending at (PVTPFTP) + (PVTLPFN). 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 5-22 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	describe the current real frame and virtual page relationship for a particular virtual address
PVT RSMHDR	basically these are RSM anchors and work areas
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 5-22. Major RSM Control Blocks and Their Functions

Real Storage Manager (RSM) (continued)

Only the leftmost 12 bits of either a real or virtual address are needed to describe a specific real frame or virtual page (a modulo 4K-bytes real and virtual addressing scheme). These 12-bit numbers are multiplied by 16 to form block numbers; for example, VBN0 and RBN0 are four-digit, hexadecimal, virtual and real block numbers. Also, note the following:

- PGTEs contain RBNx values.
- The contents of PVTPFTP plus RBN0 is the address of the PFTE for the frame whose real address is RBN000 through RBNFFF.

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 5-23 shows the relationship among the blocks.

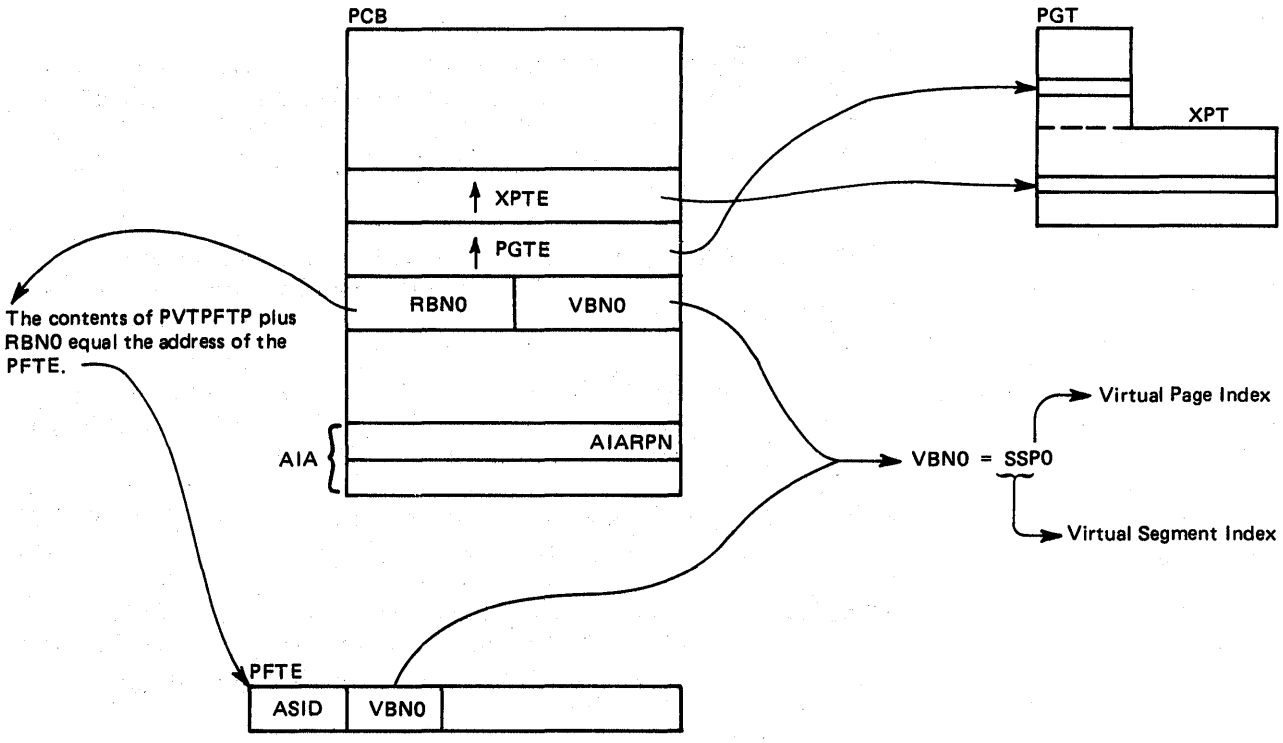


Figure 5-23. Relationship of Critical RSM Control Blocks

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' PCBRBN – This value when 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.

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.

The first portion of the SPCT contains the address of the swap root PCB (SPCTSWRT); the number of fixed and LSQA entries in this SPCT (SPCTFIX and SPCTLSQA); 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 entry.

X'20' The VBN in this entry is for common.

X'10' The page was flagged defer release at swap time.

This flag byte is followed by a three-byte LSID and a two-byte VBN. If the entry is for LSQA, 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.

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.

Page Stealing

Figure 5-24 shows the flow of the page stealing process. The circled numbers in the figure correspond to the notes below.

(1) 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.

(2) 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.

(3) 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.

(4) and (6) IEAVRFR calls a common routine, FREEPAGE, to invalidate selected pages and build a PCB for the page-out process if the page is changed.

If the frame queue from which frames are being selected does not correspond to the current address space or the CFQ, IEAVRFR must schedule an SRB (STEAL) to the appropriate address space in order to get to the PGTE in LSQA.

(5) Finally, IEAVRFR calls ASM to start output paging.

(7) and (9) Entry IEAVRFRA scans the LFQ of the address space it is scheduled into. If PFTSTEAL=1 and if a frame is still stealable and has not been referenced since "Select," IEAVRFRA sets the steal flag. FREEPAGE is then called to steal the frame.

(8) After the frame queue has been scanned, ASM is called and given a string of AIAs.

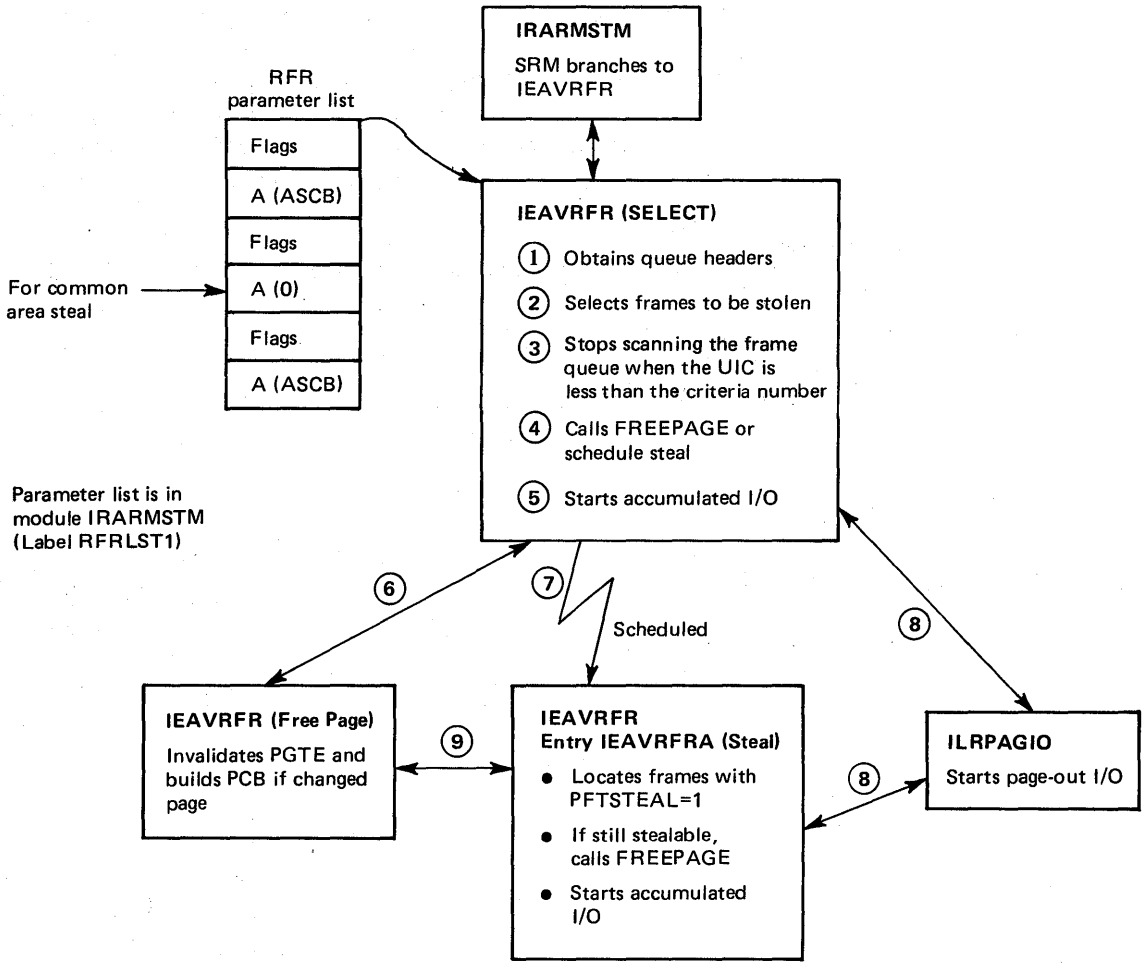


Figure 5-24. Page Stealing Process Flow

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 RBN0 value.
3. The RBN0 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 RBN0 and a DSPID.
2. The RBN0 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 PCBASCB and PCBVBN are the search arguments.
3. When the correct queued PCB is located, the current PCB is inserted in the related PCB chain between the queued PCB and the previous, first-*related* PCB.

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	IEAVPIOP	IEAVSOUT	
IEAVCSEG	IEAVPIX	IEAVSQA	
IEAVEQR	IEAVPRSB	IEAVSWIN	
IEAVIOCP	IEAVPSI	IEAVTERM	
IEAVITAS	IEAVRCF	IEAVRELS at IEAVRELV	(entry point)
IEAVPIOI	IEAVRCV	IEAVFRSB at IEAVPRSR	
	IEAVRFR		

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 or IEAVRCFI	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.
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, ILRPAGIO, 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.

Real Storage Manager (RSM) (continued)

Code (hex)	Meaning
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 communication 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.

Code (hex)	Meaning
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.
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.

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 related.

5. The XPTVIOLP flag can be confusing. If it is on, XPTXAV must be on. SVAUX=1 means the LPID in the XPT is a VIO LPID and not an address space LPID.

   When a page with XAV=1 and SVAUX=1 is stolen, it is paged out under an address space LPID and SVAUX is set to zero. If the next operation on the page is a VIO move out, RSM tells ASM to logically transfer the address space LPID contents to the VIO LPID contents.

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 COD 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.

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 real block number (RBN) in the first twelve bits and a page displacement in the remaining twelve bits (that is, rrrddd – RBN, displacement). The RBN portion of the real address of the PGT is concatenated with zero (RBN0) 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 RBN portion of the real address that corresponds with the initial virtual address. This RBN is concatenated with the displacement portion of the initial virtual address to obtain the desired real address (RBN||ddd).

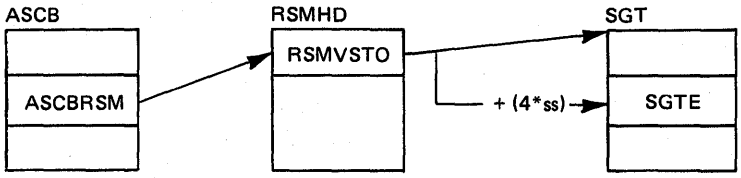
Figure 5-25 shows the relationship of the control blocks used to convert a virtual address to a real address.

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

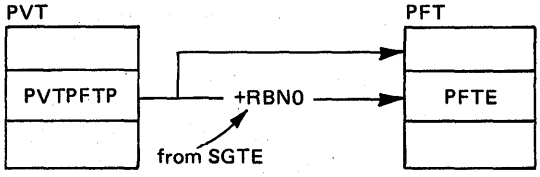
Real address = RBNIIddd
- RBN — real block number
- ddd — displacement within page

① Find the real address of the page table (RBNIId'd'd').



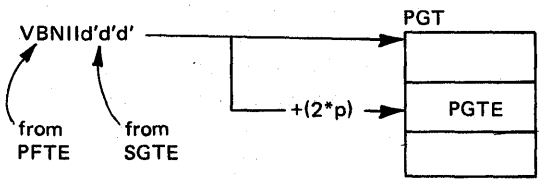
SGTE contains the real address of the page table

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 RBN portion of the real address.



PGTE contains the RBN portion of the desired real address.

④ Concatenate the displacement portion of the virtual address (ddd) with the real block number (RBN) to form the real address that corresponds to the given virtual address.

real address = RBNIIddd

Figure 5-25. 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.

RBN = 307 and RBN0 = 3070
d'd'd' = F20

PVTPFTP = 78760

```
  78760  (PVTPFTP)
+ 3070   (RBN0)
  7B7D0  (address of PFTE)
```

PFTVBN = 87B0

Virtual address of the PGT = VBN||d'd'd' = 87BF20

Step 3: Find the RBN portion of the real address.

```
  87BF20 (virtual address of PGT)
+     12 (2*p)
  87BF32 (virtual address of PGTE)
```

PGTE = 3811

RBN portion = 381

Step 4: Form the real address for the sample.

Real address = RBN||ddd = 381EC0

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 four sections:

- I/O control
- I/O subsystem
- VIO control
- VIO group operators.

I/O control is the link between RSM and the I/O subsystem for paging requests, and between RSM and the I/O supervisor (IOS) for swapping requests. RSM initiates all swapping requests; the I/O is executed by the I/O subsystem and IOS. I/O control accepts the paging/swapping requests from RSM, determines the type of I/O to be done and when it can be started, and notifies RSM when the I/O is completed. I/O control also records the auxiliary storage locations of all virtual pages.

The I/O subsystem 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	*I/O Subsystem*	*VIO Control*	*VIO Group Operators*
ILRPAGIO	ILRPTM	ILRPOS	ILRACT
ILRPAGCM	ILRSRT	ILRGOS	ILRSAV
ILRSWAP	ILRCMP	ILRVIOCM	ILRRLG
ILRSWPDR	ILRMSG00	ILRSRBC	ILRTMRLG
ILRFRSLT		ILRJTERM	ILRVSAMI

Component Functional Flow

ASM provides seven functional services. The first four 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).
- ***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 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 ILRPAGIO for I/O requests. An ASM I/O request area (AIA), or string of AIAs describes the request. ILRPAGIO determines if the request is for a VIO page and, if it is, calls ILRPOS to process it. Otherwise, ILRPAGIO continues to process the request.

For write requests, the previous slot for this page is freed. For read requests, the LSID is obtained from the extended page table entry (XPTE), and put into the AIA. The AIA is queued to the ASM staging queue (ASMSTAGQ) and ILRQIOE is called.

ILRQIOE builds an I/O request element (IOE) for each AIA on the staging queue, and queues IOEs to the paging activity reference table (PART) header or to a PART entry. Each PART entry represents a paging data set and controls activity on the data set. Since a read request is for a particular data set, the read IOE is queued to the PART entry identified by the PART index contained in the LSID. Write IOEs are queued to the PART header because the data set to be used is still unknown.

If an SRB is not already scheduled for ILRPTM, ILRQIOE schedules one. The PART monitor (ILRPTM) scans the PART entries for work and available resources (I/O control blocks) to process the IOEs. For each PART entry with I/O to be done, slot sort routine (ILRSRT) is called. ILRSRT allocates slots for writes interspersed between reads (to minimize arm movement), builds PCCW chains, and issues a STARTIO macro to initiate IOS processing.

When I/O completes, IOS calls ASM's disable interrupt 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 ILRSRT 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 non-LSQA AIAs from the LSQA AIAs and calls ILRPAGIO to process the non-LSQA pages as a regular request I/O function . The LSQA AIAs are queued to the ASM header of the address space (ASHSWAPQ). If there were no non-LSQA AIAs, ILRSWAP immediately calls ILRSLSQA to process the LSQA AIAs. Otherwise, ILRSWAP returns to RSM.

As non-LSQA AIAs complete, ILRPAGCM is given control (see Requesting I/O). When all non-LSQA AIAs have completed, ILRPAGCM calls ILRSLSQA to process the LSQA AIAs. ILRSLSQA, called by ILRPAGCM or ILRSWAP, calls ILRPAGIO to process the LSQA AIAs if there are no available swap sets. Otherwise, ILRSLSQA assigns swap sets and initializes swap channel control workareas (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 LSQA 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 5-26 shows the relationships among the 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 principle 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.

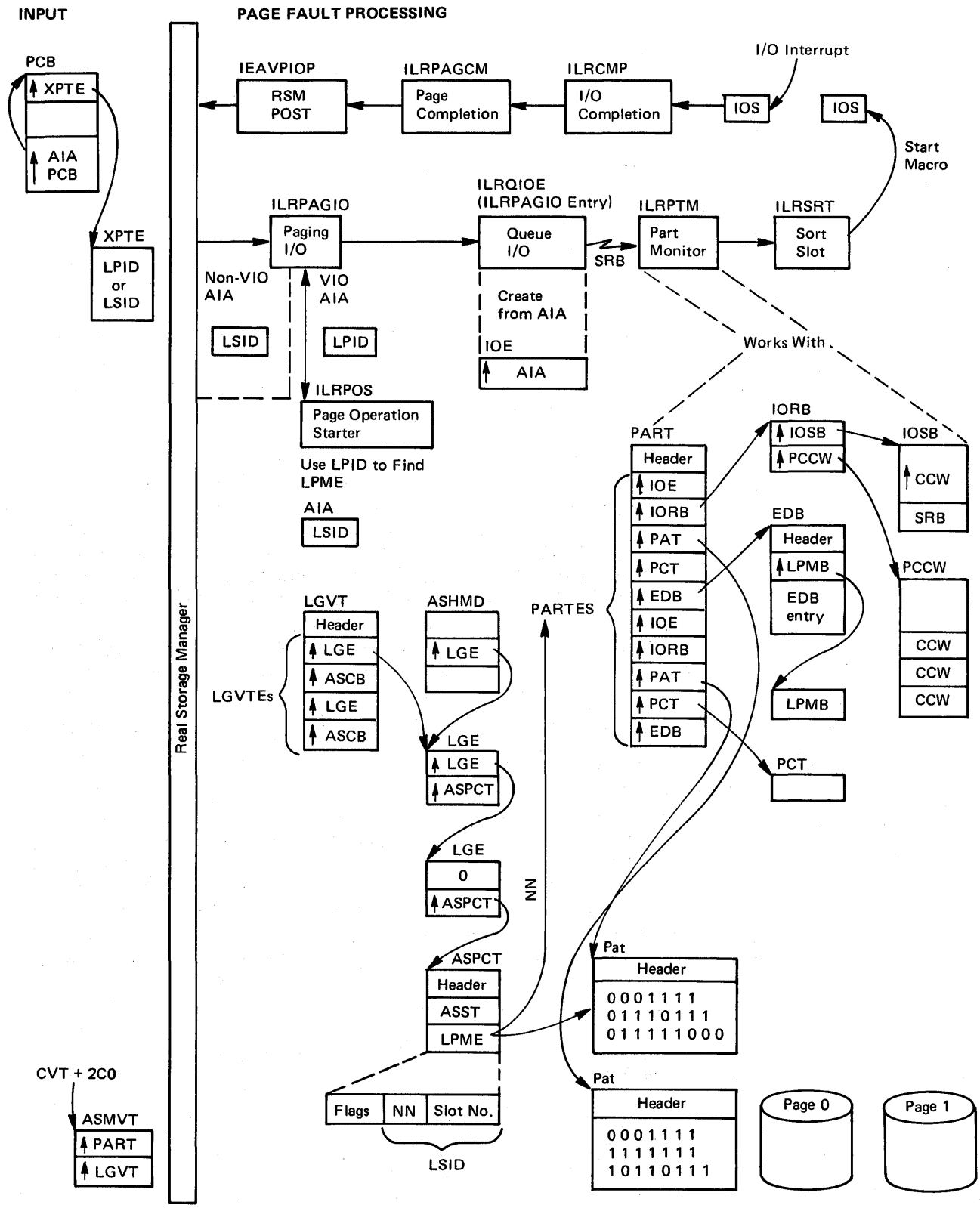


Figure 5-26. Relationship of Important ASM Control Blocks

An ASM class lock exists for each active address space (lockword in ASMHD). The ASM class lock is used by the VIO control modules. ILRPTM uses an ASM class lock (lockword in PART) to serialize the IOE write queues. The ILRCMPDI entry of ILRCMP (ASM's DIE routine) runs in physically-disabled mode since it is running under the I/O interrupt handler. The rest of ASM 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

I/O control modules receive control in the address space of the caller with the exception of ILRSWPDR, which is an SRB executing in master's address space. Note also that ILRPAGCM transfers to the correct address space (TRAS) to update the external page table entry (XPTE), which is in LSQA.

I/O subsystem modules run in SRB mode under master's address space, except for the ILRCMPDI entry of ILRCMP and the modules it calls, which execute in the address space interrupted by the I/O completion.

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 master'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 four 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. There are three expandable cellpools for work areas, ACEs, BWKs, and SWKs. The IOE cellpool is not 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.

MP Considerations

ASM takes advantage of MP by allowing both the I/O subsystem and the ILRSWPDR module of I/O control, to execute concurrently on both processors. This is achieved through extensive use of compare and swap logic. An individual PART entry or SART entry is 'flag' locked by the processing processor, but ASM can process a request for another entry on the second processor.

Interfaces With Other Components

Four 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 subsystem and ILRSWPDR routine of 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.

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 address for the current address space or the address space identified by an input parameter in register 0 or 1. The ASMHD is addressable as part of the RSMHD.
	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.

The I/O subsystem does not use the ASMHD and therefore does not maintain register 2 convention.

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. WSACASMD points to the 256 byte save/work area for ILRCMPDI (ASM's DIE routine). WSACASMS points to two contiguous 256-byte save/work areas, the first for ILRPTM, and the second for ILRSRT. The first one is also used by ILRSWPDR and ASM's I/O appendages (ILRCMPAE, ILRCMPNE, and ILRCMP).

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 waiting for processing for VIO requests.

If the PARTE is locked, part monitor (ILRPTM) has called or is about to call slot sort (ILRSRT). 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 two counts, ASMIORQR (ASMVT + X'28') and ASMIORQC (ASMVT + X'2C'), which contain both the number of I/O 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) are not equal.

- The ASMIOCNT contains a count of the number of I/O requests sent to IOS but not received by ILRCMP. It is necessary to determine why paging requests received by ASM are not completing. To learn more about these requests, it is necessary to follow I/O control block chains.

- The ASMSRBCT field indicates whether ASM's SRB for ILRPTM (ASM PART monitor) has been scheduled. If this field is not zero, ASM's 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, and four for each page data set on a drum. 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 points to the IOSB and to the first of a queue of PCCWs. For an active I/O request, the third word into the PCCW points to the associated AIA, and the second 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 extended 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 RSMHD. The ASMHD is part of the RSMHD. 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.

With the appropriate ASPCT, now use the RPN portion of the LPID as an index to locate the LPME containing the associated LSID.

Figure 5-27 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".

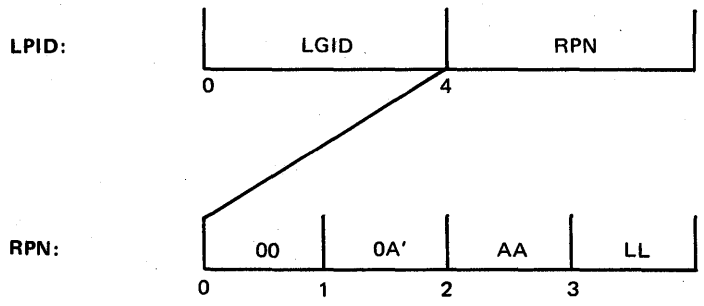


Figure 5-27. Locating An LSID From An LPID (Part 1 of 2)

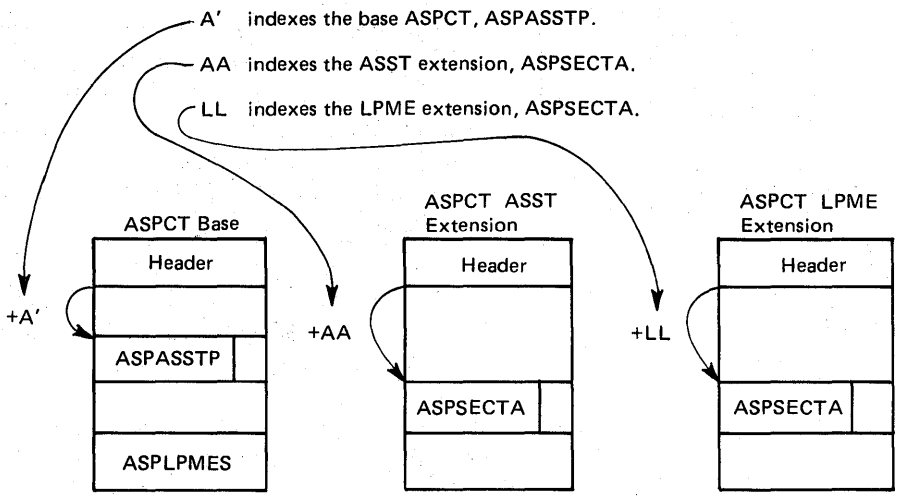


Figure 5-27. Locating An LSID From An LPID (Part 2 of 2)

Locate PART and PAT Bit

Suppose LSID 0004E3B0 was found in the XPTE that represents the sample address 07A12C:

PART entry index is 04.

Relative byte address (RBA) is E3B0.

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. PATCYLSZ, slots per cylinder, is the number of significant bits in each cylinder mapping.

For device 2305-2:

PATCYLMW is 1

PATCYLSZ is 1A (26).

To locate the bit in the PAT map for slot E3B0(58288):

1. address of map word = (address of PATMAP) + 8964 = (address of PATMAP) + ((58288/26) x PATCYLMW x (bytes in a word))
2. bit in the map word (origin 0) = 58288/26 = 22.

Figure 5-28 shows the control blocks involved in relating a virtual address to the PART and PAT.

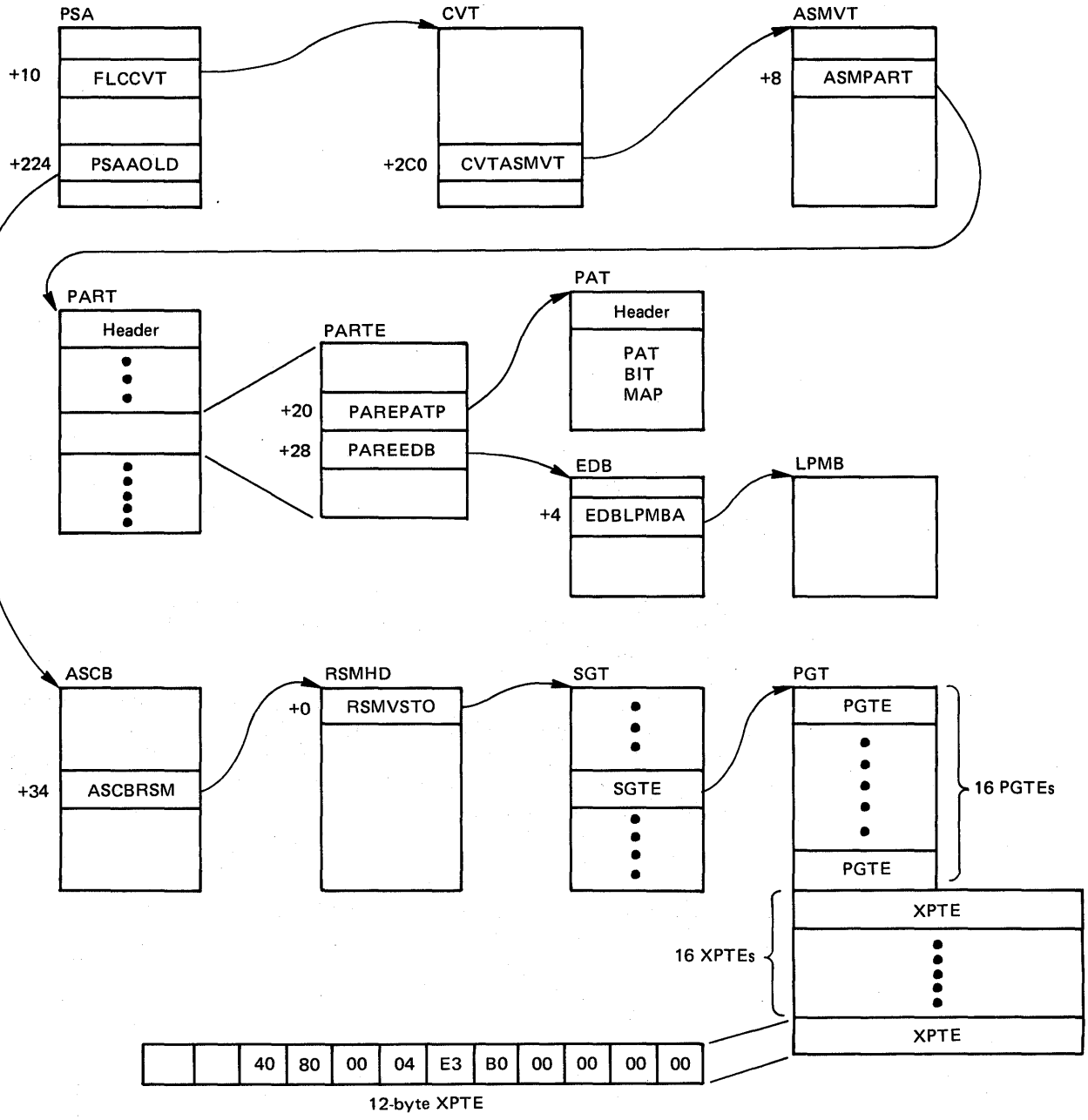


Figure 5-28. Relating the Virtual Address to the PART and PAT

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 AMB extent descriptor block (EDB) for the data set. The EDB and its associated LPMB(s) describe the data set on the device. The EDB consists of an 8-byte header followed by entries, one for each extent. The second byte of the header contains the number of entries and the next two bytes contain the length of an entry.

The relative byte address (RBA) is calculated by multiplying the slot number by 4K. The extent containing this slot is found by comparing the RBA to the low and high RBAs in each extent. These are found at X'C' and X'10' in the EDB entry. The second word of the entry thus found points to an LPMB.

To find the relative cylinder, subtract the low RBA of the extent from the RBA and divide by the allocated unit size (LPMB + 4). To find the relative track, take the remainder in the division just performed and divide it by the bytes per track (LPMB + 8).

The remainder of the bytes-per-track division is the first step toward finding the record number. Add 4095 to the remainder, divide the result by 4096, and add 1. The result is the R of MBBCCHHR.

To find the CCHH, multiply the relative cylinder by the number of tracks per allocated unit (LPMB + X'10') and add the relative track (as computed by the method just shown) and the starting track from the EDB entry + 8. Divide this result by the number of tracks per cylinder (LPMB + X'12'). The quotient is the CC and the remainder is the HH.

For example, when given an RBA of X'E3B0' (58288), calculate the MBBCCHHR for device 2305-2.

(Reference IDAEDB and IDALPMB macros for fields) M = extent number = 0
BB = return code = 00

Relative CC = (RBA – EDBLORBA)/LPMAUSZ
= 58,288 – 0/53,248 = 1

Relative HH = (RBA – EDBLORBA)//LPMAUSZ/LPMBPTRK
= 5040/13,312 = 0

CC = (Rel CC * LPMTRRAU + Rel HH + EDBSTTRK)/LPMTPC
= (1 * 4 + 0 + 8)/8 = 1st cylinder

HH = (Rel CC * LPMTRRAU + Rel HH + EDBSTTRK)//LPMTPC
= 12//8 = 4th track

R = [((((RBA – EDBLORBA)//LPMAUSZ)//LPMBPTRK) +
LPMBLKSZ-1)/LPMBBLKSZ] +1

= ((5040 + 4095)/4096) + 1 = 3rd record

Therefore: MBBCCHHR = 0000000001000403

Unuseable 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 unuseable. For page data sets, the DSBD flag in the PART entry is turned on. 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) ILR009I.

If ASM is able to continue processing without the unuseable data set, message ILR009I is written to the operator. This message indicates which volume contains the unuseable 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 unuseable 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 unuseable:

- 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).

- 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 unuseable data set (or several unuseable data sets) caused one of the following conditions:

- The unuseable paging data set contains PLPA pages and the duplex data set, if any, is already unuseable or full.

- The unuseable 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 unuseable).

- The unuseable paging data set is the last local paging data set.

Page/Swap Data Set Errors

Figure 5-29 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		ILR009I, ILR010I	Duplex Only
	Common Full	PLPA Available	ILR006I	Spill to PLPA
		PLPA Unavailable	ILR010I	Duplex Only
	Common Bad		ILR009I, 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		ILR009I	Stop Writes to Bad Data Set
	Last Local Bad		ILR008W	Wait X'02E'
	Swap Bad		ILR009I	Stop Swap-outs to Bad Data Set
	Last Swap Bad		ILR009I	All Swap-outs Done to Page Data Sets

*Available — Data Set Neither Full Nor Bad
**Unavailable — Data Set Either Full or Bad

Figure 5-29. 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 ILRPTM (address of SRB in ASMPSRB, X'58' into ASMVT), 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.
- Scan the PART entries for the PAREFSIP flag which indicates that the PART entry is locked and that sort slot or part monitor should be processing. Check the PART and PARTE IOE queues for requests waiting for I/O. 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 ILRPTM 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 ILRPTM 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 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 principle 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:

XPTEs	– complete coverage.
PCB/AIAs	– complete coverage, except AIA noted below.
SPCTs	– complete coverage.
SART	– complete coverage, except where noted below.
SATs	– 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
  - ASMNOCWQ
  - ASMCALLQ
  - ASMNODPX
  - ASMPLPAF
  - ASMCOMMF
- LGVT pointer

Auxiliary Storage Manager (ASM) (continued)

ASMVT (continued)	– 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.
PART	– Count of local page data sets.

Modules whose processing is serialized by the SALLOC lock are:

ILRPAGIO	– complete coverage, held by caller.
ILRPAGCM	– complete coverage, obtained at entry.
ILRFRSLT	– complete coverage, except ILRFRSL1 entry point where caller may or may not hold the lock. The lock is not obtained by this module, held only if by caller.
ILRSWAP	– complete coverage, held by caller.
ILRPTM	– only obtained to process data set full conditions for non-local page data sets.
ILRCMP	– only obtained to process I/O completion error conditions that may require operator notification.
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 set data and update the count of local 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. A class lock is also defined for the PART write queues with its lock word in the PART header. This lock serializes the four FIFO IOE write queues in the PART. 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.

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. The I/O subsystem and VIO group operators run unlocked and are the principle users of compare and swap. Control blocks serialized via CS include:

PART – a special CS lock exists for each PARTE controlled by PART monitor. This lock is used mainly for execution control. Most fields are still serialized by CS. The IOE write queues are the exception described above.

PATs – complete coverage .

ASMVT – I/O subsystem and group operator sections.
I/O error count.
unreserved slot count.
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 swap driver or I/O completion processing of the I/O subsystem are updated with CS.

The ASM modules that run without locks, using CS to serialize control block fields are:

ILRSWPDR
ILRPTM
ILRSRT
ILRCMP
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, 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, for example, the AIAs on the ASMSTAGQ. 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 ILRPAGIO 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 processing.

Recovery Structure

ASM has eight recovery routines for ASM mainline:

- ILRIOFRR is an FRR that provides recovery for ILRPAGIO, ILRPOS, ILRPAGCM, and ILRVIOCM. It also acts as a router, giving control to the routines in ILRSWP01.
- ILRSWP01 contains recovery routines for ILRSWPDR and ILRSWAP.
- ILRSRT01 is an FRR that provides recovery for part monitor (ILRPTM) and slot sort (ILRSRT).
- 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.

- 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:

ILRPAGIO	('01'X)	ILRSWPDR	('05'X)	ILRCMPDI	('09'X)
ILRPAGCM	('02'X)	ILRGOS	('06'X)	ILRCMPNE	('0A'X)
ILRSWAP	('03'X)	ILRPTM	('07'X)	ILRCMPAE	('0B'X)
ILPTRPAG	('04'X)	ILRSRBC	('08'X)	ILRCMP	('0C'X)

- 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 saves the ATA. ILRGOS01, ILRCMP01, and ILRSRT01 save the ATA and EPATH. ILRTMI01 saves only the EPATH.

ASM Diagnostic Aids

This section contains diagnostic aids that are helpful in debugging problems in ASM. Topics included are:

- COD ABEND Meanings for ASM
- ASM Recovery Control Blocks
- Additional ASM Data Areas

COD ABEND Meanings for ASM

An RSM routine has found one of the following conditions which should not occur and has set the appropriate return code in register 15:

RC 4 – The count of available swap sets for a specific swap data set 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.
	ATAMPGIO	01	ILRPAGIO module ID.
	ATAMPGCM	02	ILRPAGCM module ID.
	ATAMSWAP	03	ILRSWAP module ID.
	ATAMTRPG	04	ILRTRPAG module ID.
	ATAMSWPD	05	ILRSWPDR module ID.
	ATAMGOS	06	ILRGOS module ID.
	ATAMPTM	07	ILRPTM module ID.
	ATAMSRBC	08	ILRSRBC module ID.
	ATAMCMPD	09	ILRCMPDI module ID.
	ATAMCMPN	0A	ILRCMPNE module ID.
	ATAMCMPA	0B	ILRCMPAE module ID.
	ATAMCMP	0C	ILRCMP module ID.
1	ATASFLGS	3	Bit map representing logical sections of ASM routines; set to 1 on entry, set to 0 on exit.
	ATAQIOE	800000	ILRQIOE flag.
	ATASLSQA	400000	ILRSLSQA flag.
	ATASCOMP	200000	SWAPCOMP flag.
	ATAVIOCM	100000	ILRVIOCM flag.
	ATAPCOMP	080000	PAGECOMP flag.
	ATAPOS	040000	ILRPOS flag.
	ATAPAGIO	020000	ILRPAGIO flag.
	ATAPAGCM	010000	ILRPAGCM flag.
	ATASWAP	008000	ILRSWAP flag.
	ATATRPAG	004000	ILRTRPAG flag.
	ATASWPDR	002000	ILRSWPDR flag.
	ATASRT	001000	ILRSRT flag.

The remaing flags are reserved:

ATARFLGS	2	Other recovery flags.
ATACNVRT	8000	ILRSLSQA flag-converting between forward chained AIA's and lateral chained ATA's.
ATASGNST	4000	ILRSLSQA flag-in ASSIGNSET subroutine.
ATASCCWP	2000	ILRSLSQA flag-in SCCWPROC subroutine.
ATABADPK	1000	ILRCMPAE flag-in BADPACK subroutine.

Disp	*Name*	*Size*	*Description*
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	ATAWORDS	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 ILRSRT01:			
8	ATAWORKA	4	Address of PTM work-area cell.
C	ATAEPATH	4	Address of EPATH.

Disp	*Name*	*Size*	*Description*
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 accomodate the data to be tracked. The mapping of the EPATH is dependent on the recovery routine or mainline routine including the macro.

EPATH for ILRPTM, ILRSRT, and recovery routine ILRSRT01:

Disp	*Name*	*Size*	*Description*
0	EPAPARM	4	Address of parameter list.
4	EPAIOEIP	4	Address of IOE currently being processed.
8	EPAIOEQP	4	Address of first IOE on 'WORK' read IOE queue.

Disp	*Name*	*Size*	*Description*
C	EPAFFIOE	4	Address of first IOE on free IOE internal queue.
10	EPALFIOE	4	Address of last IOE on free IOE internal queue.
14	EPAWRTQ	4	Address of write queue from which last write IOEs removed.
18	EPAWTPAT	4	Address of SCYLWRT which is used to update current CYL Map.
1C	EPACYLA	4	Address of current CYL Map.
20	EPAMSPAD	4	Address of 2 word parameter list for ILRMSG00. Also serves as a switch for ILRPTM.
24	EPAWRTCT	2	Number of writes prepared for current CYL.
26	EPACPUID	2	Processor locking count for current part monitor processing.

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 work-area 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.

Disp	*Name*	*Size*	*Description*
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.
	*	X'01'	Reserved.
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.

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 MSGBUFER) are not contained in *OS/VS2 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

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.

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. Part 3 of *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 decisions 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 a member of SYS1.PARMLIB called the installation performance specification (IPS). Through the IPS, the installation divides its types of work into distinct groups called *domains,* assigns relative importance to each domain, and specifies the desired performance characteristics for each address space within these domains. A secondary input to the SRM is another member of parmlib, the OPT member. Through a combination of 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 does not occupy real storage. It has been swapped-out, because it cannot be put into execution.
3. Out – The working set of an address space in this state does not occupy real storage; however, the address space is capable of executing and can be considered for swapping-in.

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 sub-classes; 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.

In the OUCBQFL field (OUCB + X'10'), 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.

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 is 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 restore-complete 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 5-30 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 at 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 (RMCT + X'1D8'; bit 2)

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 (RMCT + X'1D8'; bit 0)

Indicates that the number of available SQA pages is critically low. If MCTSMS1 (RMCT + X'1D9'; bit 4) is 1, the operator was notified of this situation.

MCTSQA2 (RMCT + X'1D8'; bit 1)

Indicates that the number of available SQA pages has fallen below a second, more critical threshold than the one noted above. If MCTSMS2 (RMCT + X'1D9'; bit 5) is 1, the operator was notified of this situation.

MCTASM1 (RMCT + X'1D9'; bit 0)

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.

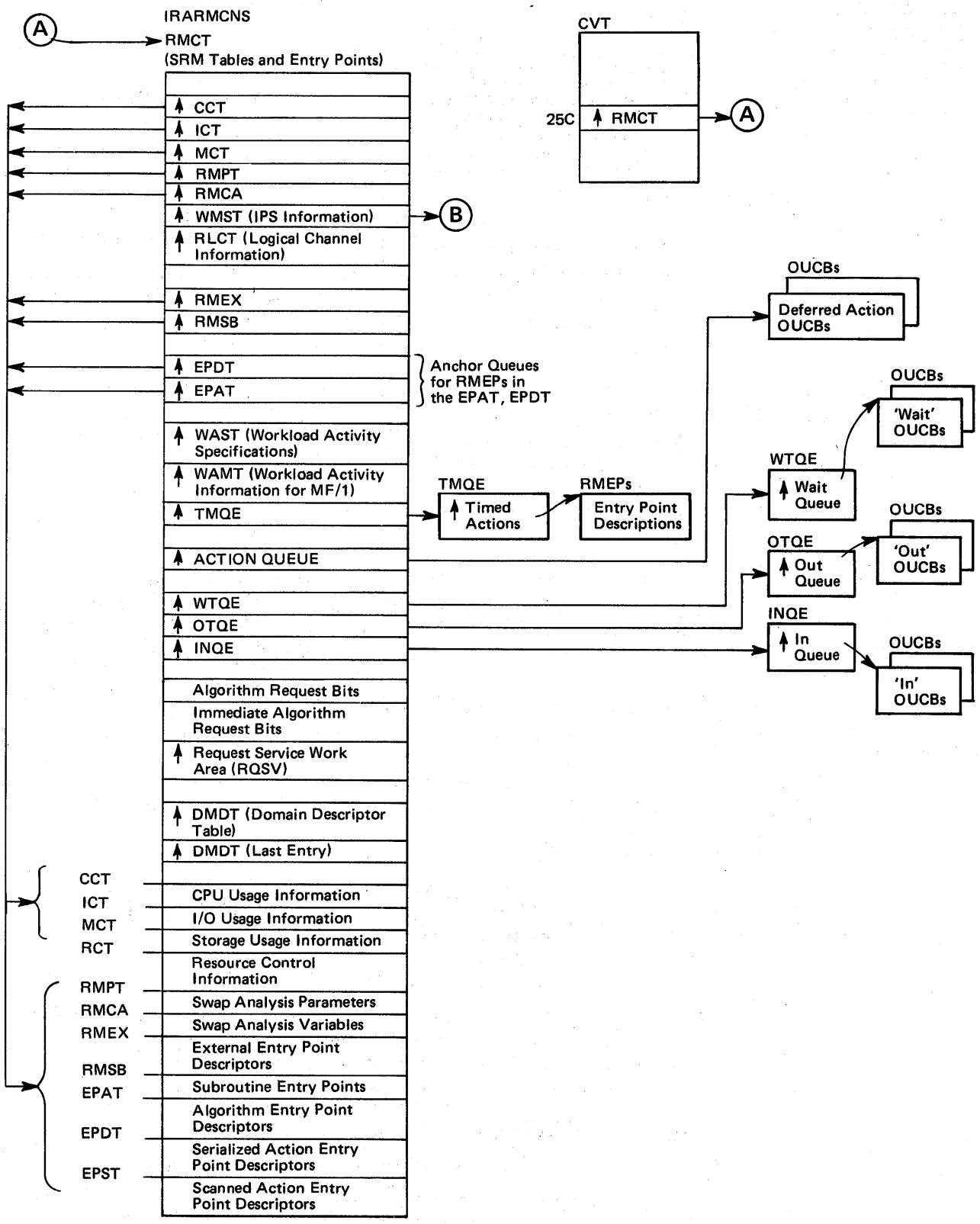


Figure 5-30. SRM Control Block Overview (Part 1 of 2)

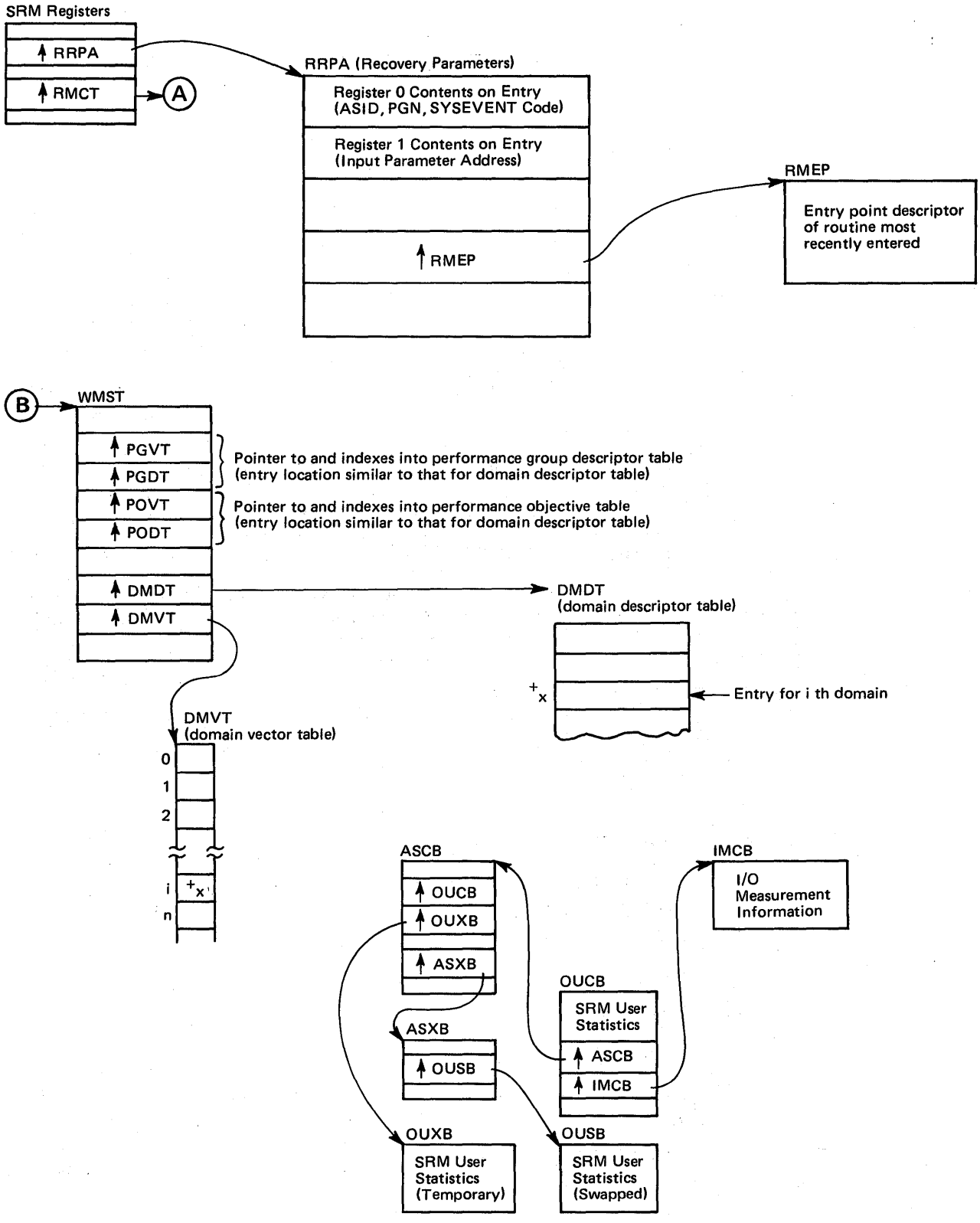


Figure 5-30. SRM Control Block Overview (Part 2 of 2)

MCTAMS2 (RMCT + X'1D9'; bit 3)

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 (RMCT + X'1D8'; bit 3)

This bit indicates that the count of fixed pages in the system is above the threshold value, PVTMAXFX, and the real storage manager (RSM) has called SRM to swap-out the users responsible for the shortage of pageable frames. If MCTFX1 (RMCT + X'1D9'; bit 6) is 1, the operator was informed of this situation.

RCVUICA (RMCT + X'21E'; halfword)
RCVCPUA (RMCT + X'220'; halfword)
RCVASMQA (RMCT + X'224'; halfword)

These values are the system contention indicators that the resource monitor examined for the last interval. They represent in the order given; the average unreferenced interval count (UIC), the average processor utilization, and the average ASM queue length. Based on these values, the target MPL for a domain is altered.

RMCAINUS (RMCT + X'29E'; 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 (RMCT + X'138'; 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.

Individual User Indicators

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.

OUCBMWT (OUCB + X'15'; bit 7)

If this bit is on, the SRM has detected that this user has not been dispatched, but was occupying storage for at least Δ 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 (OUCB + X'12'; bit 5)

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 (OUCB + X'11'; bit 6)

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 (OUCB + X'12') 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.

OUCBFXS (OUCB + X'12'; bit 7)

When this bit is on, it indicates that the user has been swapped-out of storage because the user's address space had allocated to it the greatest number of fixed frames when a pageable frame shortage occurred.

OUCBJSAS (OUCB + X'17'; bit 1)

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 (OUCB + X'17'; bit 0)

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 (OUCB + X'25'; 1 byte)

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 or transwap SYSEVENT issued

08 – ENQ exchange by swap analysis
09 – Exchange based on recommendation values by swap analysis
0A – Unilateral swapout by swap analysis.

OUCBRDY (OUCB +X'56'; bit 0)

This flag indicates that ready work became available for this address space which was swapped-out due to a long wait. The address space is now capable of executing and is a candidate for swap-in.

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.

Module Entry Point Summaries

Figure 5-31 shows a cross reference between SRM modules and entry points. Descriptions of selected SRM modules and entry points are:

System Resources Manager (SRM) (continued)

IRARMINT – SRM Interface Routine

IGC095 – SVC entry point to SRM.

IRARMI00 – Branch entry point to SRM.
Handle all external SYSEVENTs.

IRARMI48 – Branch entry point to the SRM.
Handle the internal SYSEVENT (48).

IRARMI01 – Entry point from IRARMEVT or IRARMCTL.
Return to the SYSEVENT issuer.

IRARMII0 – Entry point from IRARMEVT.
Abend a user of the SRM.

IRARMEVT – SRM SYSEVENT ROUTER

IRARMEVT – SYSEVENT processor.
Begin to process the indicated SYSEVENT.

IRARMXVT – SYSEVENT retry.
Prepare a retry of a SYSEVENT that had incurred a system error.

IRARMDEL – Synchronize address-space delete processing.

IRARMIPS – Set new IPS.
Invoke IRARMSET to establish a new IPS.

IRARMUXB – Synchronize OUXB deletion at swapout-completion time.

IRARMSTM – Storage Management Routine

IRARMPR1 – Page Replacement Normal Processing.
Examine each user in main storage and the system pageable area, and call RSM real frame replacement to update UICs for each user.

IRARMPR5 – Page Replacement Real Page Shortage Force Steal.
Steal as many pages as required to relieve a real page frame shortage. The steal decision is made at entry IRARMMS2. The oldest unreferenced pages are stolen first.

System Resources Manager (SRM) (continued)

IRARMMS2 – Real Page Shortage Prevention.
Calculate the number of frames necessary to reach the O.K. threshold, and schedule IRARMPR5 processing (if a real page shortage exists). Inform the operator of users that have the greatest number of fixed frames and direct the swaps of these users (if a pageable real page shortage exists).

IRARMMS6 – Main Storage Occupancy Long Wait Detection.
Discover users who have gone into long wait without notifying SRM. Swapout such users, if swappable.

IRARMASM – Auxiliary Storage Shortage Monitoring.
Monitor the extent of auxiliary shortage allocation. If auxiliary pages are in short supply, inform the operator and direct swaps of users who are most rapidly acquiring auxiliary storage slots.

IRARMSQA – SQA Shortage Message Writer.
Inform operator of system queue area shortages.

STEAL – Internal STM Steal Subroutine.
Add users to RFR interface list until full, then call RSM real frame replacement (RFR) routine (via IRARMI03) and record the number of pages stolen.

IRARMSRV – SRM Service Routine.

IRARMI02 – Interface to ASCB CHAP.

IRARMI03 – Interface to RSM's real page frame replacement.

IRARMI04 – Obtain or free SQA storage.

IRARMI05 – Requeue SRM TQE routine.

IRARMI06 – Cross-memory post entry point.

IRARMI07 – SWAP SRB SCHEDULE routine.

IRARMI09 – RECORD entry point.

IRARMR16 – Set a return code of 16 in register 15 and return. (Dummy routine)

IRARMERR – SRM's Functional Recovery Routine.

IRARMRR1 – Functional recovery for globally-locked entries (entries to SRM in which the SRM lock could not be obtained).

Retry the failing SRM routine when possible. Otherwise, percolate the error.

IRARMRR2 – Functional recovery for non-globally-locked entries (entries to SRM in which the SRM lock was obtained). Validate queues and clean up. Retry the failing routine if possible; otherwise percolate the error.

RMRR2RTY – Return to RTM indicating retry.

RMRR2PER – Return to RTM indicating percolation.

RMRR2INT – FRR initialization.

RMRR2VLD – Validate control blocks.

RMRR2GST – Release the dispatcher lock in order to call IRARMI04.

RMRR2CKQ – Verify the location of an OUCB.

RMRR1VFB – Verify addresses.

RMRR2REQ – OUCB enqueue routine entry point.

RMRR2SPR – Return with the return code in register 15.

IRARMCPM – Processor Management.

IRARMAP1 – Automatic Priority Group Reorder Processing.
Recompute dispatching priorities for all APG users in main storage. Invoke ASCBCHAP for each user whose dispatching priority has changed.

IRARMEQ1 – ENQ/DEQ Algorithm ENQ Time Monitoring.
Stop giving extra CPU service to users with ENQHOLD SYSEVENTs outstanding who have already received their guaranteed processor service.

IRARMCL0 – Processor Load Balancing User Swap Processing.
Compute user processor usage profile at QSCECMP SYSEVENT.

IRARMCL1 – Processor Utilization Monitoring.
Compute processor utilization variables for processor load balancing and resource management algorithms.

IRARMCL3 – Processor Load Balancing User Swap Evaluation.
Produce a numerical recommendation value that reflects the desirability of swapping a user based on processor utilization.

System Resources Manager (SRM) (continued)

CHAP – IRARMCPM Internal Chapping Subroutine.
Search queue for APG users with changed dispatching priorities. Put them in a list and call ASCBCHAP.

CPLRVSWF – IRARMCPM Internal Wait Factor Computation Subroutine.
Compute system wait factor for CPU load balancing recommendation value.

CPUWAIT – IRARMCPM Internal Wait Time and Processor Utilization Compute Subroutine.

Compute accumulated system wait time total for all processors and compute recent processor utilization.

CPUTLCK – IRARMCPM Internal Processor Utilization Checking Routine.
Ensure that the computed processor utilization percentage falls between 0 and 100 percent. If 100 percent and lowest priority user has not been dispatched, set to 101 percent.

NEWDP – IRARMCPM Internal APG Computation Routine.
Compute mean time to wait and a new dispatching priority for the APG user.

IRARMIOM – I/O Management.

IRARMIL0 – I/O Load Balancing User I/O Monitoring.
Compute I/O usage profile for all swappable problem-state users.

IRARMIL1 – I/O Load Balancing Logical Channel Utilization Monitoring.
Compute channel utilization values for I/O load balancing, page replacement algorithms, and the device allocation SYSEVENT.

IRARMIL3 – I/O Load Balancing User Swap Evaluation.
Compute numerical recommendation value that reflects desirability of swapping a user based on logical channel utilization.

IRARMIL4 – I/O Load Balancing IMCB Deletion Routine.
At the end of the user job step, clean up the control blocks used in monitoring a heavy I/O user.

LCHUSE – Internal I/O Subroutine.
Compute logical channel utilization, request rate, and I/O load balancing recommendation value computation factor.

IRARMRMR – Resource Manager

IRARMRM1 – Resource Monitor Periodic Monitoring.
Accumulate, at periodic sample intervals, several system resource contention indicators and the number of ready users for each domain.

IRARMRM2 – Resource Monitor MPL Adjustment Processing.
Compute the average system resource utilization and determine if the system MPL should be raised or lowered.

IRARMCTL – SRM Control Algorithms.

IRARMCTL – Mainline Control Processing.
Transfer to deferred user action processing (IRARMCEN) and then to the algorithm request routine (IRARMCEL).

IRARMCEN – Deferred User Action Processing.
Examine the OUCBACN field of the OUCBs on the action queue and routes control to all routines whose request bits have been set in that field. Dequeue each OUCB after its indicated actions have been performed.

IRARMCEL – Algorithm Request Routine.
Examines the RMCTALR and RMCTALA fields in the RMCT. Routes control (via IRARMCRT) to each algorithm whose request bit has been set in either of the two fields. Reset the individual request bit after each algorithm completes.

IRARMCET – Periodic Entry Point Scheduler.
Accept timer interrupts, schedule the algorithms currently due for execution, and requeue the SRM timer element to permit interrupts again when the next algorithm is due for execution.

IRARMCED – SRB-Dispatched Original Entry Processor.
Receive control under an SRB scheduled by the dispatcher and set up an entry to the mainline of SRM (IRARMCEN) by invoking SYSEVENT 48.

IRARMCQT – Periodically-Invoked Entry Point Rescheduler.
Accept a request to reschedule the execution of a periodically-invoked algorithm, and requeue the corresponding RMEP block on the timed entry queue.

IRARMCRD – SRB Scheduling Routine.
Accept a request to schedule the SRM SRB which, if available, is scheduled to obtain the SRM lock.

System Resources Manager (SRM) (continued)

IRARMCRL – Algorithm Scheduling Routine.
Accept requests for an algorithm to be run. Turn on the bit in the RMCTALA or RMCTALR associated with the algorithm.

IRARMCRN – Action Request Routine.
Accept requests for an action requiring the SRM lock. If the SRM lock is held, control is given to the action immediately via a routing routine. If the SRM lock is not held, the bit is set in the OUCBACN field of the OUCB associated with the requesting user, to indicate that the action requested is deferred.

IRARMCRT – Entry Point Table Scanner.
Accept an invocation bit pattern and an entry point table address. Compare the bit pattern to invocation flags in the entry point table entries. When a match is found, invoke the routine identified by the entry point.

IRARMCRY – User Swap Request Receiving Routine.
Accept a request for a user swap and check to see if such a swap is already in progress. Route control to IRARMCSO or IRARMCSI if a swap is not in progress and the SRM lock is held.

IRARMCSI – User Swap-In Request.
Accept a swap-in request, allocate an OUXB for the user, and initiate the swap-in.

IRARMCSO – User Swap-Out Request.
Accept a swap-out request and post the region control task's quiesce routine to initiate the swap-out.

IRARMRPS – OUCB Repositioning Routine.
Dequeue an OUCB and requeue it at the end of the queue specified in its OUCBQFL field.

IRARMWMY – Periodic Entry Point Requeuing Routine.
Requeue all of the members on the timed algorithm queue and adjust all the time-due fields.

IRARMCAP – Swap Analysis Algorithm.
Attempt to keep the multiprogramming level (MPL) at its target level in each domain by performing user swaps.

IRARMCPI – Select Swap-In Candidate Subroutine.
Scan the OUT queue for the user in a particular domain with the highest recommendation value.

IRARMCPO – Select Swap-Out Candidate Subroutine.
Scan the IN queue for the user in a particular domain with the lowest recommendation value.

IRARMCVL – User Swap Evaluation Routine.
Compute a numerical value representing the recommendation for a user to be swapped in. This recommendation value is the sum of the user's workload level and the recommendations of the I/O and processor resource managers.

IRARMWAR – Workload Activity Recording

IRARMWR1 – Workload Activity Recording Initialization Subroutine.
Constructs and initializes the workload activity measurement table (WAMT) in the buffer (storage from SQA obtained by MF/1 and input with SYSEVENT 45).

IRARMWR2 – Workload Activity Recording WAMT Initialization Subroutine.
Build the WAMT in a format suitable for updating by the SRM.

IRARMWR3 – SRM Workload Activity Recording Data Collection Subroutine.
Move the contents of the WAMT into a collection buffer capable of containing the data. (Note that the buffer is obtained by MF/1 from LSQA, storage key 0, and must be fixed in storage). If the IPS has not been changed, add to the collected data the transaction data for the current in-storage interval for each in-storage address space with an active transaction, re-initialize the data collection buffer for the next collection interval, and calculate the workload level for each performance group period that contains transaction data.

IRARMWR4 – SRM Workload Activity Recording Transaction Data Update Subroutine.
Add the service and transaction active time to the appropriate WAMT performance group period accumulator in the data collection buffer.

IRARMWR5 – SRM Workload Activity Recording Workload Level Calculation Subroutine.
Calculate the workload level for each WAMT performance group period entry in which a transaction has been accumulated during the data collection interval. Note that for those WAMT entries in which the service rate calculated can be associated with multiple workload levels, or is zero (even through at least one transaction has been active during the data collection interval), the negative value of the workload level is calculated to indicate an estimated value to MF/1.

IRARMWR6 – SRM Workload Activity Recording Transaction End Update Subroutine.
Add the transaction elapsed time to the appropriate WAMT performance group period accumulator and count the number of transactions that terminated during the current data collection interval.

IRARMWR7 – SRM Workload Activity Recording WAMT Entry Determination Subroutine.
Obtain addressability to the WAMT performance group period entry used to accumulate user transaction information.

IRARMWR8 – SRM Workload Activity Recording.
Terminate workload activity data collection whenever an IPS change occurs.

IRARMWLM – SRM Workload Manager

IRARMWM1 – Workload Manager Service Calculator Routine.
Calculate the amount of service provided to a user since the beginning of the current workload manager measurement for that user. Service is calculated using the following equation:

Service = (MP)/K)+(CT/K)+EI where:

T = the TCB processor time elapsed for the current interval.

K = the time required to execute 10,000 instructions. (dependent on the processor model).

M = the MSO service coefficient scaled by 1/50.

P = the number of page-seconds used by the user.

C = the processor service coefficient.

E = the Excp count for this interval.

I = the I/O service coefficient.

This routine calculates each of the three service factors and the total service for the user for the interval.

IRARMWM2 – Swappable User Evaluation Routine.
Scan the in-storage queue and the out-of-storage-but-ready queue, and evaluate each swappable user, assigning each his current workload level.

IRARMWM3 – Individual User Evaluation Routine.
Evaluate a swappable user on the in queue or the out queue, assigning a current workload level.

IRARMWM4 – Workload Manager Workload Level Calculator Subroutine.
Accept a service rate and a performance objective, and calculate the corresponding workload level.

IRARMWM5 – Workload Manager Update Performance Group Period Subroutine.
Test whether a user has accumulated enough service/time to be assigned to a new performance group period. If so, adjust the pointers that indicate the performance group period, the performance objective, APG priority, and the domain applicable to the transaction current for the user. Note that the frequency (resolution) at which the test for period-end is made depends on how often IRARMWM5 is called for any given user.

IRARMWM7 – WLM Recommendation Calculation Routine.
Calculate a workload manager recommendation value for a user, based on the service that was received and on the performance objective currently associated with the user. Users who have not yet received an amount of service equal to their interval service value (ISV) specification while in storage are given a recommendation value boost. The boost gives preferential treatment to users in their ISV as compared to users not in their ISV, or to users between job steps.

IRARMHIT – Workload Manager User Ready SYSEVENT Swap-In Scheduling Routine.
Reposition the now-ready user from the wait queue to the out queue. Receive control as the result of a decision to apply swap-in processing to a now-ready user.

IRARMWMI – Workload Manager In Storage Interval Change Subroutine.
Update the transaction accumulators with the service and the time received by the user during the preceding in-storage interval.

IRARMWMJ – Routine to Determine the Scope of Applicability of Analysis to a User.
Examine the current swap status and the performance specification for a user. Indicate if the resource manager algorithms are applicable to this user.

IRARMWMK – WLM Dontswap/Okswap User Analysis Routine.
Calculate the current service and ensure that the user is in the correct performance group period. Set applicable algorithm indicators based on the new swap status of the user.

System Resources Manager (SRM) (continued)

IRARMWMN – Workload Manager Transaction Start Routine.
This routine receives control as the result of a SYSEVENT that has been defined by the workload manager to signify that a new transaction should be started for that user. If the user is not in storage, a flag is set to cause the IRARMWMN routine to be reentered during the swap-in of the user. Otherwise, any existing transaction is stopped by calling IRARMWMO, and the user transaction fields are reset to reflect the new transaction field being started.

IRARMWMO – Workload Manager Transaction Stop Routine.
This routine receives control as the result of a SYSEVENT that has been specified by the workload manager as defining the end of any current user transaction. If a new transaction is to be created for the user, IRARMWMO indicates the end of the current transaction. If the next user event is known, IRARMWMO leaves the transaction accumulated values for later resumption of the transaction. In any case, IRARMWMO causes the preceding time and service to be properly recorded for the current transaction.

IRARMWMQ – Workload Manager Quiesce Completed SYSEVENT Processing Routine.
This routine receives control when a user has stopped executing and is being swapped out so that the workload manager can record the service given that user while he was in storage. The workload manager determines if a user event caused the swap-out, and flags the user to indicate whether previous service is to be considered when the user is next swapped in.

IRARMWMR – Workload Manager Restore Completed SYSEVENT Processing Routine.
This routine receives control when a user has been swapped in and is ready to begin executing. The workload manager sets up the fields used to calculate the service rate received by the user during the forthcoming in-storage residency period.

IRARMSET – Set to New IPS Non-Resident Action Routine.
Replace the internal IPS currently in use by the SRM with a new IPS. All references to the old IPS in the SRM's control blocks are resolved with offsets or addresses in the new one.

IEEMB812 – Set IPS Processor.

- IEEMB812 – Open PARMLIB. Processes the IPS parameter of the SET command.
- IRARMRDR – Obtain a buffer and reads records from PARMLIB.
- IRARMWTR – Write a message to system log.

IRARMIPS – SRM List Processor.

- IRARMIPS – Scan the IPS List in the SYS1.PARMLIB member, and if valid, build control blocks containing the IPS information.
- IRARMFRE – Free the obsolete IPS tables.
- IRARMOPT – Scan the IEAOPTxx member of PARMLIB.

IEAVNP10 – SRM Initialization.

- IEAVNP10 – 1. Initialize constants in SRM tables.*
  2. Initialize sysgened address spaces for the SRM.
  3. Process the APG, OPT, and IPS system parameters.
- IRARMRDR – Obtain a buffer and read a record from SYS1.PARMLIB.

IEEDISPD – Display Domain Processor.
Write a console display of entries in the domain descriptor table to a target console.

IEE8603D – SETDOMAIN Command Processor.
Process the SETDMN command by altering the domain descriptor table.

SRM MODULES / SRM ENTRY POINTS	IEAVNPIO	IEEDISPD	IEEMB812	IEE8603D	IRARMCNS	IRARMCPM	IRARMCTL	IRARMERR	IRARMEVT	IRARMINT	IRARMIOM	IRARMIPS	IRARMMSG	IRARMRMR	IRARMSET	IRARMSRV	IRARMSTM	IRARMWAR	IRARMWLM
CHAP						X													
CPLRVSWF						X													
CPUTLCK						X													
CPUWAIT						X													
IGC095										X									
IRARMAP1						X													
IRARMASM																	X		
IRARMCAP							X												
IRARMCED							X												
IRARMCEL							X												
IRARMCEN							X												
IRARMCET							X												
IRARMCL0						X													
IRARMCL1						X													
IRARMCL3						X													
IRARMCPL							X												
IRARMCPO							X												
IRARMCQT							X												
IRARMCRD							X												
IRARMCRL							X												
IRARMCRN							X												
IRARMCRT							X												
IRARMCRY							X												
IRARMCSI							X												
IRARMCSO							X												
IRARMCVL							X												
IRARMDEL									X										
IRARMEQ1						X													
IRARMFRE												X							
IRARMHIT																			X
IRARMI00										X									
IRARMI01										X									
IRARMI02																X			
IRARMI03																X			
IRARMI04																X			
IRARMI05																X			
IRARMI06																X			
IRARMI07																X			
IRARMI09																X			
IRARMI10										X									
IRARMI48										X									
IRARMIL0											X								
IRARMIL1											X								
IRARMIL3											X								
IRARMIL4											X								
IRARMIPS									X										
IRARMMS2																	X		
IRARMMS6																	X		
IRARMNQP																X			

Figure 5-31. SRM Module/Entry Point Cross Reference (Part 1 of 2)

SRM MODULES / SRM ENTRY POINTS	IEAVNPIO	IEEDISPD	IEEMB812	IEE8603D	IRARMCNS	IRARMCPM	IRARMCTL	IRARMERR	IRARMEVT	IRARMINT	IRARMIOM	IRARMIPS	IRARMMSG	IRARMRMR	IRARMSET	IRARMSRV	IRARMSTM	IRARMWAR	IRARMWLM
IRARMOPT												X							
IRARMPR1																	X		
IRARMPR5																	X		
IRARMRDR	X		X																
IRARMRM1														X					
IRARMRM2														X					
IRARMRPS							X												
IRARMRR1								X											
IRARMRR2								X											
IRARMR16																X			
IRARMSQA																	X		
IRARMUXB									X										
IRARMWM1																			X
IRARMWM2																			X
IRARMWM3																			X
IRARMWM4																			X
IRARMWM5																			X
IRARMWM7																			X
IRARMWMI																			X
IRARMWMJ																			X
IRARMWMK																			X
IRARMWMN																			X
IRARMWMO																			X
IRARMWMQ																			X
IRARMWMR																			X
IRARMWMY							X												
IRARMWR1																		X	
IRARMWR2																		X	
IRARMWR3																		X	
IRARMWR4																		X	
IRARMWR5																		X	
IRARMWR6																		X	
IRARMWR7																		X	
IRARMWR8																		X	
IRARMWTR	X		X																
IRARMXVT									X										
IRARMXTL							X												
LCHUSE											X								
NEWDP						X													
RMRR1CKQ								X											
RMRR2GST								X											
RMRR2INT								X											
RMRR2PER								X											
RMRR2REQ								X											
RMRR2RTY								X											
RMRR2SPR								X											
RMRR2VFB								X											
RMRR2VLD								X											
STEAL																	X		

Figure 5-31. SRM Module/Entry Point Cross Reference (Part 2 of 2)

This chapter outlines the important aspects of VTAM problem analysis. It is important that the problem solver have some understanding of how VTAM works.

The following publications provide important VTAM structure, logic, control block format, and debugging information:

- *OS/VS2 VTAM Logic*
- *OS/VS2 System Programming Library: VTAM*
- *OS/VS2 VTAM Data Areas*
- *OS/VS2 MVS VTAM Debugging Guide*

VTAM is a subsystem in itself. For VTAM problem determination, it is especially important to understand how work progresses through VTAM via its internal dispatching mechanism, process scheduling services (PSS).

Some of the VTAM concepts that are discussed in this section are:

- Process scheduling services (PSS)
- VTAM's Relationship With MVS
- Processing Work Through VTAM (PABs, FMCBs)
- VTAM Locking
- VTAM Recovery/Termination
- VTAM Debugging

VTAM's Relationship with MVS

VTAM has its own address space to manage the network control program (NCP) network. Under VTAM's main task, the following services are performed:

- VTAM initiation and termination.
- VARY, DISPLAY, and MODIFY network operator commands.
- An access method control block (ACB) is opened to VTAM so that VTAM services can be used to communicate with an NCP.

The VARY processor obtains and releases IBM 3704/3705 communications controllers through the system dynamic allocation services. A waiting subtask is posted to build an NCP resource definition table (RDT) which provides a table definition of the network. Another subtask is attached to actually load the 370x NCP. This procedure allows multiple 370x's to be activated concurrently.

If TOLTEP and NETSOL are selected, each has its own subtasks operating in VTAM's address space. Each is also connected to VTAM with an OPEN ACB.

Because the VTAM address space owns the 370x, IOS schedules global SRBs to this address space for POST STATUS processing. Normally, however, VTAM uses a disable interrupt exit (DIE) to run its channel end appendage. The DIE schedules SRBs (physically located in the I/O buffer) into the application program's address space to run the posting process. POST STATUS is used only to handle error situations or when RNIO is being traced with GTF active.

VTAM operates in the application program's address space when a service is requested by the application program. Local SRBs are used for all VTAM I/O processing to terminals or logical units. Other VTAM services such as OPNDST/CLSDST are run under an IRB from the task that opened the VTAM ACB. VTAM exits, ACB and request parameter list (RPL), are given control when VTAM issues a SYNCH under the IRB. This means that a VTAM exit runs as a parameter request block (PRB) under the task that opened the VTAM ACB. VTAM macros (for I/O or other services) can be issued from these exits; however, if the SYN option is used on the macro, a serialization bottleneck can result.

As seen from this explanation, VTAM's address space is not used for normal I/O activity. In analyzing VTAM problems, do not be concerned if several tasks are waiting in VTAM's address space. These tasks are the operator control, NCP communication, and initiation/termination tasks, and are normally waiting in VTAM's address space.

Processing Work Through VTAM

Following is an explanation of the dispatching mechanism and the associated key control blocks that the problem solver should understand.

VTAM satisfies an application program's request by executing a series of processes. Examples of processes are control layer and TPIOS; each process is a discrete piece of work.

Each process is represented by a process anchor block (PAB) which is four words long and serves as a serialization mechanism for a resource. See Figure 5-32. A PAB always resides within some larger control block called a major control block such as an FMCB or an ACDEB. A process is always executed for a particular terminal, logical unit, or option as defined by the major control block PAB.

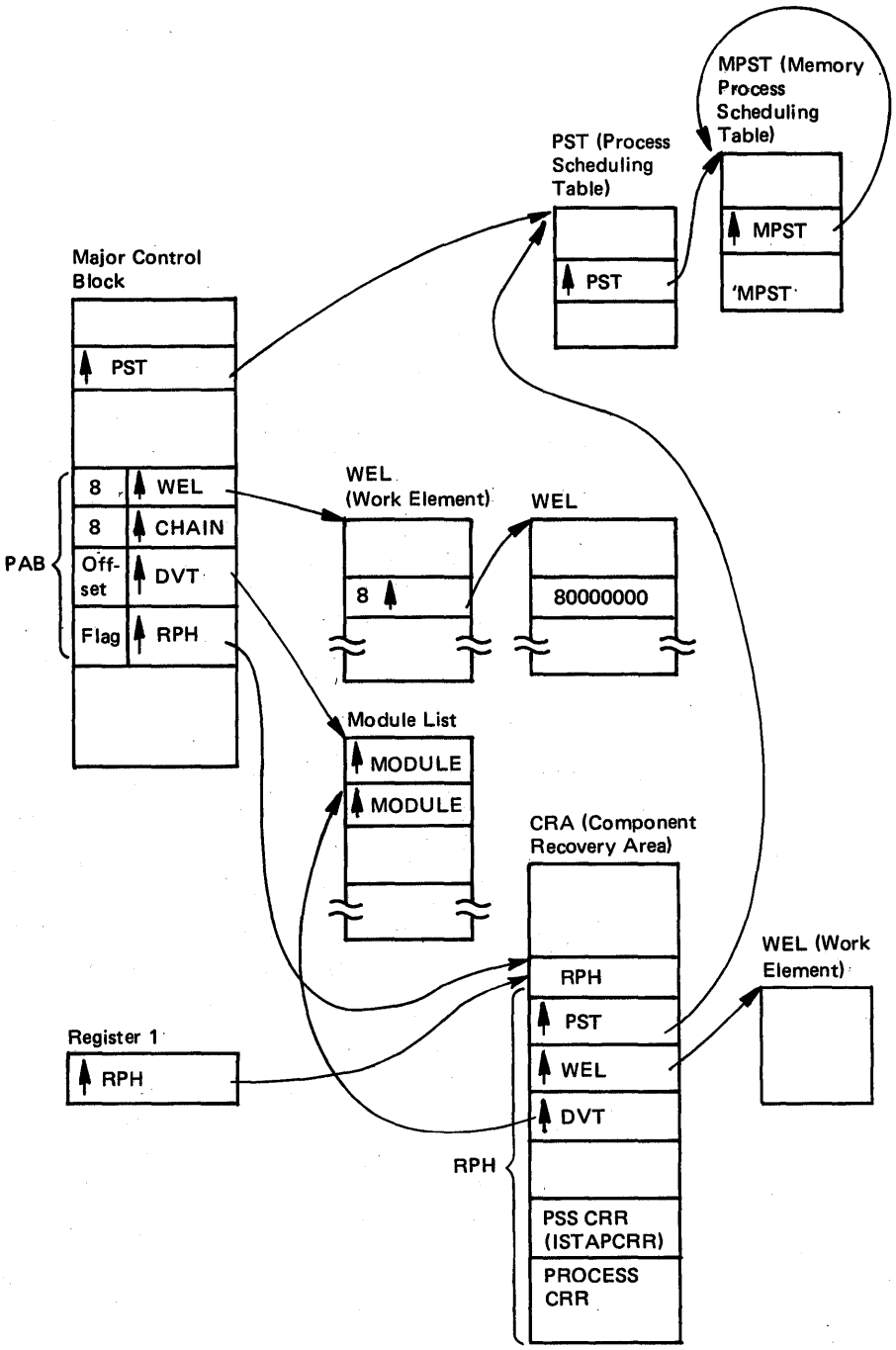


Figure 5-32. VTAM Control Block Structure

The first word of a PAB contains a work element pointer. A work element is a parameter list for the process. A request parameter list (RPL) and a logical channel program block (LCPB) are examples of work elements. The high-order bit (byte 0,X'80') of this first word is a gate bit which indicates that a work element has been queued to the PAB. The gate bit serves as a serialization mechanism; as more work elements are queued to the PAB, the gate bit prevents rescheduling of the PAB until it can handle the work. The gate bit is needed to prevent double scheduling of the PAB, because for many VTAM processes the process scheduling service (PSS) dequeues the work element before it gives the process control.

The TPQUE macro is always used to queue work elements. This in-line macro checks the gate bit to determine if scheduling is required and, if so, executes an inner macro, TPSCHED.

The second word of the PAB is the PAB chain field. As a general convention, PSS and its macros (for example, TPQUE) use the second word of any control block as a chain field. The end of the chain is indicated by X'80000000' in the chain field. The PAB chain field is used to chain the PAB to some queue, for instance, a dispatching queue. The chain field's high-order bit is a gate bit. The gate bit indicates that the PAB has been scheduled for dispatching.

Following are the three ways to schedule a PAB for dispatch:

- While running under a VTAM process, queue the PAB to the PABQ in the request parameter header (RPH). The PAB will be dispatched when the current process completes.
- If not running as a VTAM process, queue the PAB to the process scheduling table (PST) for task-related work, or to the memory process scheduling table (MPST) for address-space related or cross-address-space related work.
- DIRECT scheduling causes an SRB, with the PAB address as a parameter, to be scheduled to a special PSS entry point. TPIOS uses this method to initiate inbound processing from the DIE.

Note that if the PAB chain gate is off while the work element gate is on, the PAB is probably suspended. A TPSCHED macro is required to reactivate the process.

The third word in the PAB contains the PAB offset and the destination vector table (DVT) pointer.

The PAB offset is used to locate the beginning of the major control block. It is necessary to locate the beginning of the major control block because there is a PSS convention that uses the third word of the major control block as a pointer to the process scheduling table (PST). The fourth word of the PST points to the memory process scheduling table (MPST), which is related to a particular address space. The PAB offset then provides a means to identify task and address space relationships for a given PAB. As a rule, the PST is used to schedule processes to run under the IRB of a particular task, while the MPST is used for scheduling a local SRB into the address space.

The PAB DVT pointer points to the beginning of a module list, that is, a list of addresses that are entry points to the modules to be given control during the process. Because the DVT defines a whole process that is to be executed, many PABs will have DVT pointers to the same DVT. The next entry in the DVT to be given control is kept in the request parameter header (RPH); the RPH is updated each time the TPESC macro is used to pass control to the next module in the DVT.

The fourth and last word of the PAB contains a byte of flags and a pointer to the request parameter header (RPH). The flag byte contains scheduling indicators for PSS and a bit to indicate whether or not PSS should dequeue the work element for the process. The RPH pointer is set by PSS when the PAB is to be dispatched and is reset to zero when the process completes. Register 1 always points to the RPH when the process is given control. All of the information relating to the process is stored in the RPH. This information includes such items as pointers to the work element (RPHWEA), the PST (RPHTSKID), a resume address (RPHRESMA) and a register save area (RPHWORK) if the process is suspended, and back pointers to the PAB (RPHMAJCB). The RPH resides within the component recovery area (CRA).

VTAM Function Management Control Block (FMCB)

The function management control block (FMCB) is the primary control block used in controlling I/O processing between an application program and a destination node (terminal, component, logical unit (LU), etc). This block usually contains the most information when a problem develops in the I/O processing to a particular node. The FMCB is created at OPNDST time; at least one FMCB exists for each open connection. All FMCBs for an application are chained together (at offset X'4') out of the application's ACDEB (at offset X'40'). In addition to the application FMCBs, VTAM maintains FMCBs for such things as dial-in lines and cluster control units. For logical units, there is also an SSCP FMCB chained from VTAM's ACDEB that is used for network control.

The FMCB contains the PABs that control processing through control layer (inbound/outbound) and TPIOS (outbound). Although there is a PAB in the FMCB for TPIOS inbound, the PAB in the DNCB is normally used to control it. In addition to the PABs, the FMCB contains many flags and indicators and some queue headers. These flags and headers are described in the *OS/VS2 Data Areas* (microfiche).

The wait queues at offset X'110' and X'114' in the FMCB are important in debugging. These fields are used to queue logical channel program blocks (LCPBs) that have had channel programs built and queued to be shipped out to the 370x or local 3270. The LCPBs are dequeued from the wait queues when the requested operation completes. Expect to see read-type operations queued to the wait queue because these operations do not complete until data is entered and received from the associated terminal. However, if write or control type operations are not completing, investigate the situation further.

VTAM Operating Characteristics

The following topics describe characteristics of VTAM's operating environment.

Module Naming Convention

Each VTAM module name indicates the type of processing that it performs. Following are the major VTAM module naming groups and the processes associated with them:

Module Group	*Process*
ISTAICxx	Application program interface
ISTAPCxx	Process scheduling services (PSS – VTAM's dispatching mechanism)
ISTDCCxx	Basic and common control layer
ISTRCCxx	Record format control layer
ISTZxxxx	TPIOS
ISTORFxx	Storage management
ISTOCCxx	OPNDST, CLSDST, OPEN, CLOSE
ISTINxxx	SSCP (VARY, DISPLAY)
ISTRAMxx	Task termination and address space termination resource manager
ISTSDCxx	SYSDEF

Address Space Usage

VTAM's modules reside in the nucleus, the VTAM address space private area, and the LPA. The nucleus contains the attention handling routine and type 1 SVC routine. The private area contains modules for initialization, termination, initial command processing, SYSDEF, and NETSOL. The LPA contains all of the other VTAM modules.

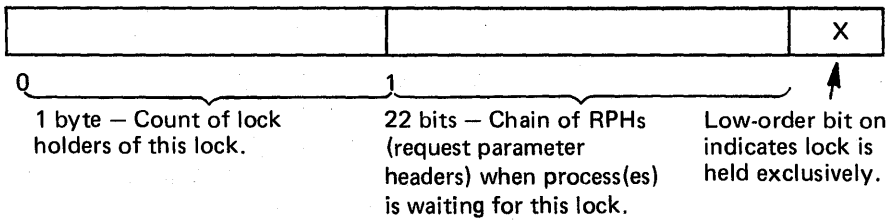
Most of the VTAM control blocks are located in the CSA. The data buffers as well as the majority of the control blocks occupy the 11 buffer pools that are allocated at VTAM initialization with the CSA.

Locking

Because VTAM uses a number of SRBs and TCBs, it is important to serialize VTAM's internal use of shared resources (that is, to prevent simultaneous update of a control block by two different processes). The VTAM locking structure accomplishes this serialization. The VTAM locking structure is an internal VTAM function not visible to the user or MVS. VTAM's locking structure is totally independent of the MVS locking structure.

In storage, the locks exist as full words in various areas of the VTAM control blocks. Following is the organization of the lockword:

Lockword



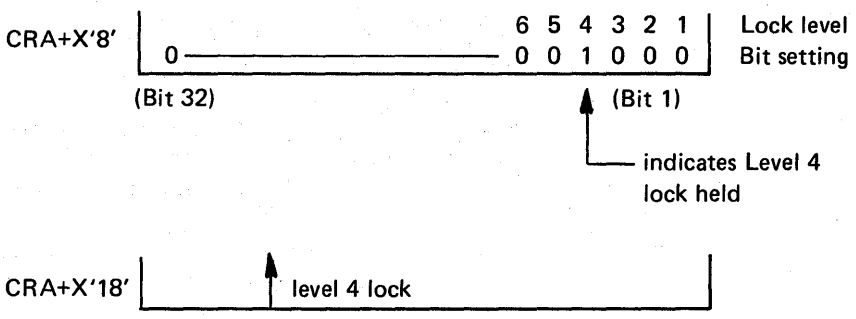
Each lock is defined as being of a certain lock level. This allows a lock to be maintained according to a predetermined hierarchy. These lock levels are usually not of significance to the debugger except that he needs to know that they exist so he can interpret the lock level bits in the component recovery area (CRA) as described below.

Locks are obtained and released by using VTAM internal macro instructions. The access to the locks is controlled as follows:

- A shared request for a lock that is free or held as shared (with no outstanding exclusive requests) is honored immediately.
- An exclusive request for a lock that is held as shared is queued until all current shared requests are released.
- Any request for a lock that is held as exclusive, or has an exclusive request outstanding, is queued until the exclusive use is complete.

The locks held by a process are indicated by the lock level bits in the CRA (at offset X'8'). The pointers to the various locks are located at X'C' through X'30' of the CRA. The pointers to the locks are filled in when a lock request is made; therefore, only the locks currently held have valid pointers. Locks are held only for the duration of a VTAM process; all locks must have been released when a process exits.

Examples of a locking situation:



If the CRA lock accounting word appeared as above, it would mean that a level 4 lock was held by the process currently active. Offset X'18' (level 4 lock pointer) of the CRA contains the pointer to the lock in question. Refer to *OS/VS2 Data Areas* (microfiche) for details..

RPHs that are waiting for a lock will be queued onto that lock. Multiple RPHs waiting for the same lock will be chained together. This relationship is shown in Figure 5-33.

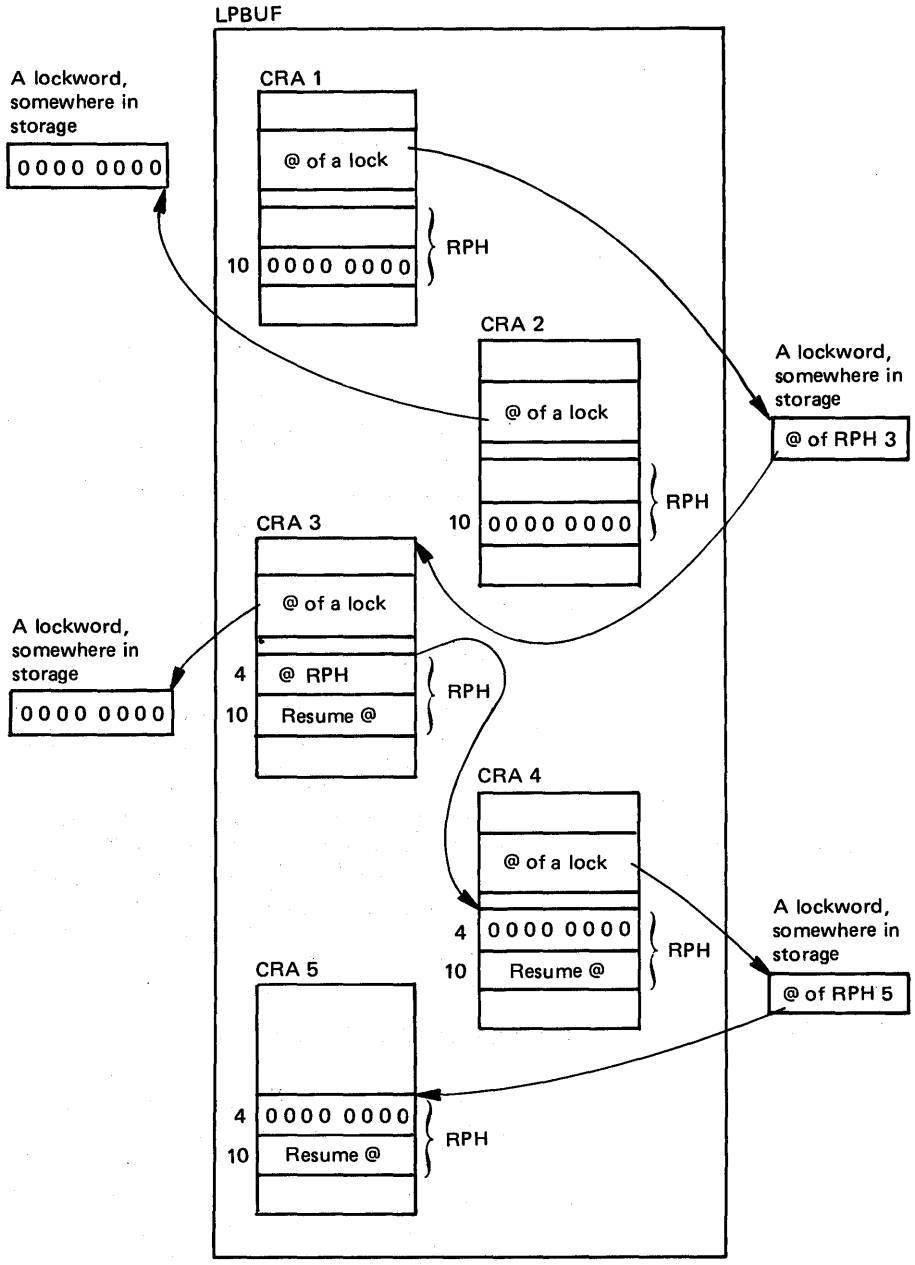
Summary of VTAM Locking: The main concern of the debugger regarding locks is that a process can be forced to wait because it cannot obtain a lock. The lock is unavailable because it is held by some other process. This situation is reflected by an active CRA with a resume address that points to code that follows a lock request. Lay out the CRAs, etc., as shown in Figure 5-33 and investigate those processes that are waiting, determine what lock is being requested, locate the current lock holder, and determine why the lock has not been released.

VTAM Recovery/Termination

VTAM recovery/termination is accomplished by means of STAE, ESTAE, FRR, and resource manager routines.

The exact recovery action attempted by VTAM depends on conditions at the time of the errors. However, for debugging purposes, the basic functions of VTAM are the following:

- To record the SDWA in SYS1.LOGREC
- To take an SDUMP.
- To terminate the application program or VTAM. (Note, if the error occurs in the VTAM address space, VTAM generally attempts to simulate a normal shutdown.)



Non-waiting RPH (CRA 1) holds the lock that RPH 3 (CRA 3) and RPH 4 (CRA 4) are waiting for. Non-waiting RPH 2 (CRA 2) holds a lock no RPHs are waiting for. Waiting RPH 3 (CRA 3) holds a lock that no RPH is waiting for. Waiting RPH 4 (CRA 4) holds a lock that RPH 5 (CRA 5) is waiting for.

Figure 5-33. Several RPHs Waiting for the Same Lock

The termination of VTAM or VTAM applications causes the VTAM resource managers to get control. The resource manager routines clean up the VTAM resources allocated to the terminating task or address space.

VTAM recovery/termination functions affect debugging in the following ways:

1. A dump and SDWA in SYS1 LOGREC are provided for the error condition. If the error was in a VTAM application address space, VTAM and other VTAM applications continue to run. This allows you to debug certain problems without having a major impact on the installation's operation.

2. Subsequent errors can occur in termination, in VTAM, or in other VTAM applications as a result of the original error that was undetected. If this is the case, a dump can be very difficult to understand because cleanup was attempted or performed on behalf of the original error. Always be aware of any problems that have occurred prior to the particular problem being diagnosed. To detect these previous problems, inspect the in-storage LOGREC buffer and SYS1.LOGREC.

In addition to the major recovery action described above, there are other recovery actions:

- A failure during command interpretation (but not during command execution) results in the loss of the current operator command, but continued availability of the operator control function.

- Failure during various SSCP functions can result in the immediate termination of VTAM without simulating a normal shutdown.

- Failure in the storage management services (SMS) modules, ISTORFBA and ISTORFBD, results in a failure of the storage request, but does not cause termination. The module requesting SMS service is informed of this action by return codes.

- Authorized path entry/exit errors are retried or the RPL is posted with an error indication.

VTAM Debugging

Because VTAM is a large component that interacts with other components and application programs, when you debug VTAM you must look at a number of factors besides the storage dump. Begin the debugging process by considering the operating environment and all the conditions that could have led to the suspected error.

Following are some items you should look at when attempting to solve VTAM problems:

- Console sheet
- GTF traces (especially for VTAM I/O activity)
- SYS1.LOGREC entries for program checks or MDR records
- NCP generation listing
- PTF level of the system

Waits

VTAM waits can happen to the following groups:

- Entire VTAM component and all VTAM applications
- One or more applications only
- VTAM network operator commands only (possibly only VARY)
- One or more terminals only

Also, a wait can occur when VTAM will not halt.

VTAM process scheduling services (PSS) routines control the flow of work through VTAM by performing an internal VTAM dispatching function. In debugging, the most important control block in determining the status of the dispatching activity and the wait states is the request parameter header (RPH), which is located within the component recovery area (CRA).

If a VTAM internal process is waiting for a VTAM lock, for storage to become available, or for another process to complete, this condition is reflected by an active CRA. CRAs are found in the LPBUF buffer pool. (See Figure "How to Locate the CRA" in *OS/VS2 MVS VTAM Debugging Guide.*) This pool is located as shown in Figure 5-34. The RPH is located X'34' bytes into the CRA; it can be recognized by the string X'016C' in the first halfword. Offset X'10' into the RPH is the RPHRESMA field; this field contains zeros if the RPH is not waiting or a resume address if it is waiting.

Once you find a waiting RPH, the best way to determine why it is waiting is to find the module at the address in the resume address field, and then look at the module listing. Unless the wait is for a lack of buffers (which can be resolved by increasing the number of buffers), further analysis is necessary to determine why the process is not being posted or why a lock is not being freed.

RPHs waiting for a VTAM lock are queued onto that lock. Multiple RPHs waiting for the same lock are chained together (as shown in Figure 5-33). If a process holds any locks, the lock level bit at offset 8 in the CRA indicate the level of the lock(s) being held. Pointers to the various locks are located at offsets X'C' through X'30' of the CRA. Note that although all these pointers can contain addresses, only the pointers to the locks held or requested during the dispatching of the current process are valid.

A VTAM internal process can often be waiting for storage. VTAM routines obtain and release buffers by using VTAM internal macro instructions. These macro instructions branch to the VTAM storage management modules that control the buffer pools allocated at VTAM start time. Because the number of buffer pools is specified at VTAM initialization and is constant, it is possible to encounter a shortage condition in unusual situations. See the section on tuning VTAM in the *OS/VS2 System Programming Library: VTAM* for information on specifying the proper storage pool values, the threshold value effect, and slowdown processing.

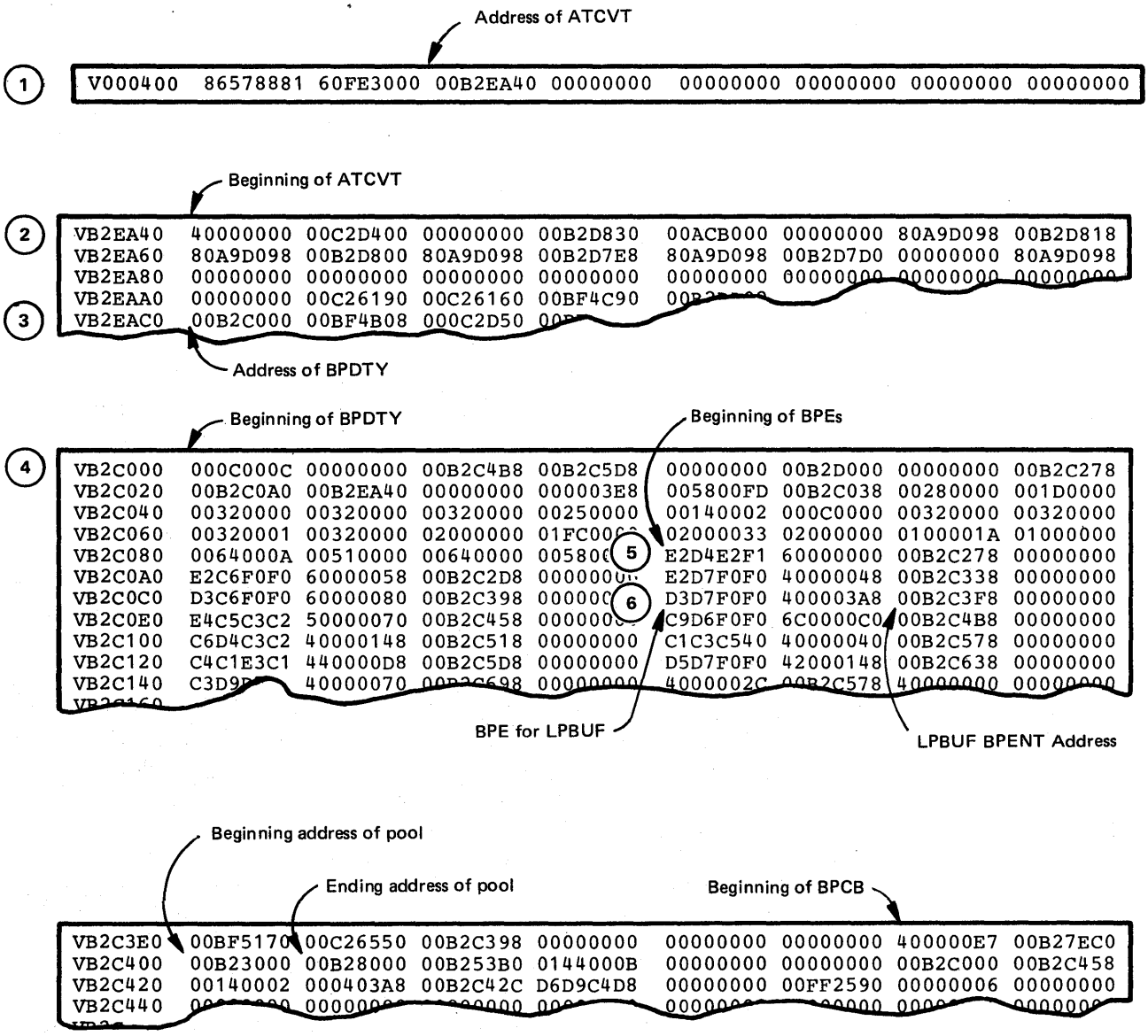
To determine if a buffer pool is in a slowdown state, do the following:

- Locate the buffer pool control blocks (BPCBs) as shown in Figure 5-34.
- Look at offset X'10' of each BPCB. If the X'40' bit is on, the buffer pool is in a slowdown state. (The BPCBs are located in contiguous storage and can thus be scanned quickly.)

If a dump has been taken because of a wait-type problem in VTAM, and the dump shows a buffer pool in slowdown, you can usually conclude that a buffer shortage has caused the wait problem. If you increase the number of buffers in the appropriate pool, this usually eliminates the problem. However, the problem should be investigated further to determine if a VTAM logic error has caused the buffers to be wasted and thus depleted.

VTAM routines that request buffers can choose to wait or not to wait if there are not enough buffers available to fulfill the buffer request.

If a routine chooses to wait (and most do) when buffers are unavailable, the process is represented by an active CRA with a non-zero resume address. In addition to the slowdown bit being on in the BPCB, the RPH for the process is queued onto the BPCB. The queue headers in the BPCB are located at offsets X'18' and X'1C'; X'18' for queuing priority requests and X'1C' for queuing normal requests. Figure 5-35 represents the queuing of RPHs and Figure 5-34 shows how to find the BPCBs. Any address in either of the queue headers of a BPCB indicates a buffer problem with that pool. The RPHs queued from the BPCB have a resume address that points to code following the buffer request. Examine the routine in question to determine if an error in the code has caused the buffer problem or if the condition exists because the buffer specification was too small.



Notes:

1. Locate the address of the ATCVT (VTAM communication vector table) by going to absolute storage location X'408.'
2. Go to the specified address to locate the ATCVT.
3. Locate the address of the BPDTY (buffer pool directory) by indexing into the ATCVT a value of X'80.'
4. Go to the specified address to locate the BPDTY.
5. Locate the BPEs (buffer pool entries) by indexing into the BPDTY a value of X'90.'
6. The BPEs contain the name of the pool in the first 4 bytes — the length of the pool element in the second 2 bytes of the second word — and the address of the BPCB (buffer pool control block) in the third 4 bytes. Each BPE is X'10' bytes long. (Note that the example shows the LPBUF.)

Figure 5-34. Sample Storage Pool Dump

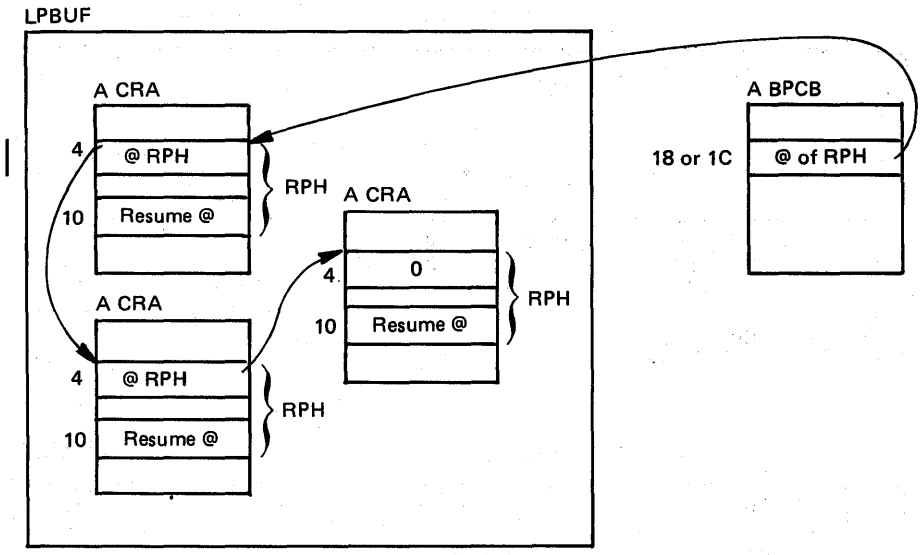


Figure 5-35. Queuing of RPHs While Waiting for Storage

If a routine chooses not to wait when buffers are unavailable, return codes notify the routine of the lack of buffers. There are no specific flags that are always set to indicate that the request was rejected. Therefore, you cannot easily determine if a particular routine requested buffers but did not get them. However, you can tell that there is a buffer problem because this is usually indicated by the slowdown bit being on in one of the BPCBs.

Usually there is an active CRA for problems that are described as VTAM waits. However, for some problems (for example, one or more terminals waiting), a dump might be obtained that has no active CRAs. The best place to start with a problem such as this one is to locate the FMCB/DNCB for the terminal(s) in question. The FMCB/DNCB control blocks contain the following:

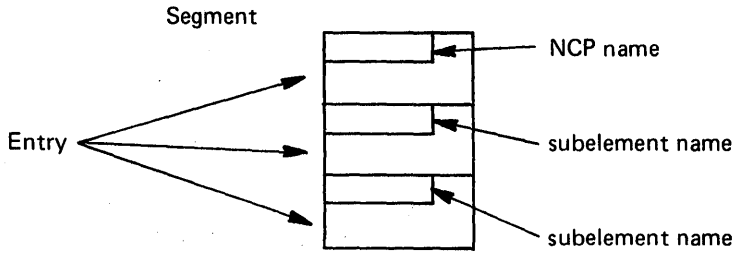
- various flags
- PABs to control the inbound and outbound request processing
- queues of outstanding requests

Investigate further any work elements found unprocessed on the PABs or queues of these control blocks.

To find the FMCB/DNCB for a particular node name, look at the following:

LOC X'408' = ↑ ATCVT
ACTCVT + X'C' = ↑ QAB
QAB + X'8' = ↑ first RDT segment
RDT + X'4C' = ↑ next RDT segment
⋮

RDTs are segmented tables and each segment contains information about a major node as defined in SYS1.VTAMLST. Each segment contains entries for the groups, lines, clusters, terminals, components, etc., for the major node, as shown below:



The node name is in the first eight bytes of each RDT entry. Stop chaining through the RDT entries when the major node name (that is, NCP or LBUILD) is found; then scan down the RDT entries of this major node until the node in question is found. Offset X'28' of the appropriate RDT entry points to the DNCB (offset X'28' from beginning of name). DNCB + X'10' points to the FMCB.

Another way to find the FMCB/DNCB for a process that is waiting (but has an active CRA and an RPH with a non-zero resume address) is to look at the module that is waiting and determine its register usage. Find the registers that were saved in the RPH + X'28'. See the module for the order in which the registers were saved.

Program Checks

The information generally available on VTAM program checks is in the SDWA and the SDUMP that the ESTAEs or FRRs provide. This information is used in the normal manner to determine the cause of the program check. However, there can be cases where more exact or timely information is required. This additional information might have to be obtained through the use of traces or traps. Traps at the entry point of the FRRs or ESTAEs (ISTAPC61 and ISTAPC62, in particular) are often useful.

Miscellaneous Hints On VTAM

1. VTAM waits can occur because of buffer depletion. Such a situation usually occurs just after VTAM is installed and before the actual buffer requirements are determined. Running GTF (with the USR trace option) at this time can be helpful because VTAM creates an SMS trace record whenever a storage request is queued. Because VARY ACTIVATE/INACTIVATE of an NCP puts the heaviest stress on the buffer pools, start GTF before an NCP is activated.

2. VTAM places warmstart copies of major nodes into the data set SYS1.VTAMOBJ the first time that a node is activated. These warmstart copies are used for subsequent activations of the node. If a node definition is changed in SYS1.VTAMLST, be sure to scratch the corresponding member in SYS1.VTAMOBJ to ensure that the new definition is used by VTAM. Also scratch the members of SYS1.VTAMOBJ after PTFs have been installed because some of the bit definitions might have changed.

3. Most VTAM control blocks are in the 11 VTAM buffer pools. By simply scanning the buffer pools and looking for unusual conditions you can often uncover many of the problems. Each buffer in the buffer pools is preceded by a two-word buffer header. The high-order bit of the first word indicates whether the buffer is allocated or available: on = allocated, off = available. The address portion of the first word points to the module that last released the buffer. The second word contains a pointer to the buffer pool control block.

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 *VS2 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 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*	*Modules*	*Reason*
SYSVSAM	NNNCCCCB (Note 1)	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 (Note 1)	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 (Note 1)	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 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 it 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.

Flag	*Description*
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 FPL 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

Field	*Description*
CCAREGS (continued)	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 a full tracking buffer. If the buffer is full, a spill routine (IGGPARFS) is called before the tracking MVC 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 – 'IGG0CLA1'
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 cleaning 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		
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.

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 5-36 illustrates the control-flow discussion that is presented in the following paragraphs.

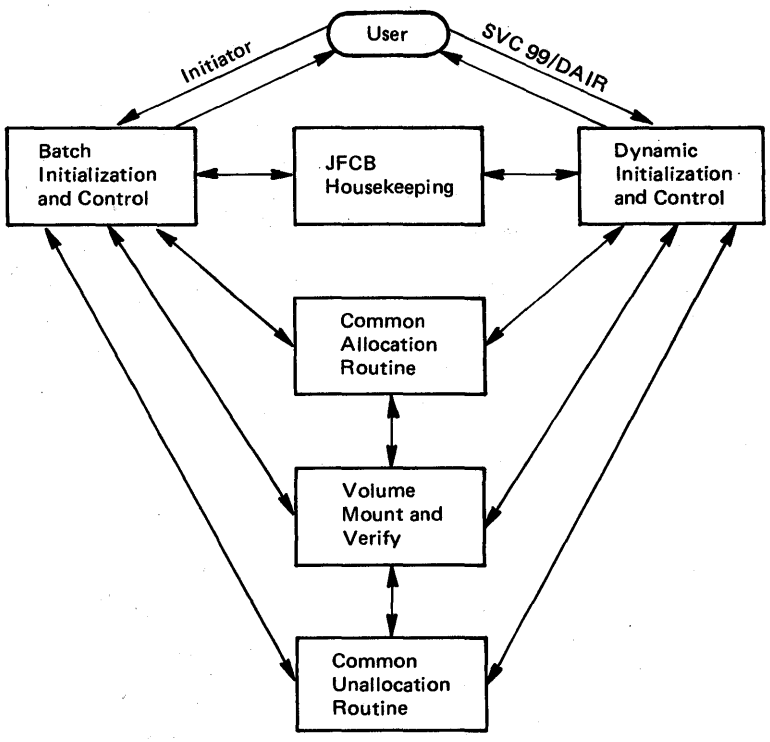


Figure 5-36. The Relationship of the Six Major Functions of Allocation/Unallocation

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
- Demounting incorrect volumes (except in the case of fixed device allocation)
- Scheduling a mount (by building a VM&V request block (VM&V RB) if a volume must be mounted)

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 MV&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 (IEFVPOST) posts the device-end ECBs in all MVCAXs. Any VM&V that is waiting looks at all UCBs being waited for. Volume serials are read and verified when the devices become ready.

Volume unloading is accomplished for DASD by 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 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.

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 Volume 6 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 5-37 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 5-38 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 were found to be mounted with the permanently-resident mount attribute).

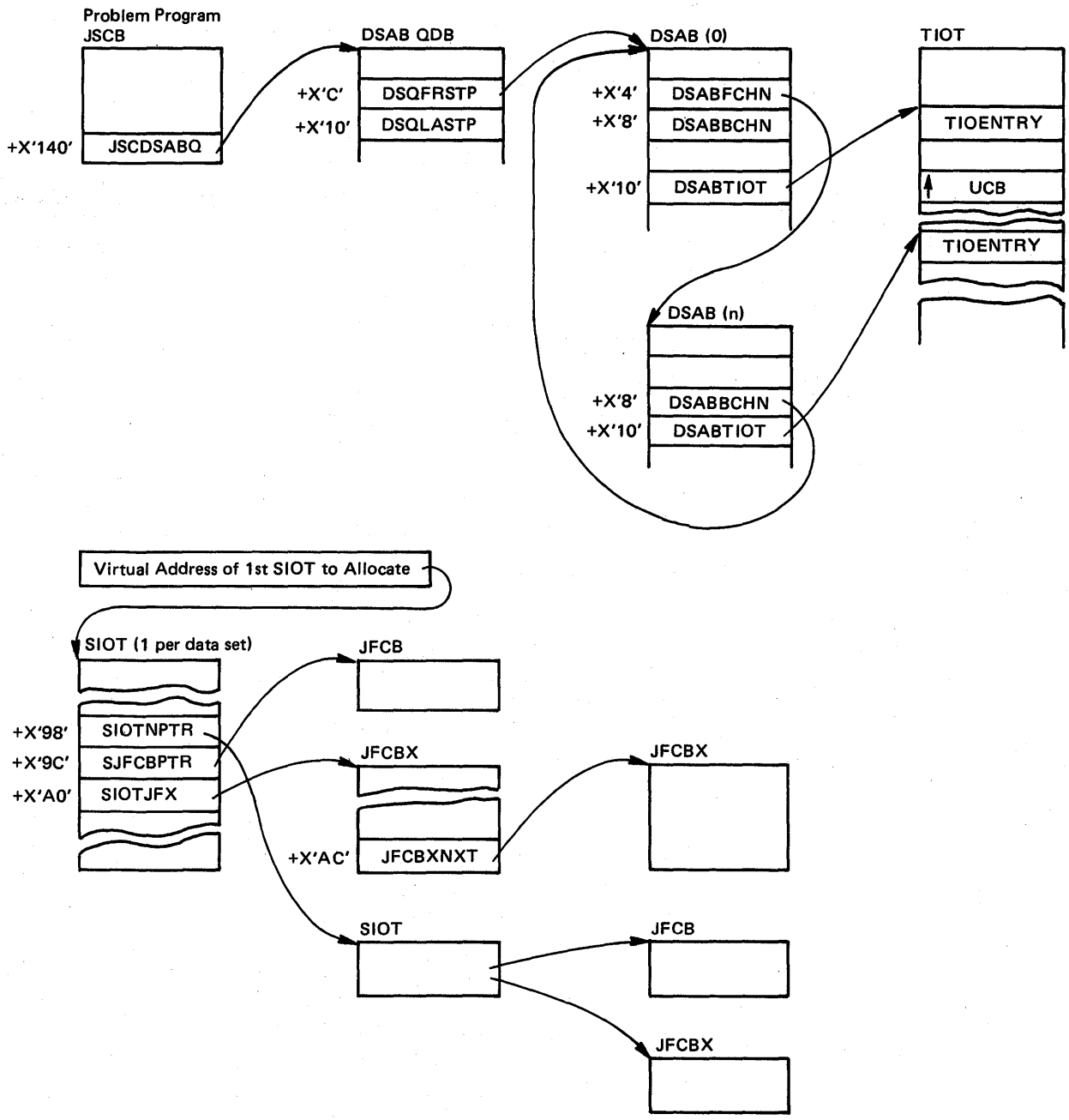


Figure 5-37. Common Allocation Input

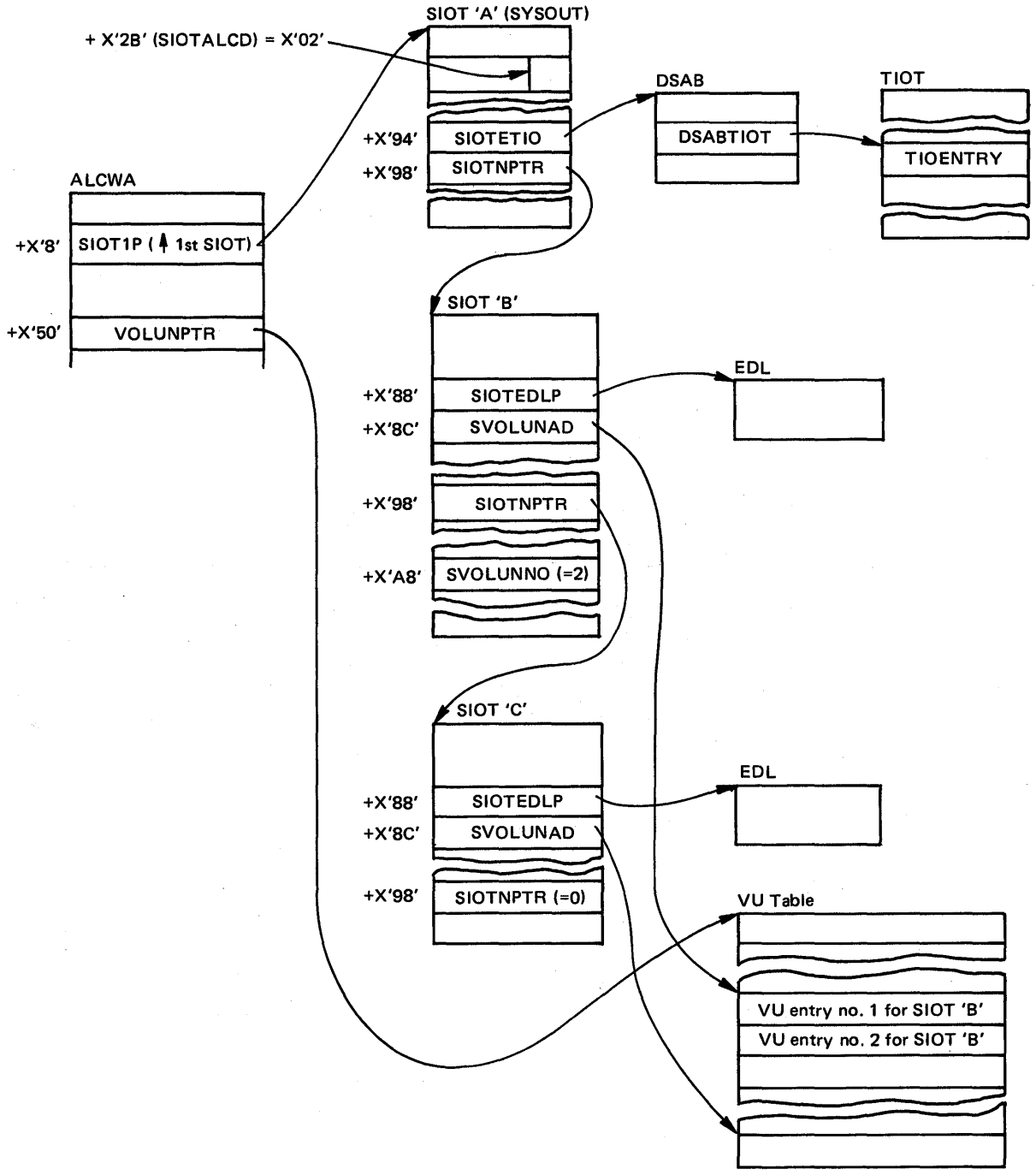


Figure 5-38. Common Allocation Control Blocks After Construction of Volunit Table and EDLs.

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.

Debugging Hints

Hints for debugging specific problem areas are described here including:

- Allocation Serialization
- Device Selection Problems (Non-Abend)
- 0B0 Abend
- 0C4 Abend in IEFAB4FC, or Loop in IEFDB413
- 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 de-mountable 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 non-specific 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 devices 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 allocation load module IEFW21SD. 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 *OS/VS2 Data Areas* (microfiche), via mapping macro IEFZB421.

Address Space Termination

When an address space is being abnormally terminated, the allocation address space termination routine, IEFAB4E5, gets control. This routine releases any allocation groups held by the address space and unallocates any non-shareable units allocated to the address space. Non-shareable units include all units except shareable direct-access devices. The ASID of the address space allocated to a non-shareable unit is at X'E' (halfword) in the common UCB extension.

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, or Loop in IEFDB413

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. If this error occurs, it can be diagnosed as follows:

1. Find the device type (+ X'10') in the UCB of the specific unit.

2. In the EDT (a CSECT that is well mapped in its assembly at SYSGEN), find the look-up entry representing the device type in the UCB. If the requested unit is not among the units represented by the look-up entry, the problem is that the device type in the UCB was changed.

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 5-39 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.

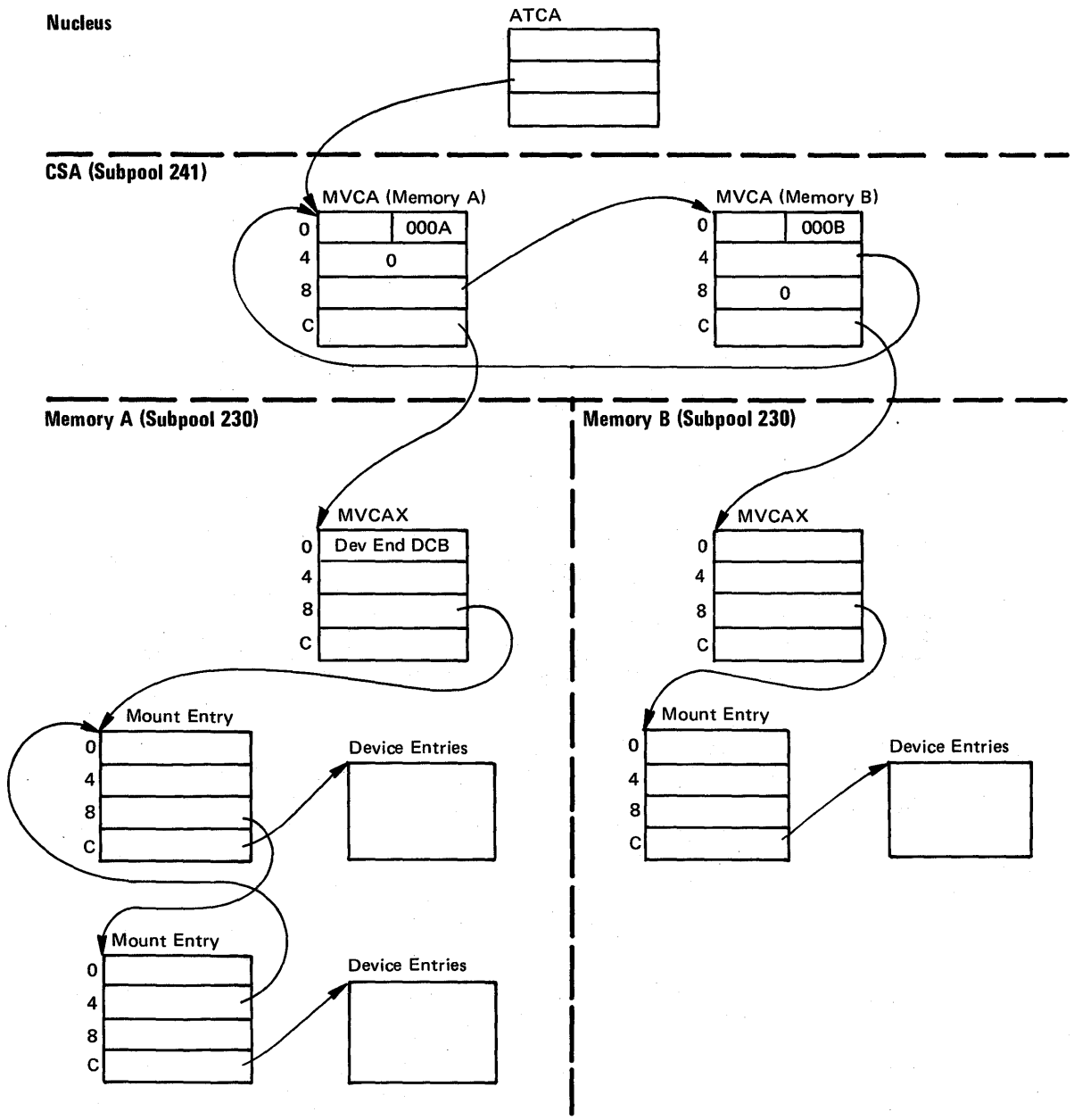


Figure 5-39. VM&V Control Block Structure

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	Dynamic Allocation 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 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	4738	IEF140I	Directory space request too large.
35	reserved		
36	4740	IEF273I	Invalid user label request.
37	reserved		

* — means that the error cannot be set in dynamic allocation.

Reason Code	Dynamic Allocation Error Reason Code	Message	Meaning
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	IEF485I	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.
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 (SU 32 only).
96	03A4	IEF741I	Protect request failed – invalid data set/volume specification (SU 32 only).

* – means that the error cannot be set in dynamic allocation.

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	Dynamic Allocation 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.
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	IEF715I	MVS volume inaccessible.
24	04A0	IEF717I	Specified virtual volume group (VVGRP) name does not exist.
25	*	IEF718I	Space or virtual volume group (VVGRP) 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 is a job entry subsystem for OS/VS2 MVS. An overview of the JES2 structure is presented in this section. For detailed information on JES2 structure, logic, and control block formats, see *OS/VS2 JES2 Logic*. A partial list of major JES2 control blocks showing storage location and primary use may be seen in Figure 5-49 at the end of this section.

JES2 is a subsystem that runs as an operator-started job in a separate address space. It provides input and output spooling for local and remote unit record devices, and simplified batch scheduling. A subsystem support module, provided by JES2 and located in the pageable link pack area (PLPA), is utilized to communicate with other system components in performing job selection and execution. JES2 may be connected to as many as seven other JES2 subsystems via the multi-access spool direct access storage devices.

Job Processing Through JES2

JES2 job processing is divided into the following five major phases:

Input

Jobs are read into the system from online card readers, remote terminal and internal reader interfaces (TSO LOGONs, TSO-submitted jobs, system tasks, or jobs presented to the internal reader from other sources). These jobs are then entered into a priority queue to await processing by the next stage.

Conversion

As soon as the converter is available, the JCL for a job is passed through the converter, scanned for syntax errors, and converted into internal text. Any jobs having JCL errors will bypass execution and be queued for output processing immediately. Those jobs that successfully complete conversion are queued by priority, within class, to await an eligible initiator for execution.

Execution

Jobs are selected by priority, within class, for an eligible initiator. Input cards are supplied as required to the executing program. Output records are received and written onto JES2 spool devices. At the completion of execution, the job is placed in a queue to await output processing.

Output

The print data sets created during execution, and messages created during earlier stages, are printed. The punch data sets are punched.

Purge

Upon completion of all processing required for the job, the direct access space acquired by JES2 for the job and all JES2 resources associated with the job are released.

JES2 Structure

JES2 consists of two basic modules. HASJES20, which operates in JES2 address space and provides the subsystem's job processing functions; and HASPSSSM, which is located in PLPA and provides the interface between the operating system and the HASJES20 programs.

HASJES20 Program Structure

The HASJES20 module is made up of seven tasks that perform JES2 job processing. The JES2 main task provides the basic functions of reading and spooling job input, converting JCL, selecting jobs for execution from the JES2 job queue, receiving and outputing job output, and job cleanup, all accomplished with a set of programs called basic functional processors. These processors are supported by another set of programs, the control processors, which provide subsystem control and JES2 facilities. Both sets of processors use numerous subroutines called control service programs.

The heart of HASJES20 is the dispatcher, which schedules and dispatches various processors under the single TCB of the main task. Since the main task cannot afford to go into a wait state, any JES2 programs that have the potential for waiting are isolated as subtasks. There are six JES2 subtasks:

1. Conversion subtask – links to OS/VS2 converter.
2. Image loader subtask – loads universal character set (UCS) and forms control buffer (FCB) images.
3. System management facility (SMF) subtask – issues SVC 83 to write accounting records for main task.
4. Communications subtask – issues SVC 34 and SVC 35 for main task operator communications.
5. SNA subtask – initializes JES2 use of the VTAM interface with OPEN ACB.
6. Dynamic spool allocation subtask – initializes JES2's spool volumes (SYS1.HASPACE and SYS1.HASPCKPT).

HASJES20 Module Structure

HASJES20 shown in Figure 5-40, consists of ten source modules which contain JES2 main flow job processing code and associated directories. The HASPINIT module is loaded in JES2 address space to initialize the subsystem. After initialization HASPINIT is deleted. HASPRDR, HASPXEQ, and HASPPRPU contain the functional and control processors needed to effect major job processing steps, along with related specialized subroutines and subtasks. The first 4K of HASPNUC is fixed in JES2 memory since it contains the system routines which provide support to the other modules. HASPNUC also contains the HCT and JES2 module directory (see Figure 5-41). HASPRTAM contains all the access method and line management functions to support both bisynchronous and systems network architecture (SNA) remote job entry terminals. Communication functions are isolated to this module. All processing associated with JES2 multi-access spool systems is contained in HASPMISC along with spool initialization, checkpoint, and job purge operations.

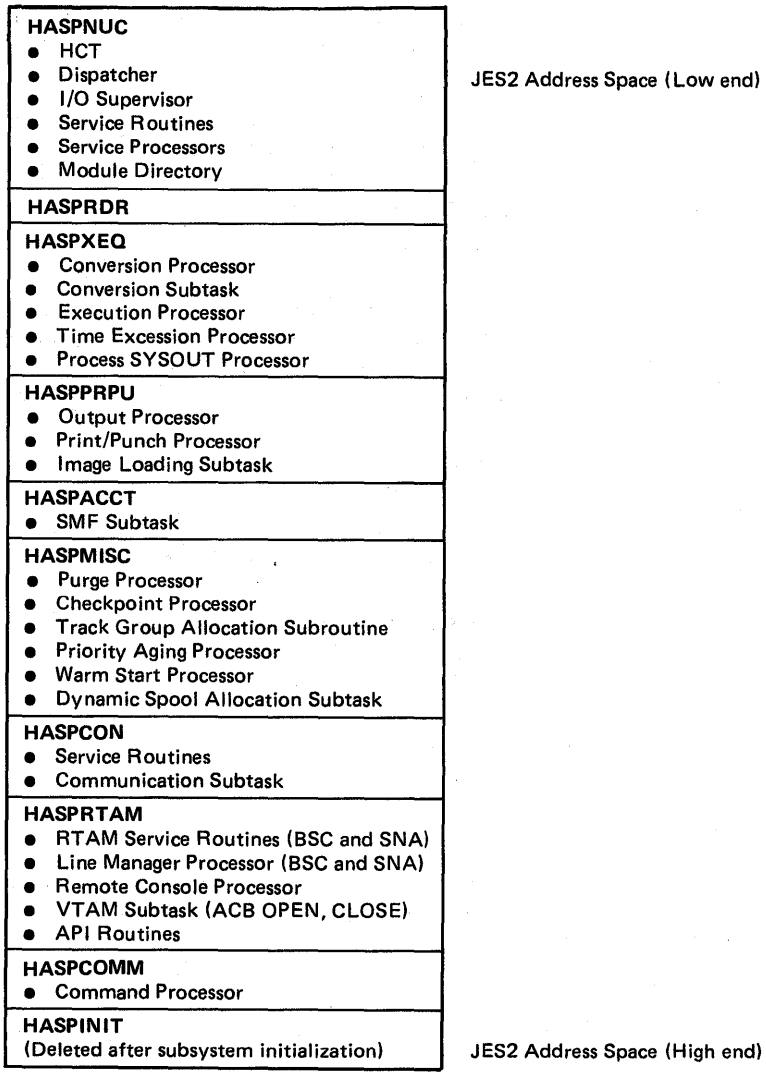


Figure 5-40. HASJES20 Module Map

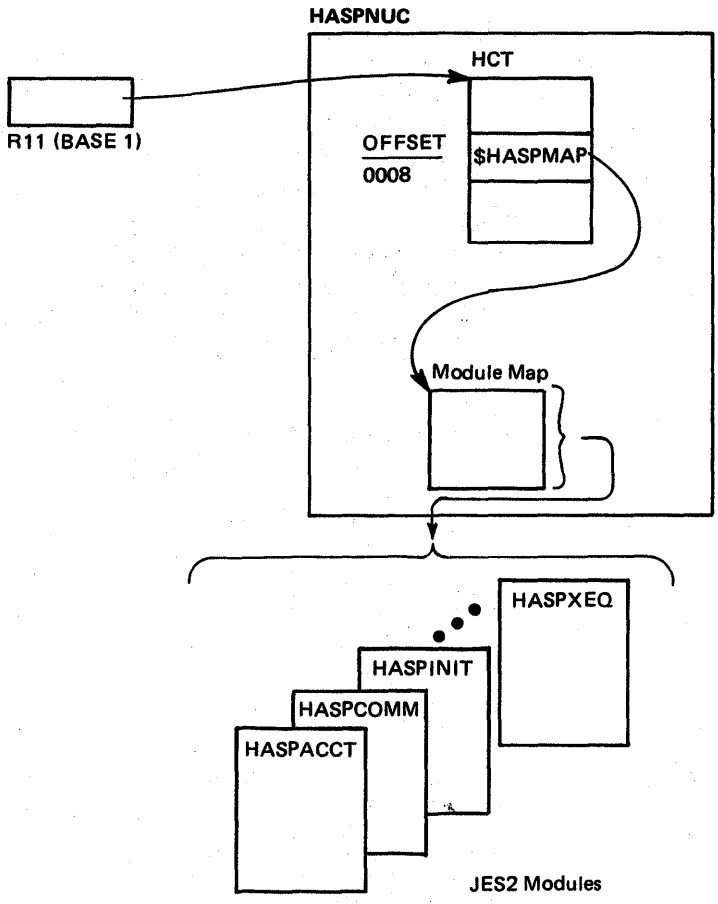


Figure 5-41. Locating the JES2 Module Directory in HASPNUC

HASP Control Table (HCT)

The global directory for HASJES20 is the HASP control table (HCT), which is found in HASPNUC. (Figure 5-42 shows the major vector fields of the HCT.) In HASJES20, R11 may be used to locate the HCT. Eight bytes into the HCT is the address of the JES2 module map which contains a symbolic name and VCON entry for each of the JES2 modules.

HASJES20

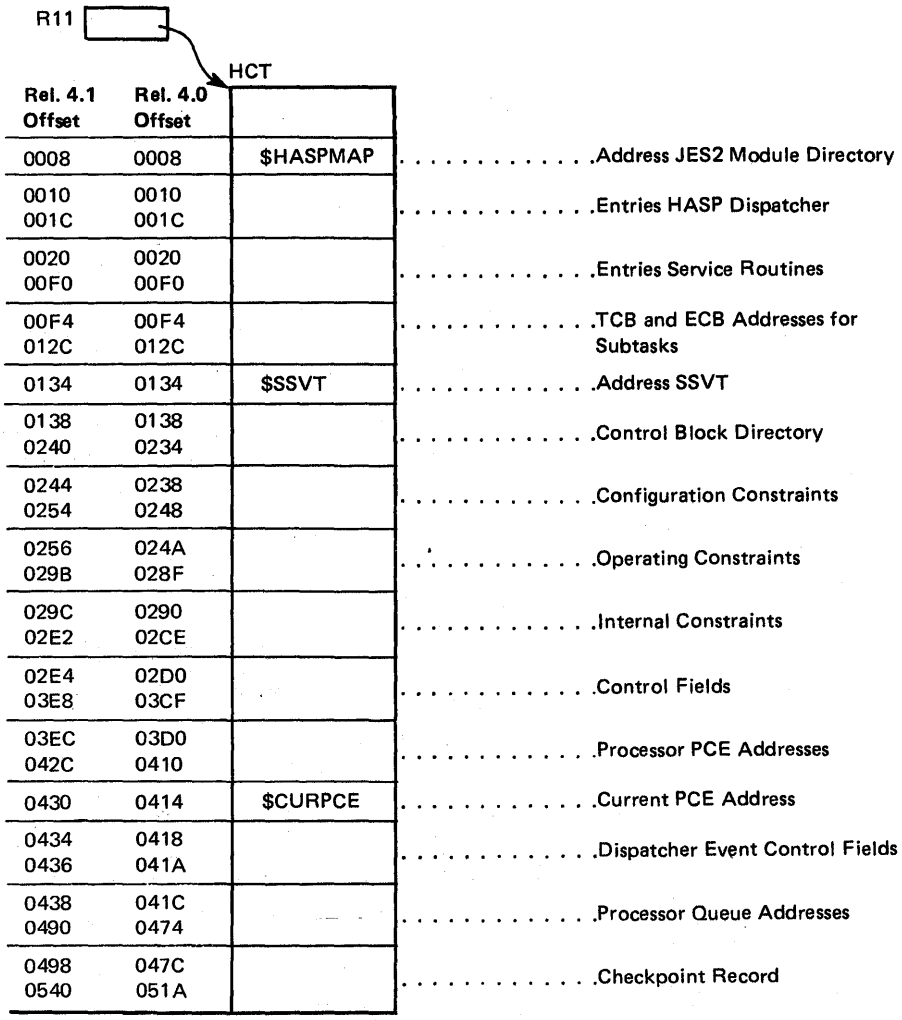


Figure 5-42. HCT Major Vector Fields

HASPSSSM

Located in PLPA, HASPSSSM interfaces directly with the operating system through the formal subsystem interface (SSI) to provide job scheduling, data management (SYSIN and SYSOUT), and operator communications. HASPSSSM contains function routines which are invoked through the use of vectors in the subsystem vector table (SSVT) shown in Figure 5-43. The vectors are used by the operating system to invoke functions which are defined by the IEFJSSOB macro expansion. Additional SSVT vectors are used by the HASJES20 module to provide services to the rest of the JES2 system. During execution of functions represented by the SSVT vectors, additional vectors are set into data extent blocks (DEBs) and access method control blocks (ACBs) for data management support. In the performance of its functions, HASPSSSM makes requests for services to the HASJES20 module running under the JES2 TCB as well as to the operating system. The module is entered in the privileged state. The storage relationship of HASPSSSM and HASJES20 to the operating system is shown in Figure 5-44.

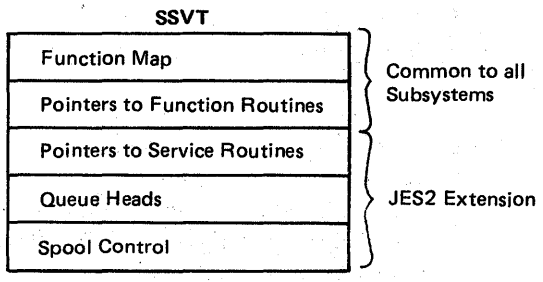


Figure 5-43. The Subsystem Vector Table

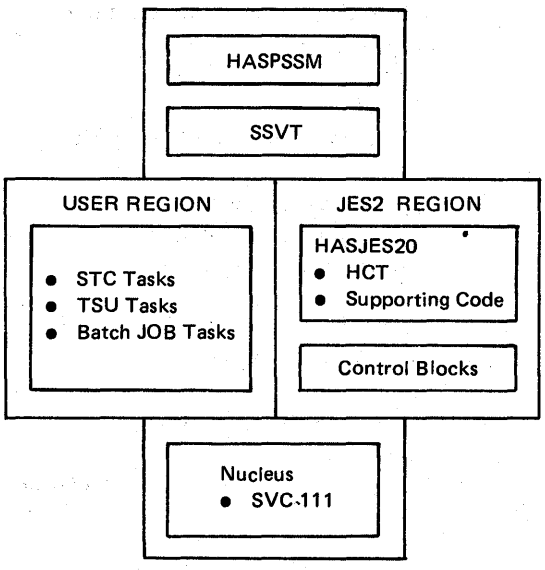


Figure 5-44. HASPSSM – HASJES20 – OS/VS2 Relationship

Subsystem Interface

MVS interfaces formally with JES2 by building a subsystem options block (SSOB) and issuing the IEFSSREQ macro. This formal subsystem interface is shown in Figure 5-45. The subsystem is entered by indexing the SSVT for the entry pointer into HASPSSSM. HASPSSSM performs the requested function, if necessary communicating with HASJES20 by cross-memory post and returns. The SSVT, located in CSA subpool 241, contains the pointers necessary for MVS/JES2 and HASPSSSM/HASJES20 communication. With the HCT, the major control block in HASJES20, the SSVT forms the central directory for JES2.

The subsystem interface is used for the following:

A. Job scheduling and control functions
   1. Job selection
   2. Job deletion (termination)
   3. Re-enqueue job
   4. Request job identification
   5. Return job identification
   6. End of memory
   7. End of task

B. Data set access method functions
   1. Allocation
   2. Open – activates following interfaces:
      a. GET/PUT/PUT ENDREQ/NOTE/POINT
      b. End of block (SVC 111)
   3. Checkpoint
   4. Restart
   5. Close
   6. Unallocation

C. TSO/external writer communications
   1. Process SYSOUT
   2. CANCEL
   3. STATUS
   4. User identification validity check

D. Operator communications
   1. Command processing (SVC 34)
   2. Write to operator (SVC 35)

In addition to the formal SSOB interfaces, the following miscellaneous subsystem interfaces are defined:

1. Exit from the OS/VS2 converter
2. Unsolicited device end
3. Privilege status from the program properties table.

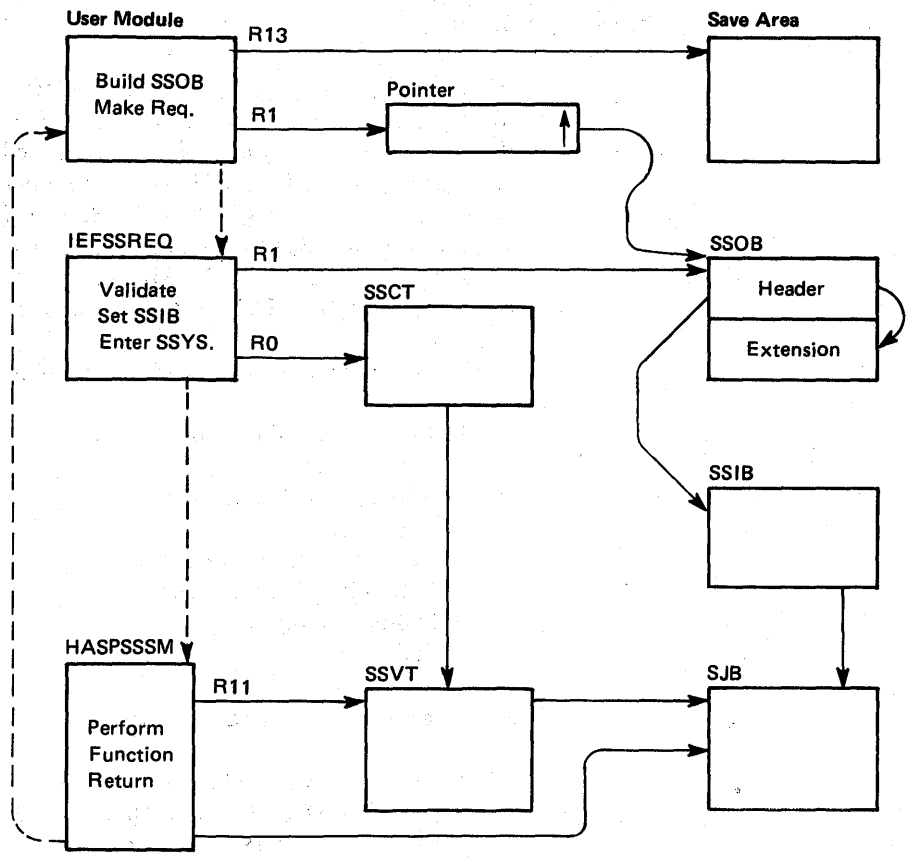


Figure 5-45. Formal Subsystem-Interface Vectors

Dispatcher Structure

The JES2 dispatcher allocates processor time to the JES2 main task processors. Each processor is represented by a control block called a processor control element (PCE). When a processor is eligible for dispatching, its PCE is on a dispatcher queue called the $READY queue. When a processor is waiting on an event, it is ineligible for dispatching. If the processor is waiting for a resource, its PCE is chained to the designated resource wait queue; if the processor is waiting for a specific event, its PCE is queued to itself via a specific event wait field, 'PCEEWF', in the PCE. The currently active processor's PCE is at the top of the $READY queue and is addressed by the $CURPCE of the HCT. Major queue and event-control fields in the HCT and SSVT are shown in Figure 5-46.

$WAIT

A processor that is currently active remains so until it issues a $WAIT macro instruction, at which time the dispatcher is entered at entry point $WAIT (for a specific event), or $WAITR (for a general resource). The dispatcher continues to dispatch eligible processors from the $READY queue until the queue is found to be empty. At this time control is passed to the dispatcher's resource posting routine, which looks for waiting PCE's that have been posted for events and are therefore eligible for dispatching. All eligible PCE's are moved to the $READY queue, and control passes back to the dispatcher.

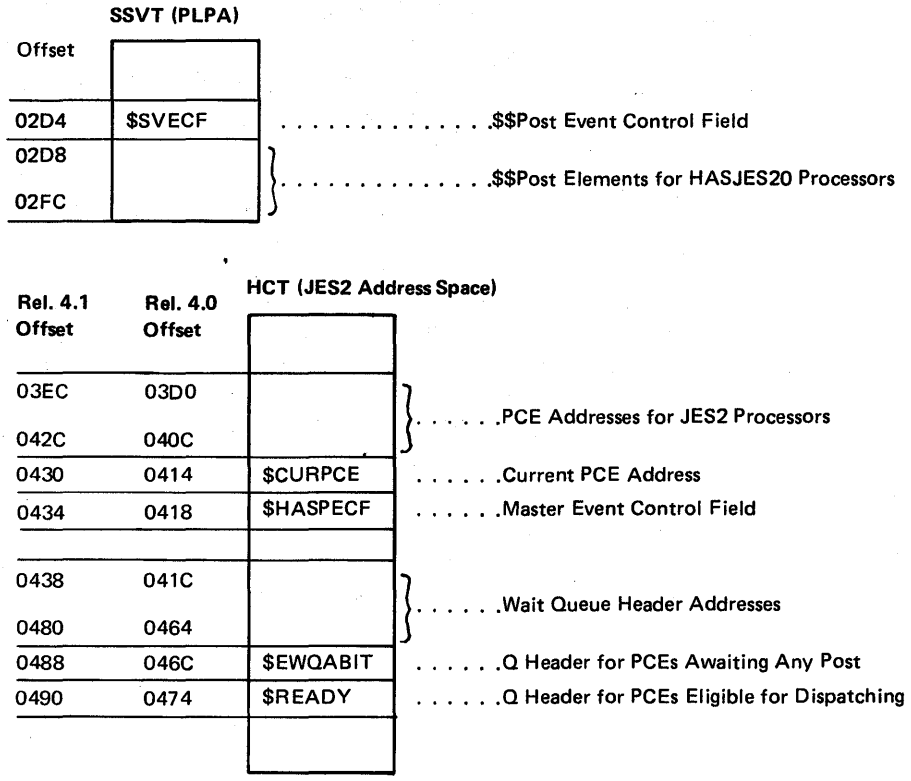


Figure 5-46. JES2 Queue Control Fields

$$POST

The JES2 dispatcher can be notified of work from within its own address space by the $POST macro. In addition, the dispatcher can be notified of work from other address spaces or from subtasks within its address space by the $$POST macro, which causes a HASPSSSM interface routine to cross-memory post the JES2 main task. In this case, the dispatcher post promulgation routine, which receives control when the resource posting routine runs out of work, propagates event posts from the SSVT fields used by HASPSSSM interface routines to the HCT fields in JES2 address space. Here the resource posting routine can pick them up and mark the corresponding processors eligible for dispatching. Control then returns to the dispatcher.

JES2 WAIT

When the JES2 dispatcher determines that there is no more work to be done, it issues an MVS WAIT macro, and waits to be posted for more work. When a $$POST macro is issued, the dispatcher post promulgation routine receives control and transfers the event notifications to the HCT, where they are picked up by the resource posting routine. The corresponding PCE's are transferred to the $READY queue.

Dispatcher Queue Structure

The dispatcher queues are double headed and double threaded. Each PCE (as shown in Figure 5-47) has a chain field to the following PCE entry and one to the preceding entry on the queue. In the special case of the first PCE (referred to as PCE zero), the preceding entry field points to the queue headers, offset so that the queue header appears to be a PCE itself. The last PCE has a following entry field which points back to the queue header. The queue header itself is double, with pointers to the first and last PCE's in the chain. An empty queue has both queue header fields pointing to itself, offset to appear as PCE zero. A PCE that is not on a queue has both its preceding and following entry fields pointing to its origin. In addition to the chain fields, each PCE has a PCEEWF field, which contains information about the type of event the processor is waiting for. Figure 5-48 provides an example of a dump of JES2 processor queue chains.

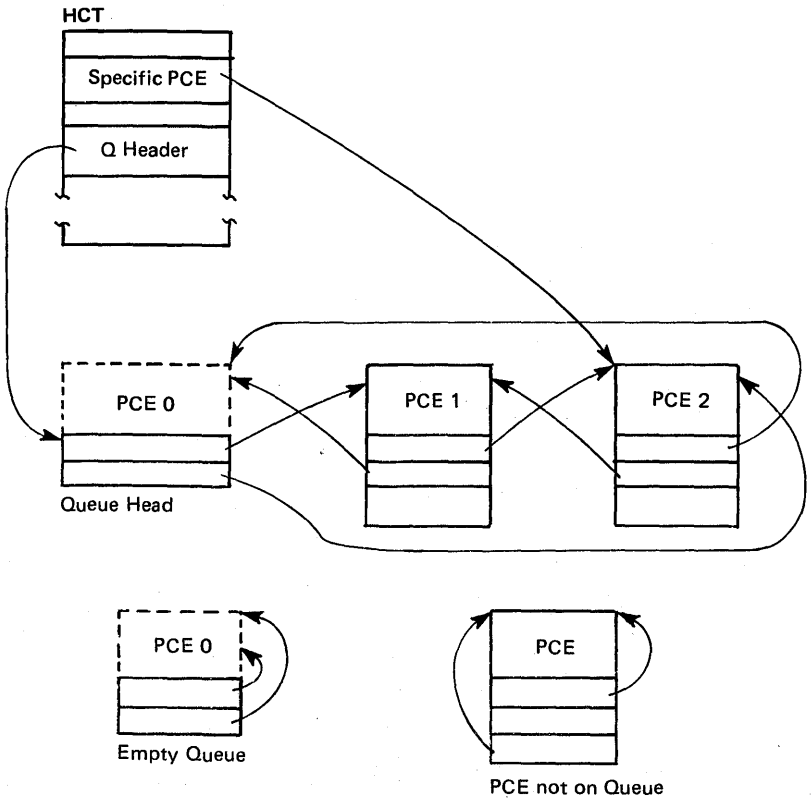


Figure 5-47. JES2 Processor Control Element Relationships

JES2 Error Services

The following routines make up the JES2 error services:

- Disastrous Error Routine
- JES2 ESTAE Routine
- Catastrophic Error Routine
- JES2 Exit Routine
- Input/Output Error-Logging Routine

Disastrous Error Routine

This routine is entered at entry point $DSTERR in HASPNUC whenever a physical I/O error occurs, or whenever a logical error is detected when reading a job control table (JCT) or an input/output table (IOT). The symbol and module names are moved into the message from the $DISTERR macro expansion. A $WTO is issued to notify the operator of the error, and control is returned to the calling processor. The message to the operator is as follows:

$HASP096 DISASTROUS ERROR AT SYMBOL symbol IN MODULE module

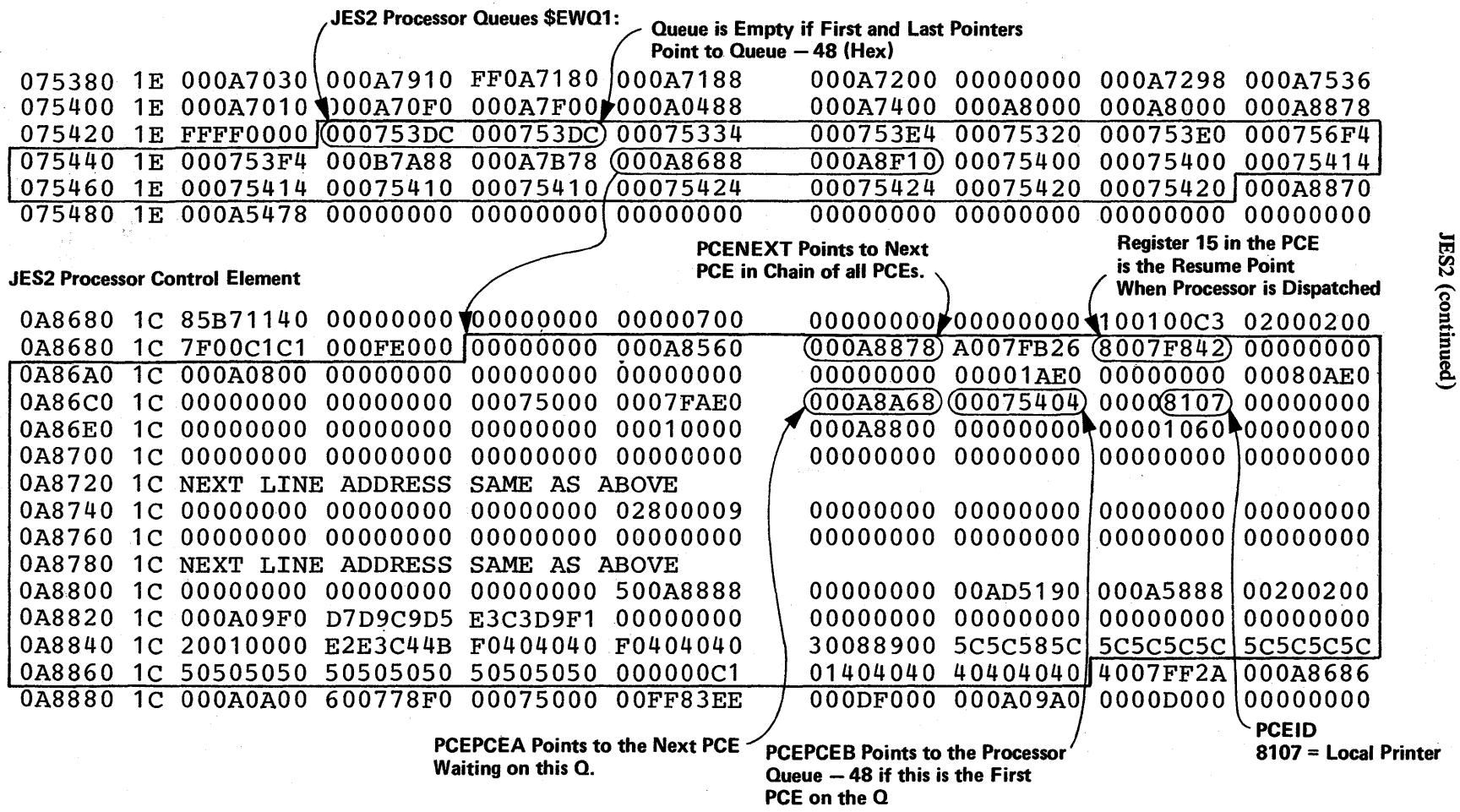


Figure 5-48. Example Dump of JES2 Processor Queue Chains

JES2 should be quiesced and restarted as soon as it is practicable in order to recover any direct-access space that might have been lost as a result of the error.

JES2 ESTAE Routine

This routine is entered at entry point $ABEND in HASPNUC whenever JES2 abends for any reason. The catastrophic error routine is called with an error code of ABND and control is passed to the JES2 exit routine.

Catastrophic Error Routine

This routine is entered at entry point $ERRORTN in HASPNUC whenever an unrecoverable error is discovered by JES2. Register 0 contains the address of a three or four character left-justified error code. The four byte error code field is moved into the operator message which is then written on the operator's console (using a WTO macro instruction) as follows:

$HASP095 JES2 SYSTEM CATASTROPHIC ERROR. CODE=code.

Control is then passed to the JES2 exit routine. The error codes and their meanings are listed in *OS/VS Message Library: VS2 System Codes.*

JES2 Exit Routine

This routine is entered from the catastrophic error routine whenever JES2 is to terminate under abnormal circumstances, and whenever a $P JES2 command is successfully executed. When entered from the catastrophic error routine, the following WTOR message is issued:

$HASP098 ENTER TERMINATION OPTION

The routine waits for the operator to respond with one of the following replies:

- EXIT
- PURG
- DUMP text

If the reply is EXIT, the subsystem vector table (SSVT) termination complete flag and the $SVPOSTP byte are set on. Control is returned to the system in the case of JES2 error detection, by an SVC 3 instruction with register 15 set to 24; or in the case of a JES2 task abend, by a branch to the location in register 14.

If the reply is PURG, the routine attempts to clean up commonly addressable control blocks. If the subsystem is the primary subsystem: the UCB attention index values are set to zero; tasks waiting for CANCEL/STATUS, process-sysout, and storage cell expansion queues are posted; a system management facility (SMF) record may be optionally written, and JES2 subtasks are terminated and detached. Control is then returned to the system as with the EXIT option.

If the reply is DUMP, a $DUMP macro instruction is executed with the text (if any) used as the header. Processing continues as with the PURG reply.

If entry to the routine is through the normal execution of the $P JES2 command, processing is the same as with the PURG option for abnormal terminations, except that control is returned to the system by an SVC 3 instruction with register 15 set to zero.

Input/Output Error Logging Routine

This routine is entered whenever an unrecoverable input/output error occurs on a JES2 spooling volume, or whenever a line error occurs which may require the attention of the operator. A message to the operator is generated as follows:

- The channel status, channel command code, sense information, track address, and line status are retrieved from the IOB (pointed to by register 1) and formatted.
- The unit address and volume serial are obtained from the UCB.
- The device name (if applicable) is acquired from the device control table (DCT).

The format of the message is described in *OS/VS Message Library: VS2 System Codes.*

JES2 $DEBUG Functions in a Multi-Access Spool Configuration

JES2 systems in a multi-access spool configuration share a single job queue, job output table, master track group map, and remote message spooling queues, all of which are kept on the JES2 checkpoint record. In addition, the checkpoint record contains shared system queue elements (QSEs) and other miscellaneous information needed for inter-system control. The checkpoint record is allocated to one processor at a time. Access to any part is controlled by a system's JES2 checkpoint processor, found in the HASPMISC module. The processor has four major sections:

- Initialization
- Read
- Write
- Release

Initialization

Initialization is executed once when the processor is first activated by the JES2 dispatcher. If the debug option has been selected (&DEBUG=YES), storage is obtained, if possible, for debug copies of the job queue and the job output table (JOT). If sufficient storage is not available, message $HASP452 is issued and processing continues.

Read

This is executed at the beginning of each shared queue ownership period by systems in a multi-access spool configuration. If parameter &DEBUG=YES, the job queue and JOT areas are compared with copies saved just prior to the last checkpoint write. A mismatch indicates an invalid alteration to these shared queue areas and JES2 is terminated with a K01 catastrophic error. This step is skipped for the first read following initialization. All checkpoint records are read from DASD. A lockout warning timer (parameter &WARNTIM) is started and the read is started by $EXCP. IOS performs the actual hardware reserve of the checkpoint device. The processor then waits for read completion or timer expiration.

If the timer expires before read completion, warning message $HASP260 is issued and the warning timer is restarted. The message is issued repeatedly, at warning intervals, until read completion occurs.

If a permanent read error occurs, JES2 is terminated with a K02 catastrophic error.

After a successful read completion, the time stamps in this system's QSE, and in any QSE for which the $ESYS command had been entered, are compared with a time stamp saved in the HCT. A mismatch indicates that another system has illegally taken ownership of a QSE owned by this system, or that the reserve mechanism (hardware or IOS) has failed to prevent simultaneous access to the multi-access spool checkpoint records. JES2 is terminated with a K03 catastrophic error.

Write

This is executed repeatedly as a loop, in response to various requests by other processors or timers. In a multi-access spool environment, the loop operates only during an ownership or hold period.

If parameter &DEBUG=YES, the saved copies of the job queue and JOT areas are compared with the current job queue and JOT areas. If a record has been modified, but its corresponding checkpoint control byte does not indicate so, JES2 is terminated with a K05 abend. A K05 abend indicates failure to issue a $QCKPT or $#CKPT macro prior to executing a $WAIT macro, after modifying the job queue or JOT.

The current hardware time-of-day (TOD) clock is recorded in the HCT, along with this system's QSE and a QSE for which any $ESYS command had been entered. In multi-access spool systems, these stamps are verified following the next read operation to ensure the integrity of QSE ownership.

If parameter &DEBUG=YES, copies of the job queue and JOT areas to be written are saved. In multi-accessed spool systems, these are used prior to the next read operation to detect invalid updates to shared but not owned information.

The write is started by $EXCP and the processor waits for completion. If a permanent error occurs, JES2 is terminated with a K04 catastrophic error. Following a successful completion, the $CKPTACT bit in the HCT is cleared.

Release

Release is executed only by systems in a multi-access spool configuration at the end of each shared queue ownership period.

Miscellaneous Hints on JES2

Starting JES2 – Enqueue Wait on STCQUE

The installation can choose the option to manually start JES2 by changing MSTRJCL with AMASPZAP. When MSTRJCL is changed, JES2 parameters are entered on an operator-issued START command (that must be issued before MVS processing can occur). If the operator misspells JES2 on the START command (such as entering JES), a wait occurs with no indication other than STC (IEESB605) being exclusively enqueued on SYSIEFSD STCQUE behind IEEVWAIT, which does not release the resource until JES2 is initialized. Also note that the CSCB (command scheduling control block) pointed to by the parm list to IEFJSWT is not formatted in the dump.

Therefore, if you encounter an enqueue wait on STCQUE, check that the START command is entered correctly. This also holds true for any normally started tasks (such as mounts or installation started tasks) which cannot be started until the primary job entry subsystem is started.

JES2 (continued)

TYPE	ABBR.	NAME	STORAGE TYPE	PRIMARY USE
Organizational	SSVT	Subsystem Vector Table	CSA SUBPOOL 241	Contains system pointers and parameters for HASPSSSM interface routines.
	HCT	HASP control table	JES2 Address space, HASPNUC module	Major directory for HASJES20. Contains queue headers, control blocks pointers, module and entry pointers and system parameters.
Processor Management	PCE	Processor Control Element	JES2 Address Space	Unit of JES2 dispatcher. Has associated work space and save areas for JES2 processors.
Buffer Management	BUFFER	Buffer	JES2 Address Space	Basic building block for JES2 control blocks (JCT, IOT, Special)
JOB Management (TRANSIENT)	JCT	Job Control Table	SYS1.HASPACE User address space during XEQ	Primary job oriented control block. Contains accounting information and pointers to other job information.
	IOT	Input Output Table	SYS1.HASPACE User address space during XEQ	Contains job DASD information and PDDB's for input/output data sets.
	PDDB	Peripheral Data Definition Block	SYS1.HASPACE User address space during XEQ	Describes a job input or output data set.
	OCT	Output Control Table	SYS1.HASPACE User address space during XEQ	Contains output control records (OCB's) to describe data output records (forms, route, etc.)
Job Management (RESIDENT)	JQE	Job Queue Element	In JQB (JES2 address space)	Represents a job in process. Resides on appropriate job queue chain.
	JQB	Job Queue Buffer	SYS1.HASPCKPT & JES2 address space	Contains the job queue chain header's JQE's and I/O parameters for checkpointing.
	JOT	Job Output Table	SYS1.HASPCKPT & JES2 address space	Central control block for all JES2 output. Contains three kinds of JQE's.
	JOE	Job Output Element	IN JOT (JES2 address space)	Represents output data set by units of work, characteristics of data set, and class of output.
	SJB	Subsystem Job Block	CSA SUBPOOL 231	Represents a job in process to OS/VS2 used by HASPSSSM interface routines.
	SDB	Subsystem Data Block	User address space SUBPOOL 229	Used by HASPSSSM to control processing of data set using HASP Access Method (HAM).
Job Management (MIS-CELLANEOUS)	PIT	Partition Information Table	CSA	Completely describes a JES2 logical partition, its job classes and current state.
	CAT	Class Attribute Table	JES2 Address Space	Describes the attributes of a job class.
	SCAT	SYSOUT Class Attributes Table	In SSVT Space	Describes output classes by print, punch, plot, etc. characteristics.
Unit Management	DCT	Device Control Table	JES2 Address Space	Represents a unit record device or RJE line. Contains all the information necessary to set up EXCP.
	RAT	Remote Attribute Table	JES2 Address Space	Consists of one entry per remote device, containing attributes of device.
Multi-System Management	QSE	Shared Queue Control Element	SYS1.HASCKPT & JES2 address space (JQB)	One per system of a Multi-Access Spool environment, containing identification and cross-system communication parameters.

Figure 5-49. Major JES2 Control Blocks

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 trace subsystem.

System Initialization Processing

At SYSGEN, 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 IEFJESNT). Alternatively, secondary subsystems may be specified in the subsystem names table in load module IEFJSSNT.

The master scheduler base initialization module (IEEVIPL) gives control to the subsystem interface initialization module (IEFJSINT). This module builds a subsystem communication vector table (SSCVT) for each unique name in the JES name table and in the subsystem names table. The SSCVTs are chained together with the primary job entry subsystem SSCVT first, and the master subsystem SSCVT second. These are followed by (in the same order as in their respective tables) the SSCVTs in the JES names table and in the subsystem names table. IEFJSINT also puts the name of the primary job entry subsystem in the JES control table (JESCT). The subsystem vector table (SSVT) for the master subsystem is built and initialized. The subsystem allocation sequence table (SAST) is built 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.

Additional subsystem initialization processing is performed by module IEEMB860. This module has two subsystem initialization functions. The first is to issue operator messages for errors that occurred during subsystem interface initialization; the second is to LINK to the subsystem initialization routines specified in IEFJSSNT.

Operator messages are issued by IEEMB860 rather than IEFJSINT because IEFJSINT executes before the communications task has been initialized. Message IEE730I is issued to indicate that a duplicate subsystem has been specified in the subsystem names table. A subsystem is a duplicate if it is:

- A respecification of the primary job entry subsystem
- A respecification of the MSTR subsystem, or
- A respecification of a subsystem that has been initialized previously.

Message IEE858I is issued if the subsystem names table (IEFJSSNT) could not be found; message IEE859I is issued for each subsystem initialization routine which could not be found. It is the responsibility of the subsystem initialization routine to inform the operator of and recover from errors in that routine. If the subsystem initialization routine fails to recover from these errors, the next entry in IEFJSSNT is processed and the failing subsystem may not be completely initialized.

Subsystem Interface Major Control Blocks

Subsystem interface's major control blocks are the JES control table (JESCT), 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 which are described in the *Debugging Handbook.*

Subsystem Interface (SSI) (continued)

Control Block	Created By	Subpool	Key	Size	Pointed To By	Function	Mapping Macro
JESCT	SYSGEN	NUCLEUS	0	44 bytes	CVT	Contains information needed by the Subsystem Interface and addresses of Scheduler routines.	IEFJESCT
SAST	IEFJSINT	241	0	Note 1	JESCT	Defines the order in which subsystems will be invoked to allocate subsystem data sets.	IEFJSAST
SSCVT	IEFJSINT	241	0	24 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 indications 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 list 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 suppored by the subsystem). Minimum size is 264 bytes, maximum – 1284 bytes.

Control Block Usage is shown in Figures 5-50 and 5-51.

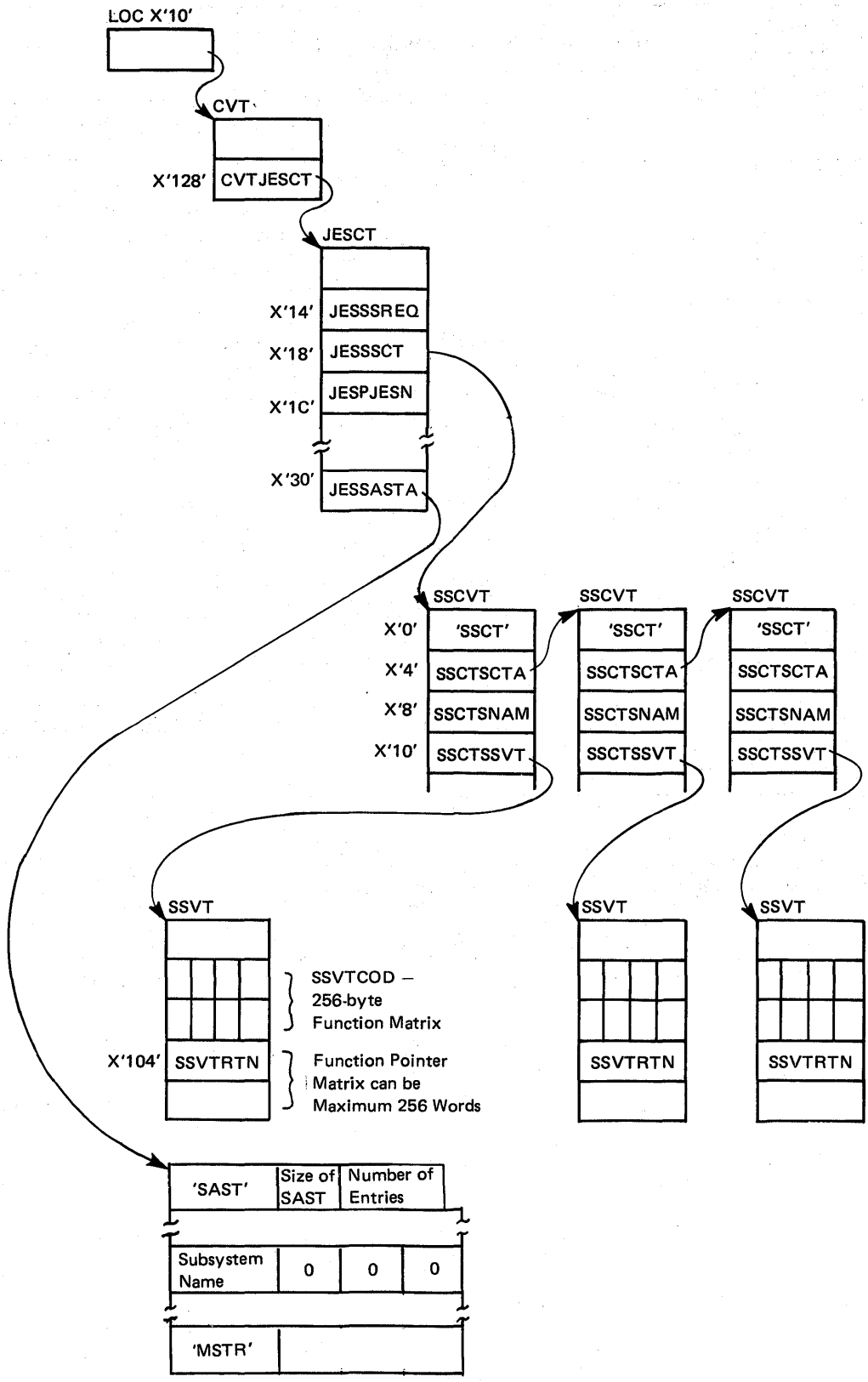


Figure 5-50. Subsystem Interface Control Block Usage

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 5-52).

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

SSIBSSNUM – The name of the subsystem to which the request is being made.

SSIBJBID
SSIBDEST – If the function requires these fields

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

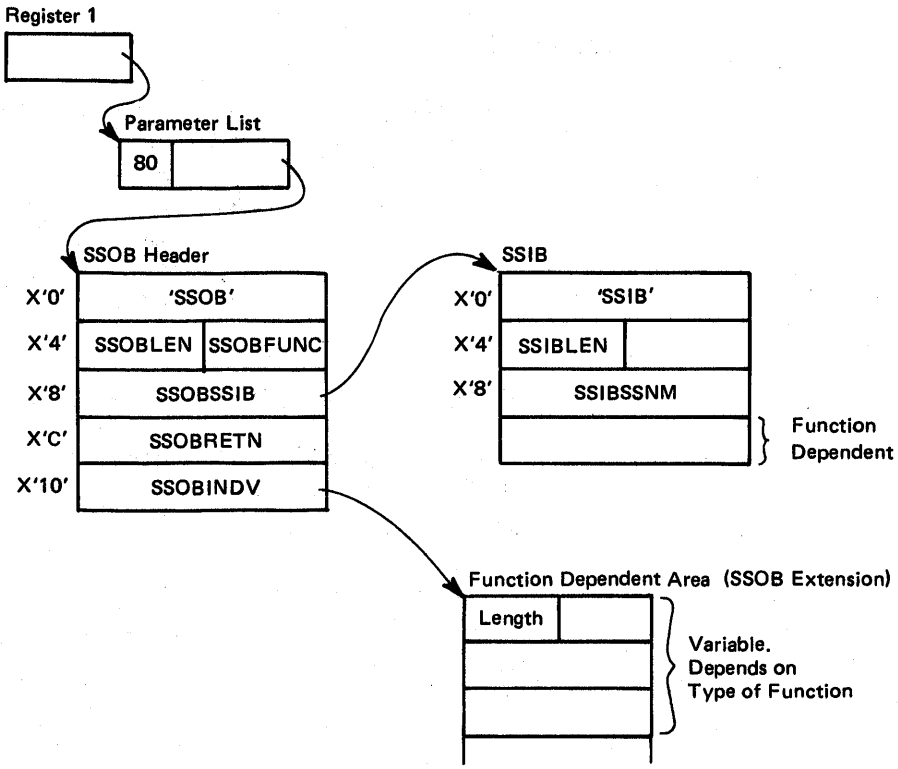


Figure 5-51. Control Block Structure for Invoking Subsystem Interface

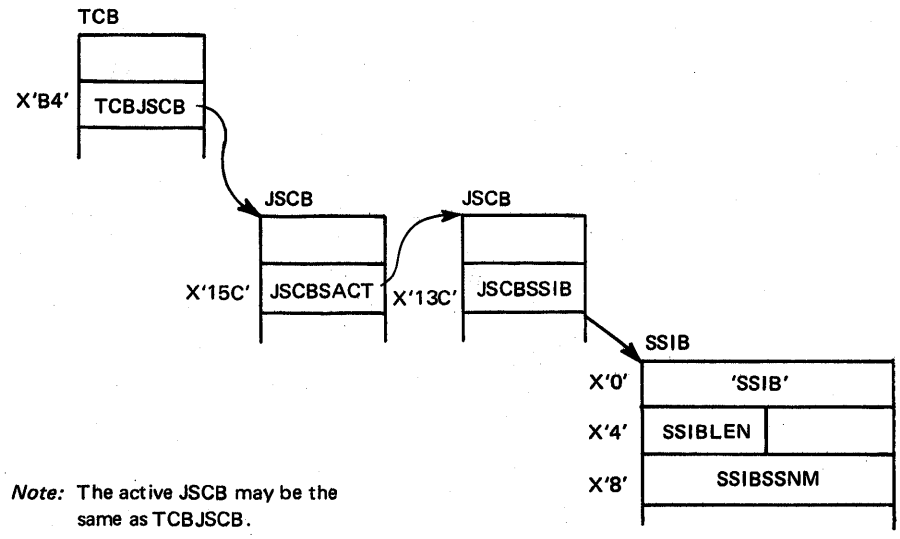


Figure 5-52. Finding the SSIB for a Job When SSOB Pointer is Zero

Register 1 points to a one-word parameter list which points to the SSOB. (See Figure 5-51).

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, Volume 3*, 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, Volume 3.*)

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. The SSIB points to the master subsystem, so module IEFJRASP is the function routine that receives control.
   a. IEFJRASP makes a copy of the SSIB.
   b. For each SSCVT, the name of the subsystem is copied into the SSIB copy. (The master subsystem's SSCVT is skipped.) IEFSSREQ is then invoked for each subsystem.
   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 since 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 IATYSSVT. 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 subystem 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.

- Function code 9 is used in an SSOB with the pointer to the SSIB always zero. This causes the SSIB pointer in the JSCB to be used. If that 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.

The recovery termination manager (RTM) cleans up system resources when a task or 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 (SVC13) 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*, 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 four modules:

1. IEAVTRT1 – RTM entry point processor
2. IEAVTRTM – RTM1 mainline
3. IEAVTRTS – system recovery manager
4. IEAVTRTR – RTM1 recovery routines

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:

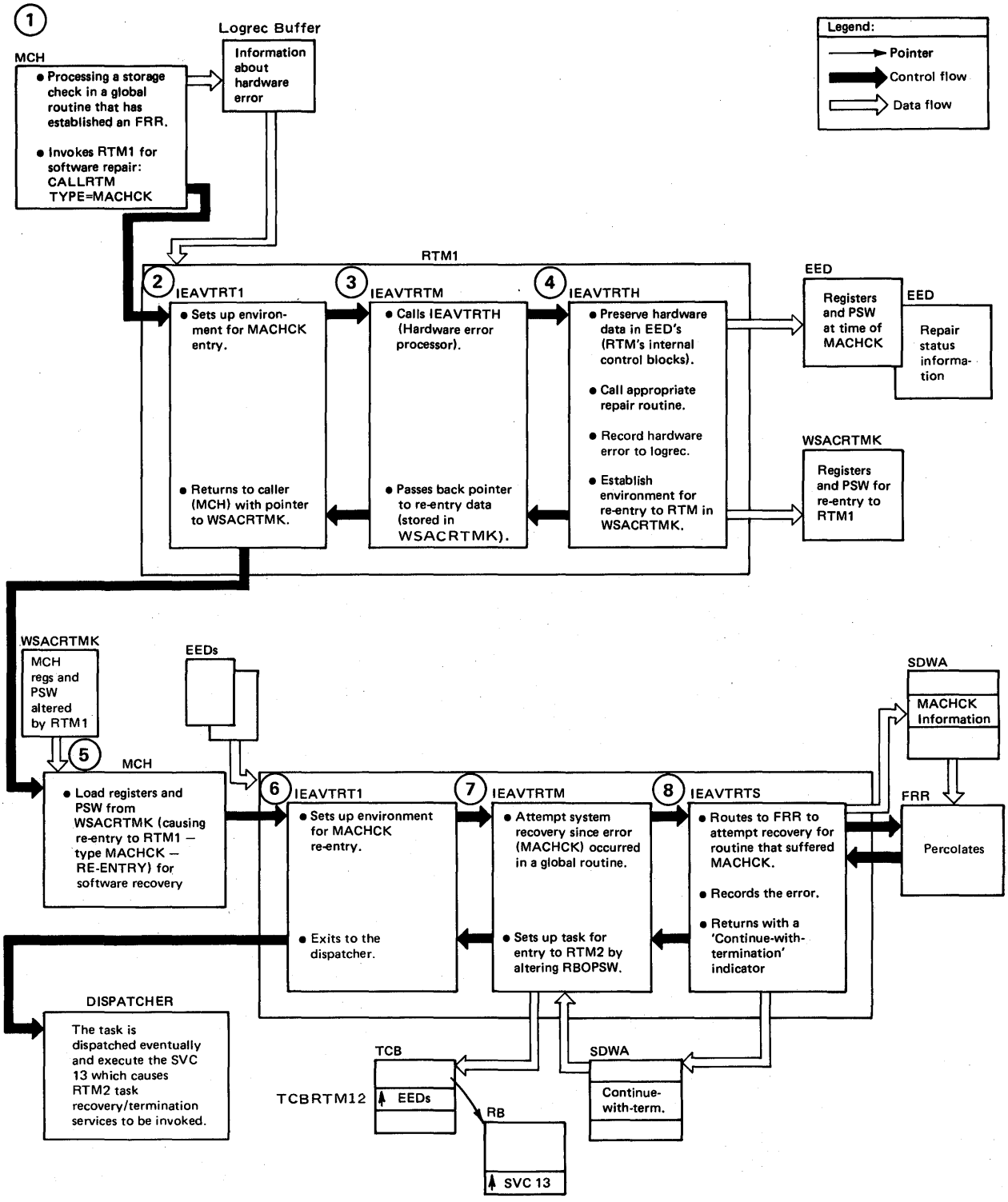
- Hardware error processing
- Normal end-of-task termination
- Abnormal end-of-task termination
- Retry
- Cancel
- Address-space termination.

Hardware Error Processing

Depicted here 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.

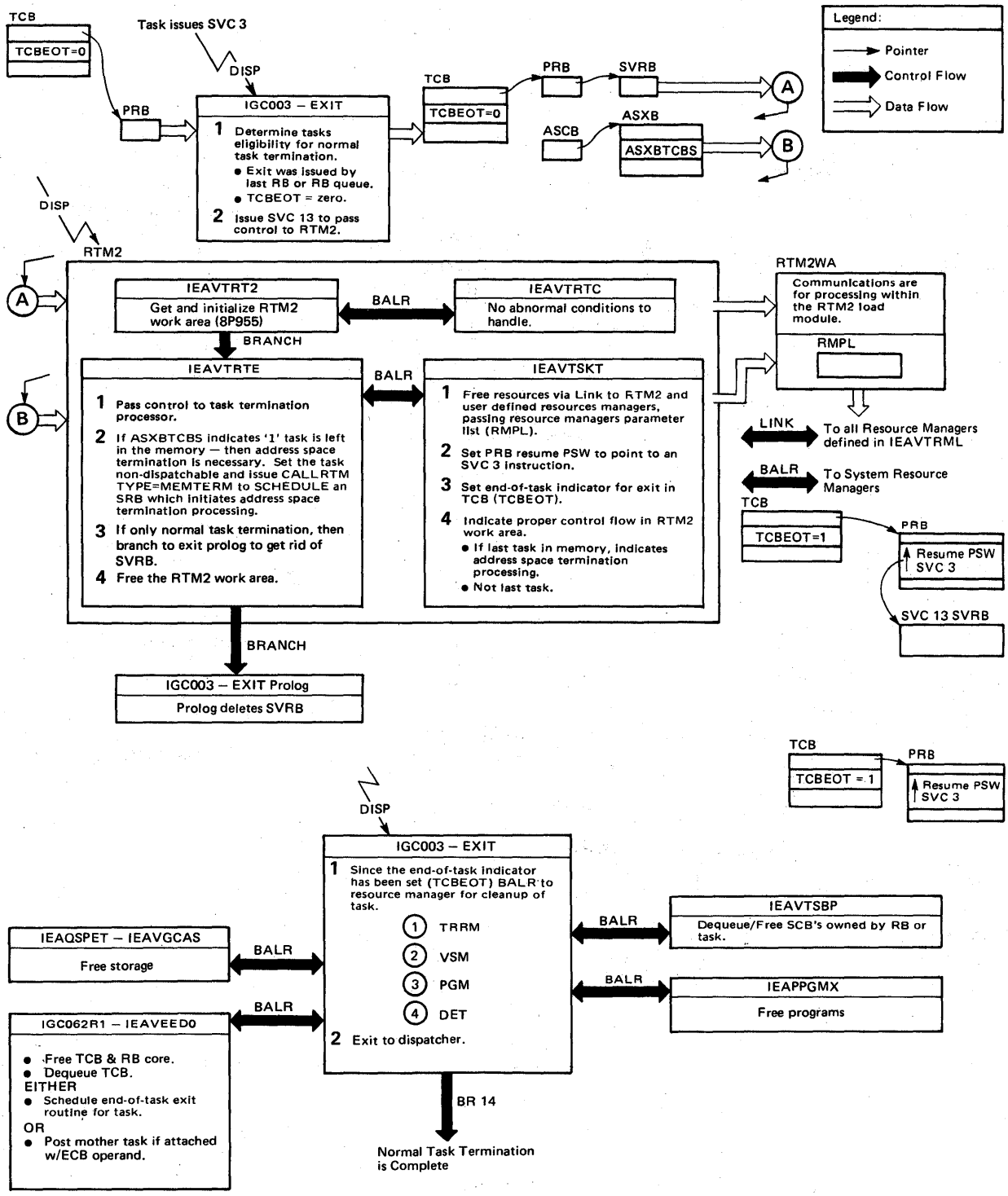
The use of extended error descriptors (EEDs) allows the LOGREC buffer to be available for further possible machine checks and is the mechanism for passing information to RTM1 and RTM2. The information in the global system diagnostic work area (SDWA) used by RTM1 recovery was obtained from the EEDs. RTM2 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
Registers 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
EEDs
SDWA
MACHCK Information
5 MCH
• Load registers and PSW from WSACRTMK (causing re-entry to RTM1 – type MACHCK – RE-ENTRY) for software recovery
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
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.

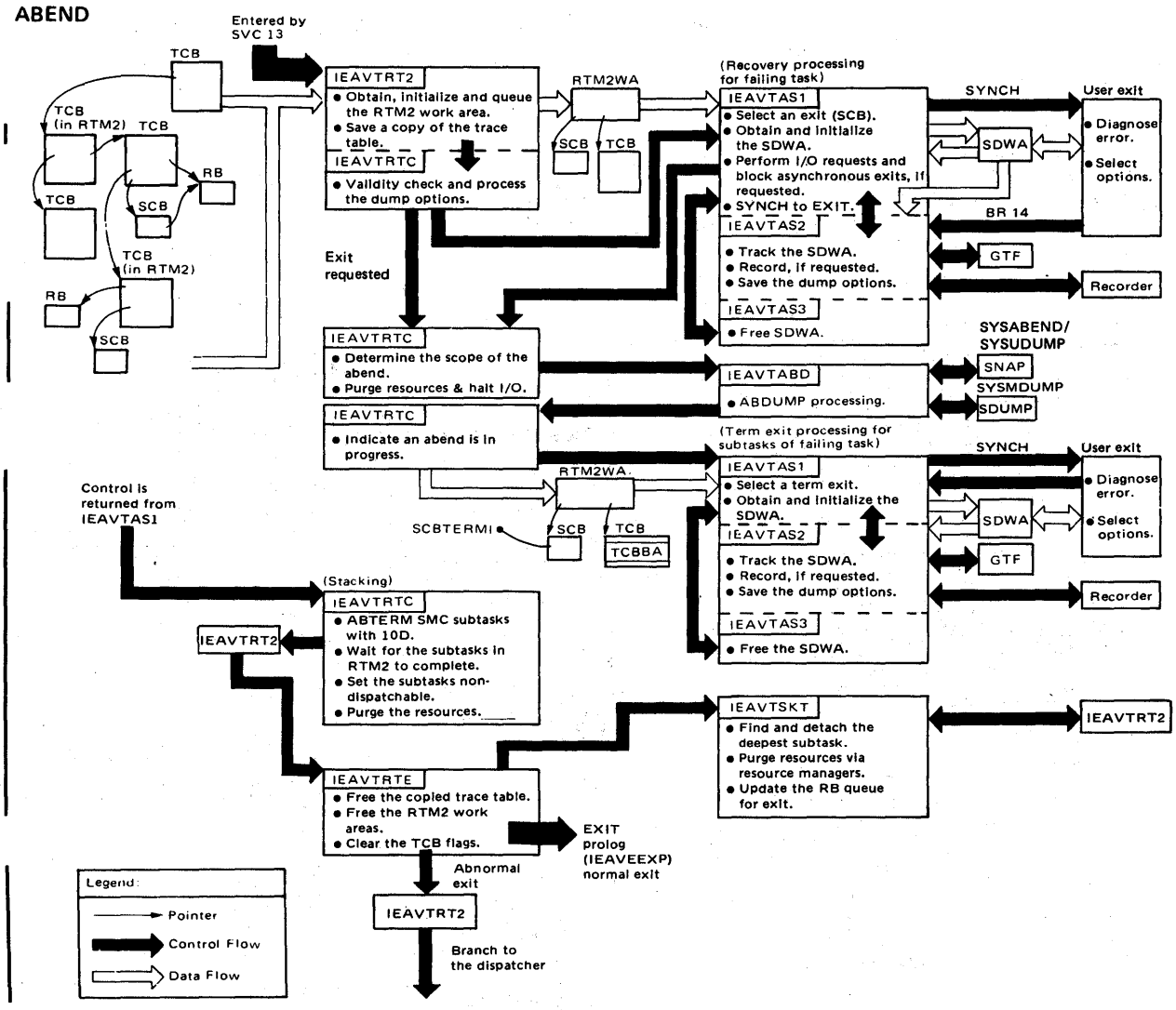
Normal Task Termination

EXIT and parts of RTM2 make up this function. 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.



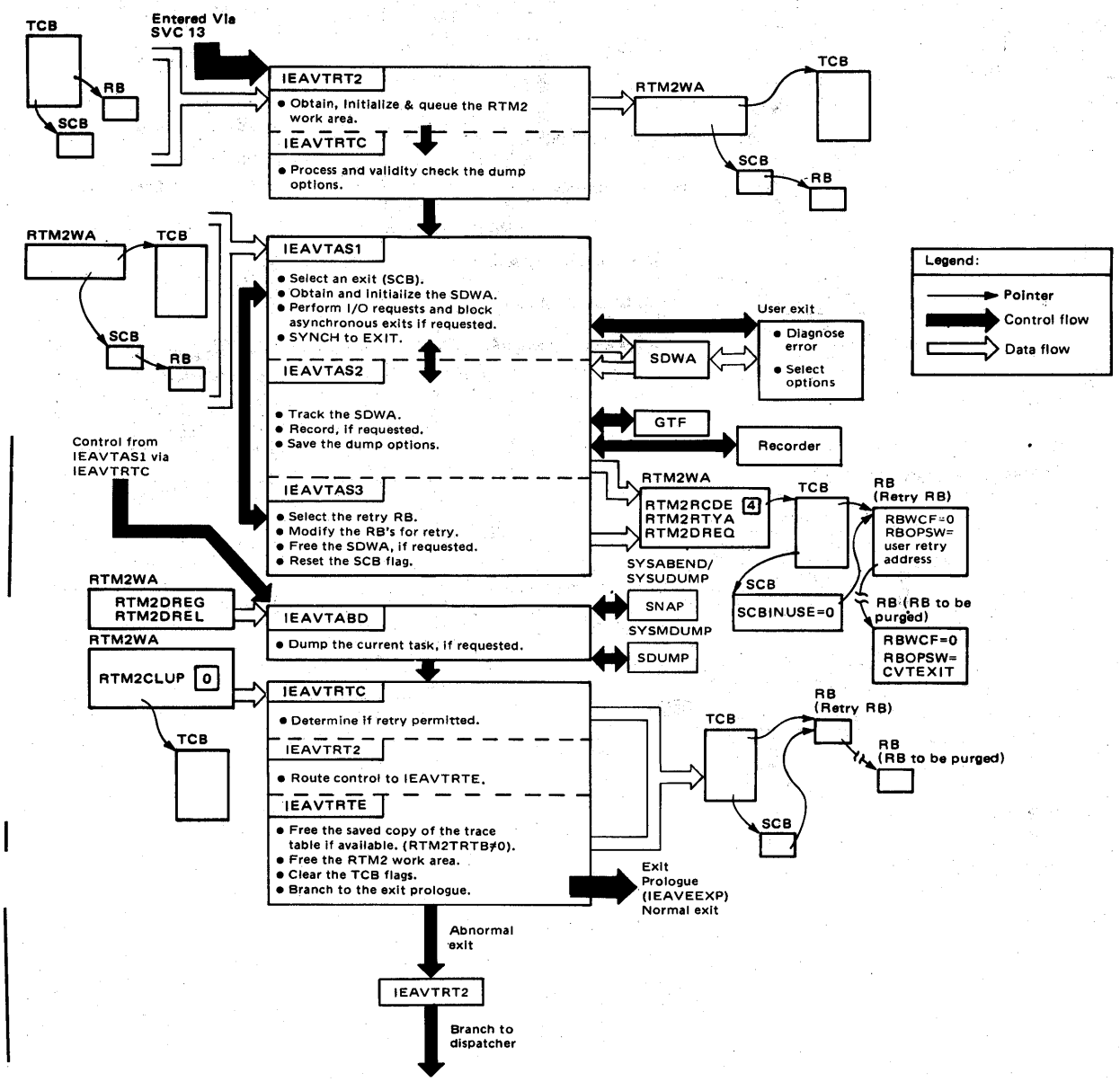
Abnormal Task Termination

Shown here 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.



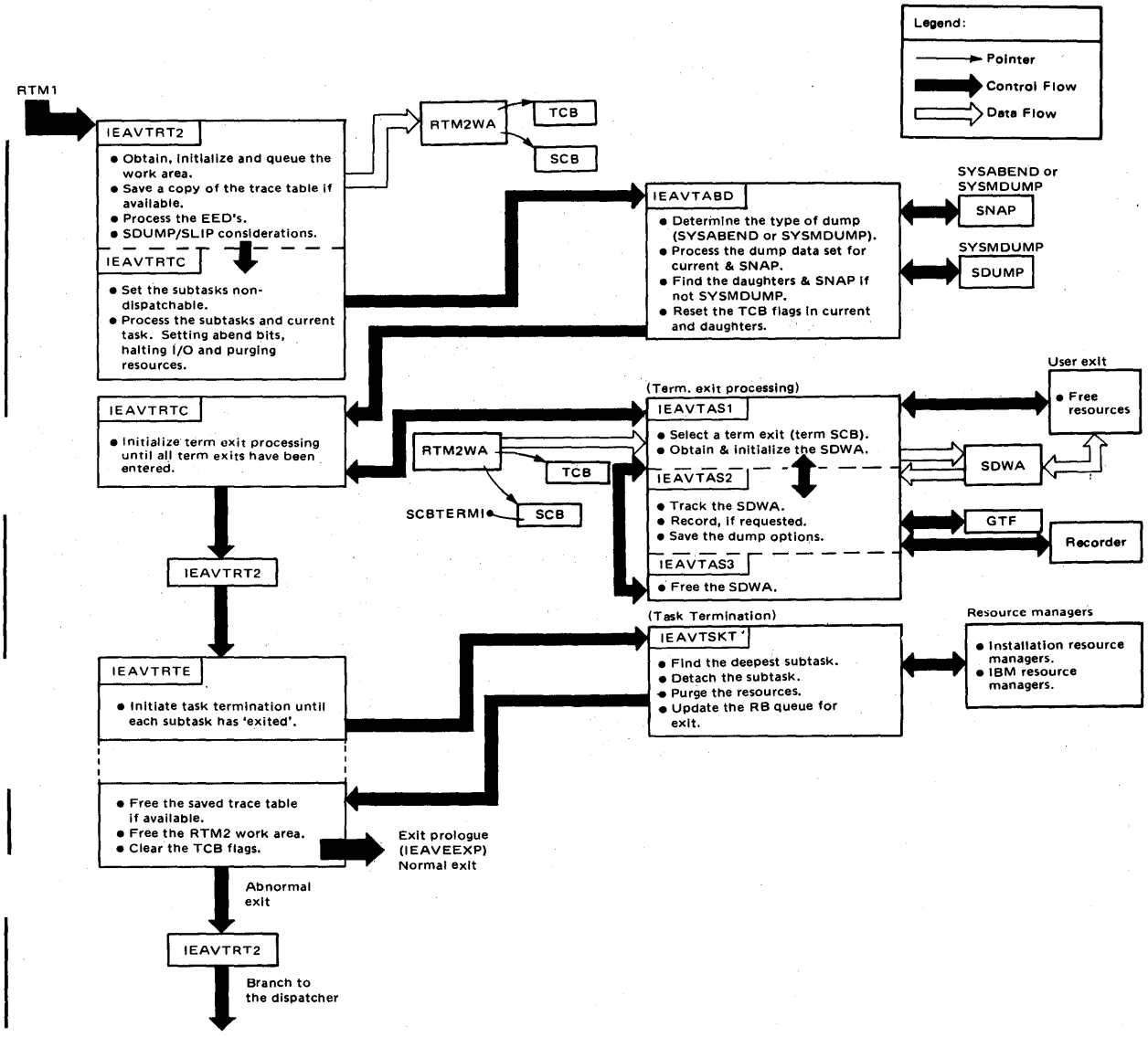
Retry

Shown here 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. 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 here 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.



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.

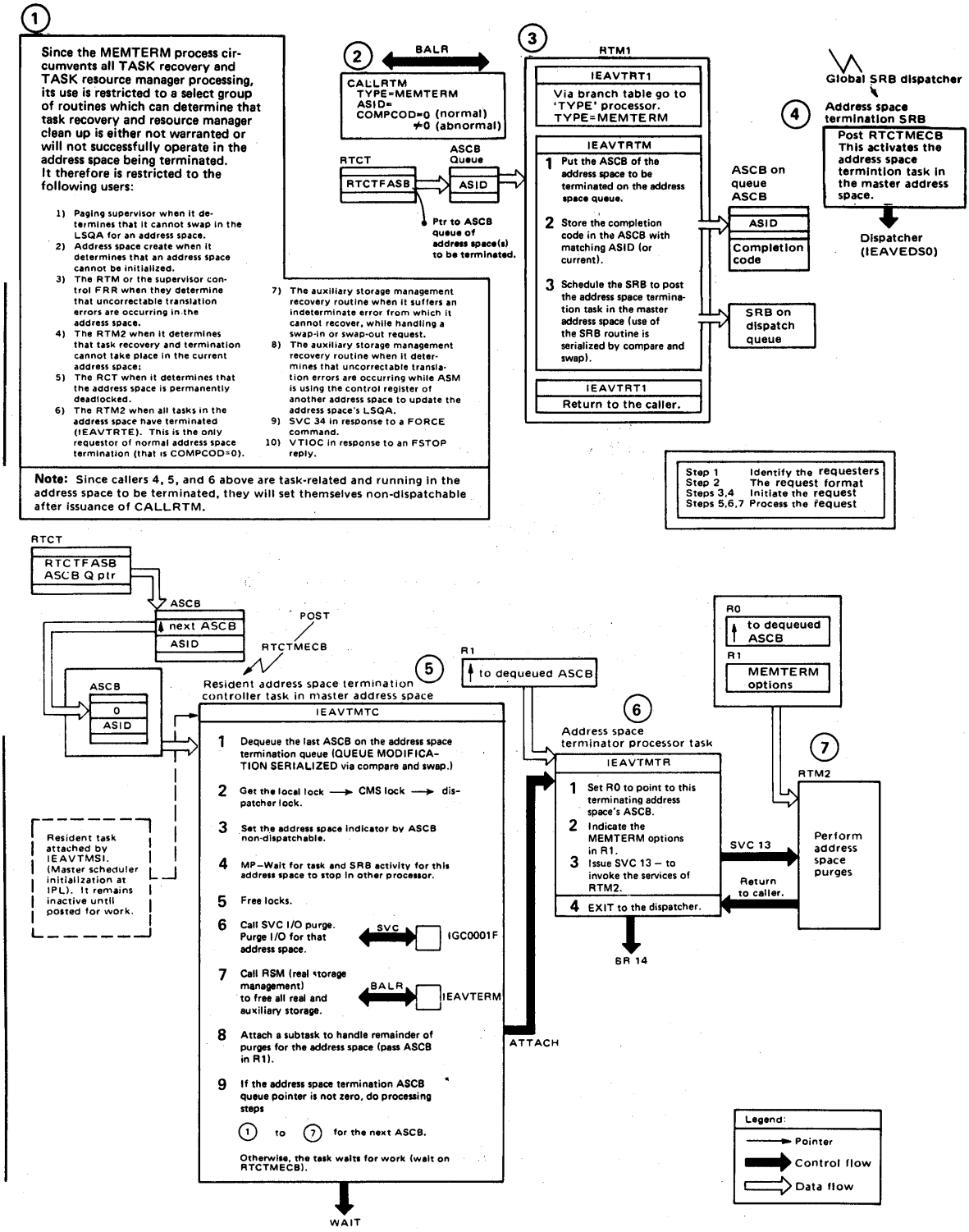
FORCE processing is dependent on the recovery termination manager (RTM), and on the command scheduling control block (CSCB), which contains a new bit definition in CHAFORCE. When the FORCE command is entered on a console having system authority, control is given to the CANCEL-FORCE processor which verifies 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: OS/VS2 MVS 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 here.



Error ID

Error ID ensures that problem information recorded in SYS1.LOGREC, can be easily correlated with 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 or ASIDLST 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 and ASIDLST 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).

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 SYS1.LOGREC, 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 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 name of the containing CSECT, and a description of the recovery processing.

Label	CSECT	Description
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.
SCHFRR	IEAVTSDX	FRR routine for branch-entry to SVC dump and for the timer disabled interrupt element (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.
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 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 SDWA for recording to SYS1.LOGREC is:

SDWAVRA – Contains the 24-byte recovery routine parameter area if DTESTAE1, SDESTAEX, SDFRRRTN, SUMFRFRR, or SUMFRR is the recovery routine name in SDWAREXN. 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').

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)
- 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.

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.

SVC Dump Work Area (SDWORK)

The SDWORK is pointed to by the RTCTSDPL field (at X'9C') in the RTCT. 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.

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.

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 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 space terminates during SVC dump processing, SVC dump's MEMTERM exit (IEAVTSDR) cleans up the resources related to that address space (such as setting tasks dispatchable). 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).

5.14.16 OS/VS2 System Programming Library: MVS Diagnostic Techniques

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 5-53 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 5-53. 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 UCM 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.

Refer to Figure 5-54 for the relationship of these control blocks.

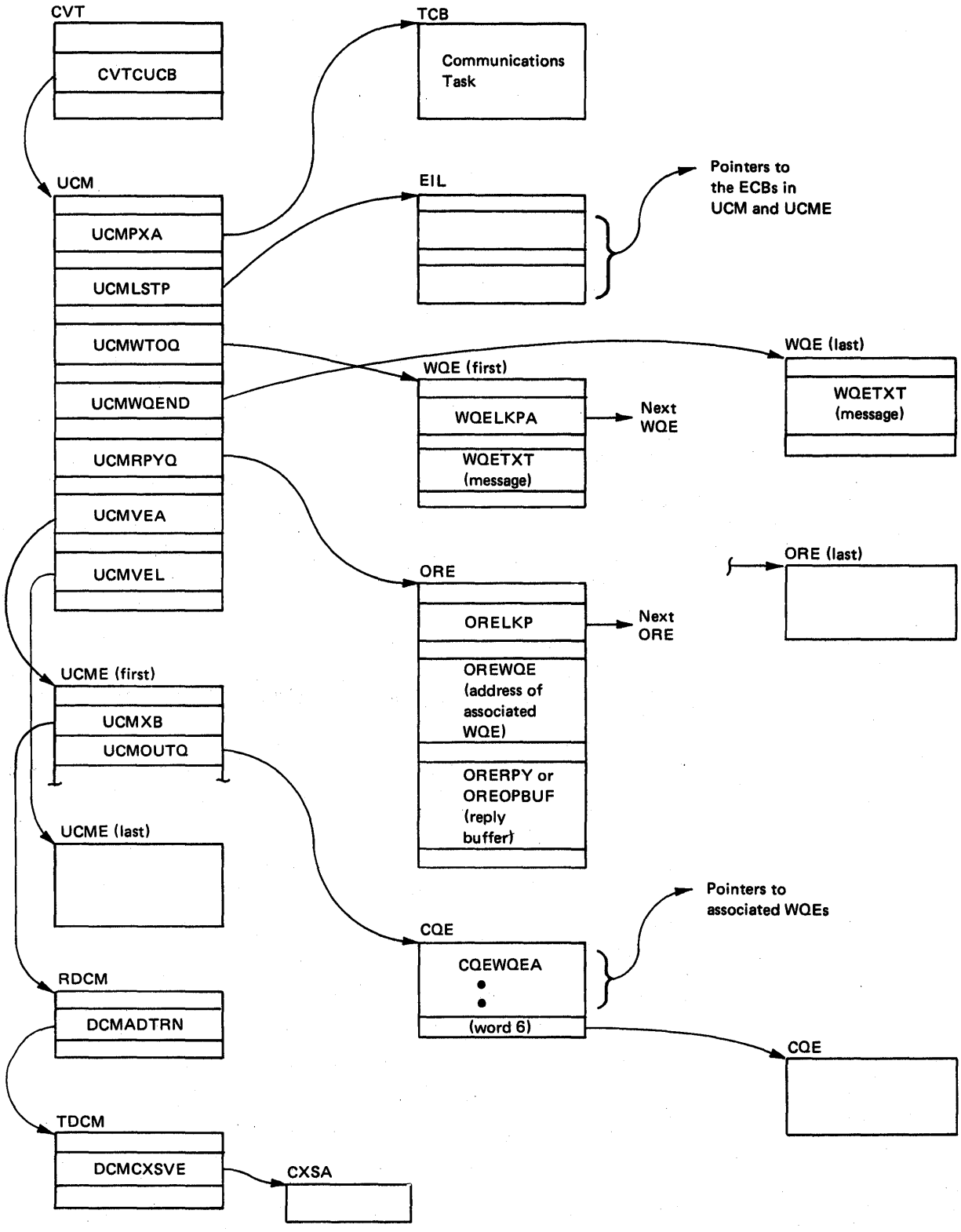


Figure 5-54. Communications Task Control Block Structure

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 5-53 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.
  - 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.

Communications Task (continued)

- 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 hard-copy 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.
    OREWQE – points to the WQE associated with this ORE.

Communications Task (continued)

- 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 six CQEs.
  - In each group of CQEs, the CQEWQEA field in the last (sixth) 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 six. The first five CQEs point to WQEs and the sixth points to the next group of CQEs.
  - In each group of CQEs, the CQEWQEA field in the first five 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.

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 IEECVETA) or an "F" module (such as IEECVFTA). 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).

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

A.1.2 OS/VS2 System Programming Library: MVS Diagnostic Techniques

This chapter describes the important aspects of the RSM component's page-fault processing. Figure A-1 outlines the major functions in this processing.

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.

IEAVGFA Tests

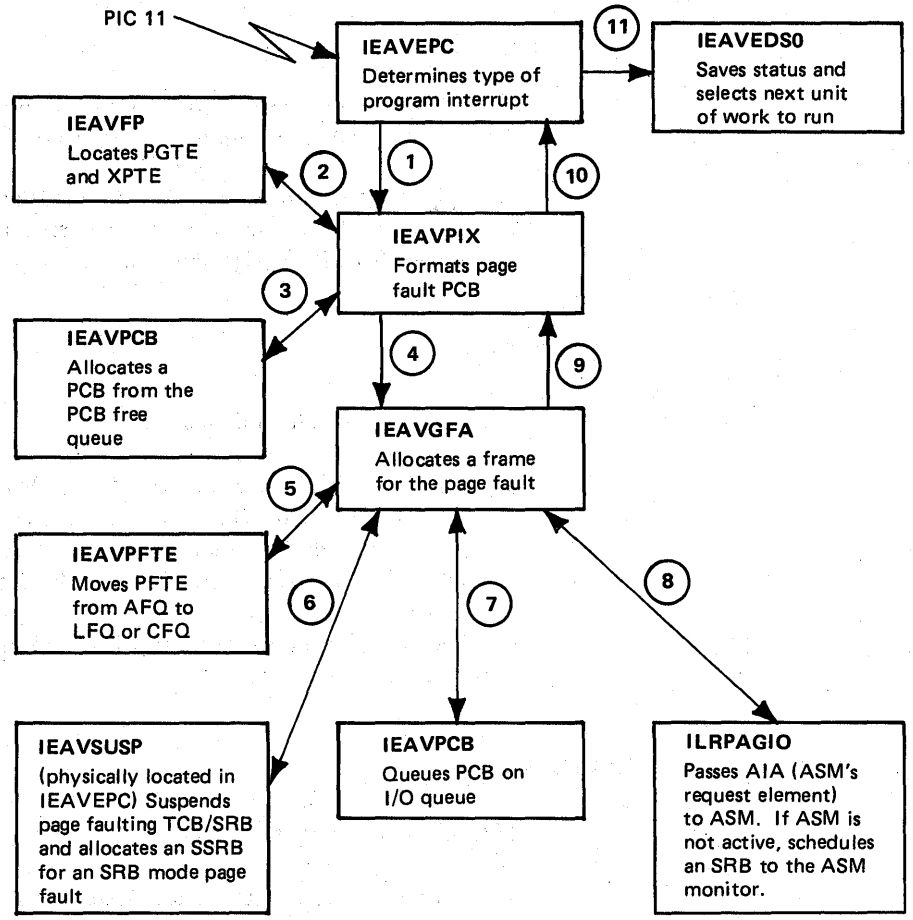
IEAVGFA performs the following:

- Checks XPTE to ensure that XPTDEFER is off. If it is on, some paging request (PCB) for this page has 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.

- Checks to see if RBNx from the PGTE is non-zero. If it is non-zero, the last used PFTE can be located. Once that PFTE is located, you can determine if the frame has been used for another purpose since last backing the requested VBN.

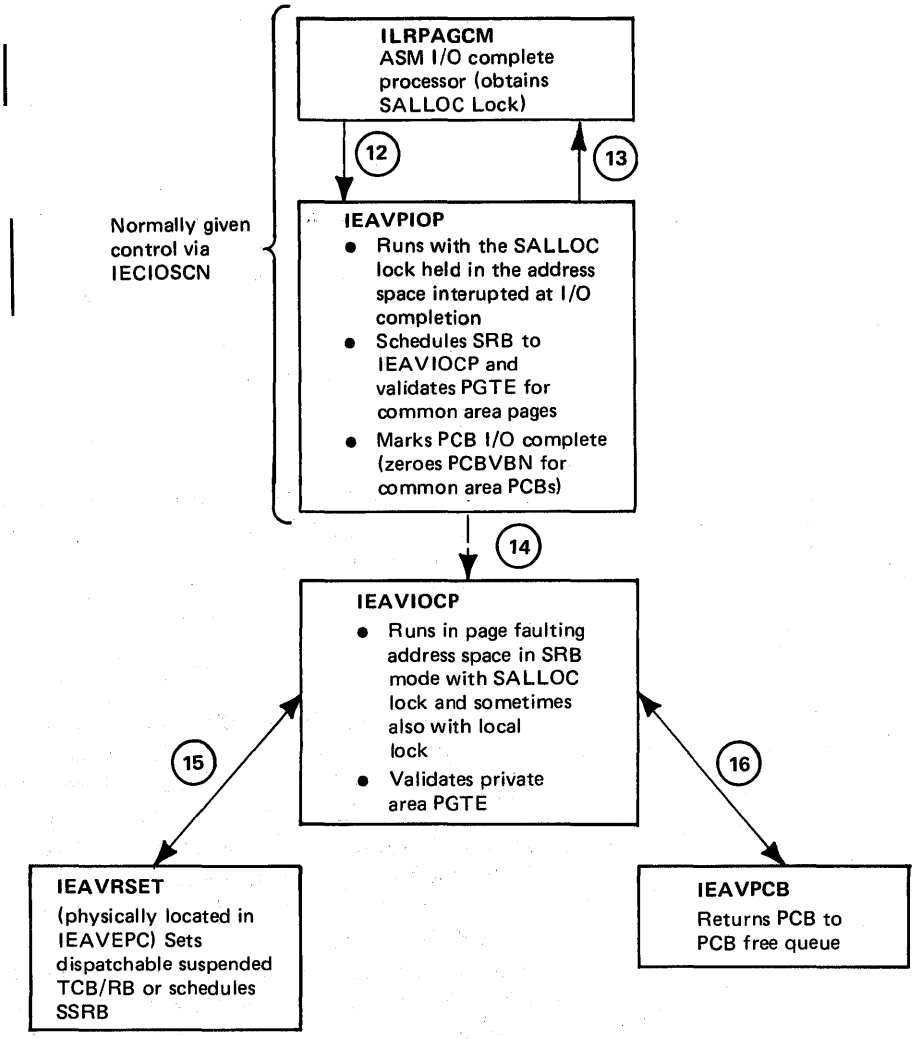
- Checks the XPTE for XPTFREL. If XPTFREL is on, the RBNx of the PGTE is zero and there is a paging request (PCB) on an I/O queue for this page that must be "related to." Because only a swap produces this condition, which applies only to stage 2 working set pages, this bit can only be on in private area XPTEs.

- If reclaim is successful, checks PFTONAVQ. If it is on, the reclaimed frame is available immediately; that is, no page I/O can be in progress for it. If it is off, the frame must be on either a local queue or the common queue and a PCB must be on an I/O queue.



Note: Circled numbers indicate the sequence of processing.

Figure A-1. Page Fault Process Flow (Part 1 of 2)



Note: Circled numbers indicate the sequence of processing.

Figure A-1. Page Fault Process Flow (Part 2 of 2)

- Checks to see if PFTPCBSI is zero. If it is, there is an inconsistency and IEAVGFA issues a C0D abend to record the error. If PFTPCBSI is on, the VBN0 value is used to select the PCB I/O queue to be searched and a PCB relate function is performed.

- If an old frame with or without I/O in progress cannot be found, 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 PCB on the PCB defer queue (PVTGFADF) of the PVT. The XPTDEFER flag is set in the XPTE.

- If a page-in is needed, the RBN0 of the allocated frame is placed 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 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 must be validated. IEAVPIOP validates common area PGTEs but must schedule IEAVIOCP to validate private area PGTEs because they are in the LSQA of the page-faulting address space. If IEAVPIOP validates the common area PGTEs, PCBVBN is set to 0 to prevent a second validation by IEAVIOCP. IEAVIOCP will be scheduled if PCBRESET=1. PCBRESET is still one unless the PCB has been NOPed.

IEAVIOCP Tests

IEAVIOCP runs in SRB mode and gets the local lock according to SRBPARM. SRBPARM is set earlier by IEAVOPBR (a subroutine of IEAVPIOP) if IEAVIOCP will need the local lock. IEAVOPBR is called from several places in RSM; its sole function is to determine if IEAVIOCP will need the local lock and to schedule IEAVIOCP.

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.

A.1.8 OS/VS2 System Programming Library: MVS Diagnostic Techniques

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 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 ASCBSTO
- 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.

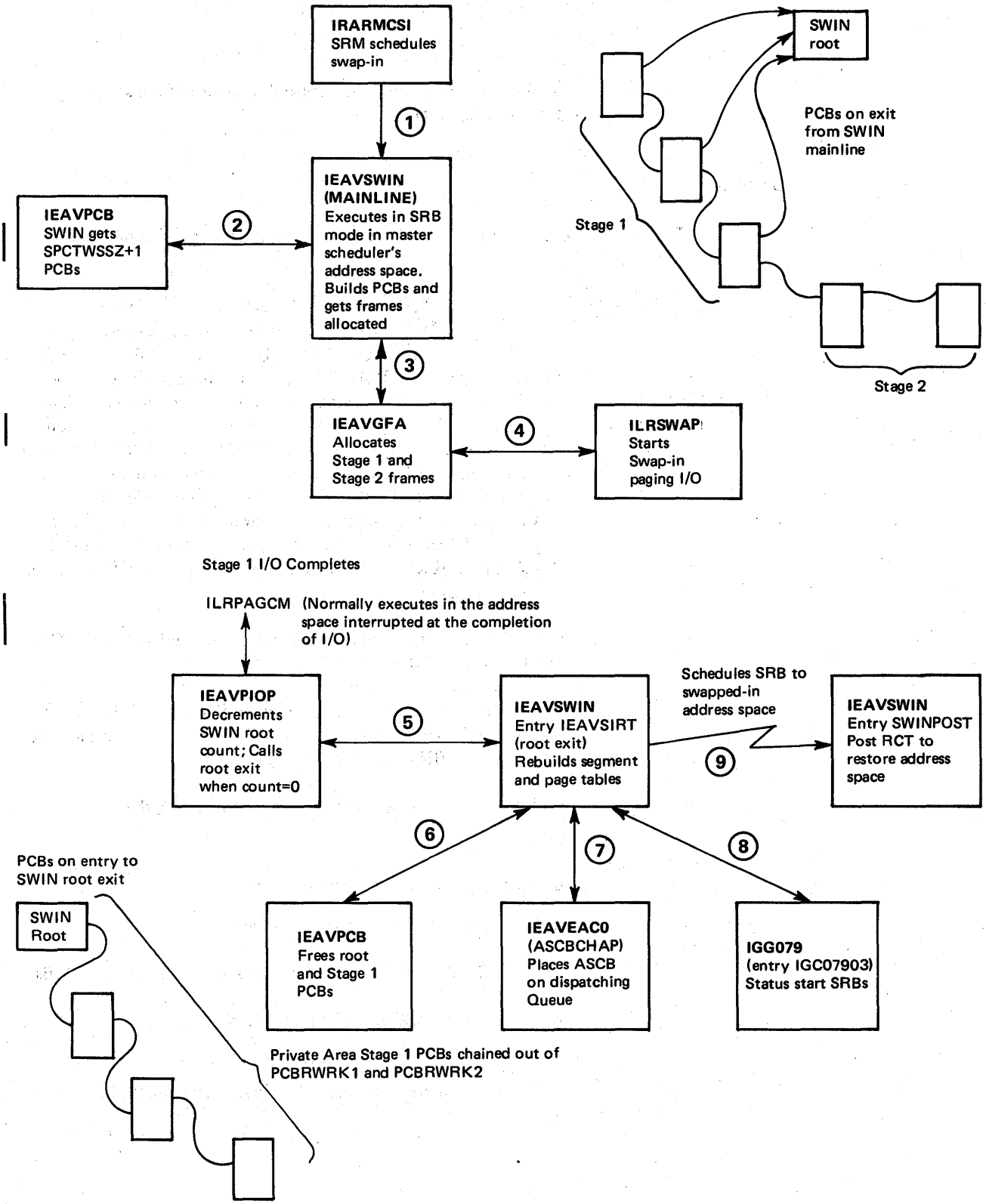


Figure A-2. Swap-In Process Flow

Swapping (continued)

(9) IEAVSIRT obtains an SRB from the RSM cell pool and schedules an entry point in IEAVSWIN (SWINPOST) into the swapped-in address space so it can post the region control task. SWINPOST posts RCT's ASCBQECB to restore the address space.

Note that stage two frames are allocated at the same time as stage one frames. The XPTFREL flag is on in each stage two PCB's corresponding XPTE. Then, if a page fault or other request reaches IEAVGFA prior to stage two I/O complete, IEAVGFA can relate the request to ongoing I/O (see the chapter on RSM in Section 4 for a discussion of RSM's relate functions). IEAVIOCP sets the XPTFREL flag to zero and fills in the PGTRSA field when stage two I/O completes. IEAVSOUT sets to 0009 all PGTEs for which it made a bit map entry in the SPCT.

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 quiesceable 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, 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.

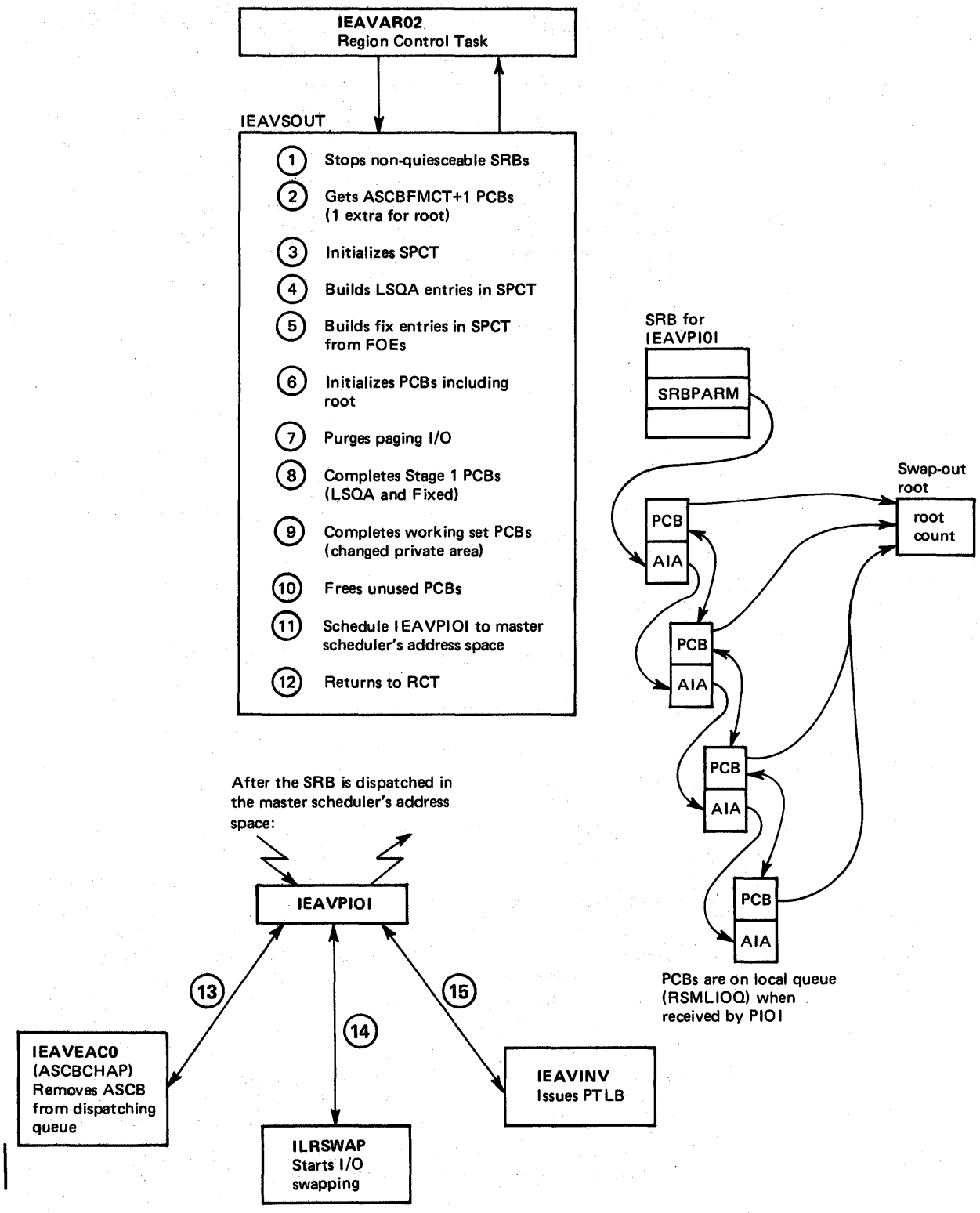


Figure A-3. Swap-out Process Flow

(9) IEAVSOUT next initializes a PCB for each changed page on the local frame queue and sets a bit in the bit map (SPCTBITM) for all pages that are not to be stolen. 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 dispatchable.
- 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.

(4) IEAVSOUT builds a two-byte LSQA entry for each frame on the LSQA frame queue.

(5) IEAVSOUT builds a four-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 and places the address of IEAVSORT in PCBRGOTO. It initializes the remaining PCBs, which might be used to swap-out a page as follows:

Partially initialized Swap-Out PCB

```
         FF  000000
         00  000000
         06  A(ROOT)   Root and output
         00  000000
         80  000000    PCBFREAL
         80  000000    Swap-out
         00  000000
         00  000000
         A(ASCB)
         00  000000
         00  000000
         18  C00000    Swap-out and write
AIA      00  000000
         00  000000
         00  000000
         00  000000
```

(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 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 RBNs and VBNs into PCBs for each LSQA or fix entry now in the SPCT. Even unchanged fixed or LSQA pages are paged out.

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.

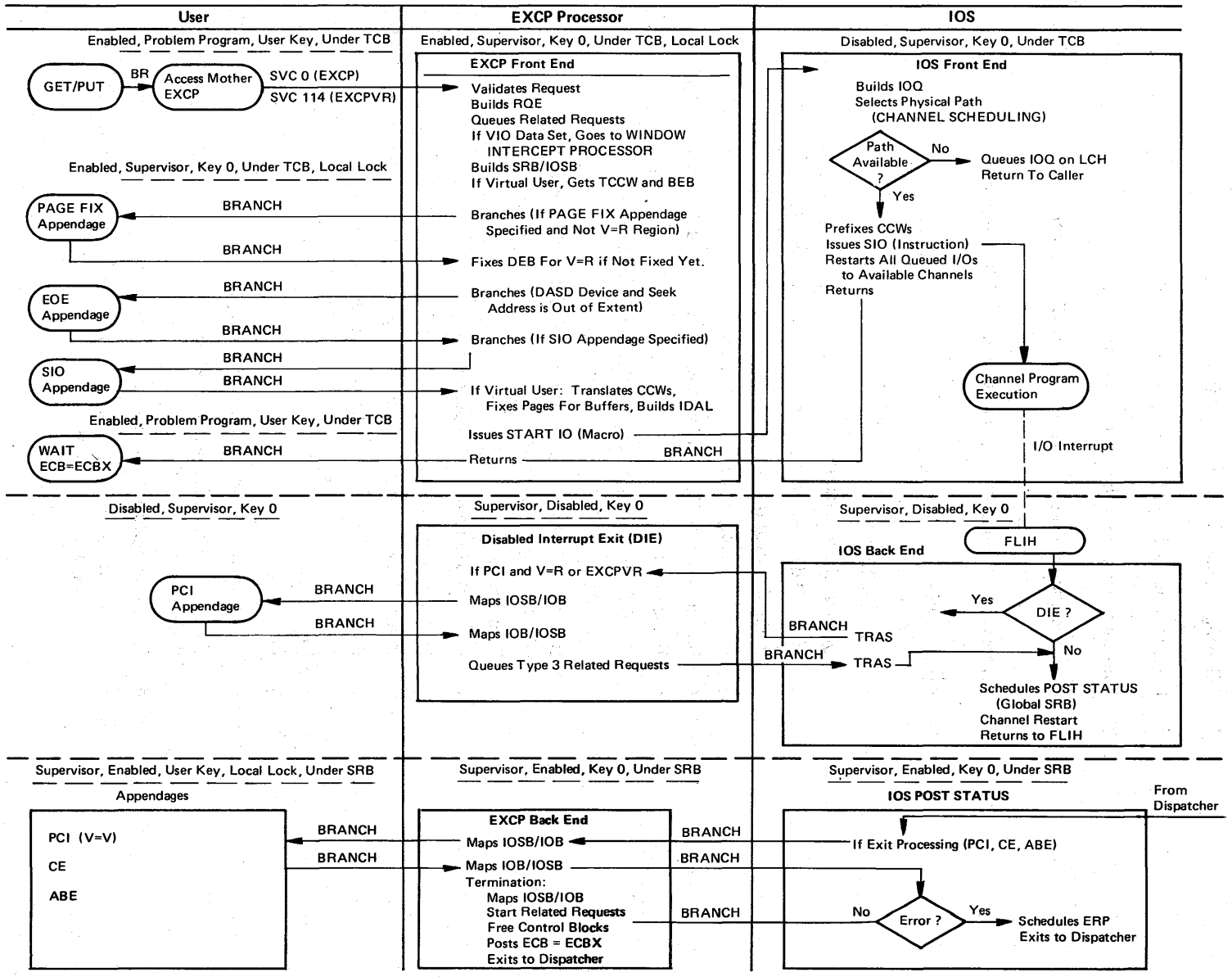


Figure A-4. IOS/EXCP Process Flow

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 to DIE.
      (6) Maps IOB to IOSB.
      (7) Queues type-3-related requests.
      (8) Branch returns to IOS front end.
      (9) TRAS (returns to addressability at time of interrupt).
   b. Schedules POST STATUS [global] (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.

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 exactly 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 IEAVGM00'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 CSPCHK to find the SPQE for the requested subpool. CSPCHK is a key routine for defining the characteristics of various subpools. For subpool 0, CSPCHK searches the TCBMSS chain. If no SPQE is found, CSPCHK returns a zero for the address of the SPQE and saves the address of the previous SPQE on the chain in SPQE SAVE.

   GMCOMM1 then calls routine QSPQESPC 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).

4. GSPQESPC passes control to (label) ROUND where 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 best-fit algorithm is used to find the smallest 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) or an internal freecell (FMAINC).

   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 CSPCHK to locate the SPQE. The associated DQEs are searched to locate the one DQE that describes the area to be freed.

3. 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, IEAVGM00 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 book-keeping, 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 by IEAVGM00 because freeing them might cause register save area overlays on the double recursion into FREEMAIN processing.

A.4.4 OS/VS2 System Programming Library: MVS Diagnostic Techniques

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, it 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 releases 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.

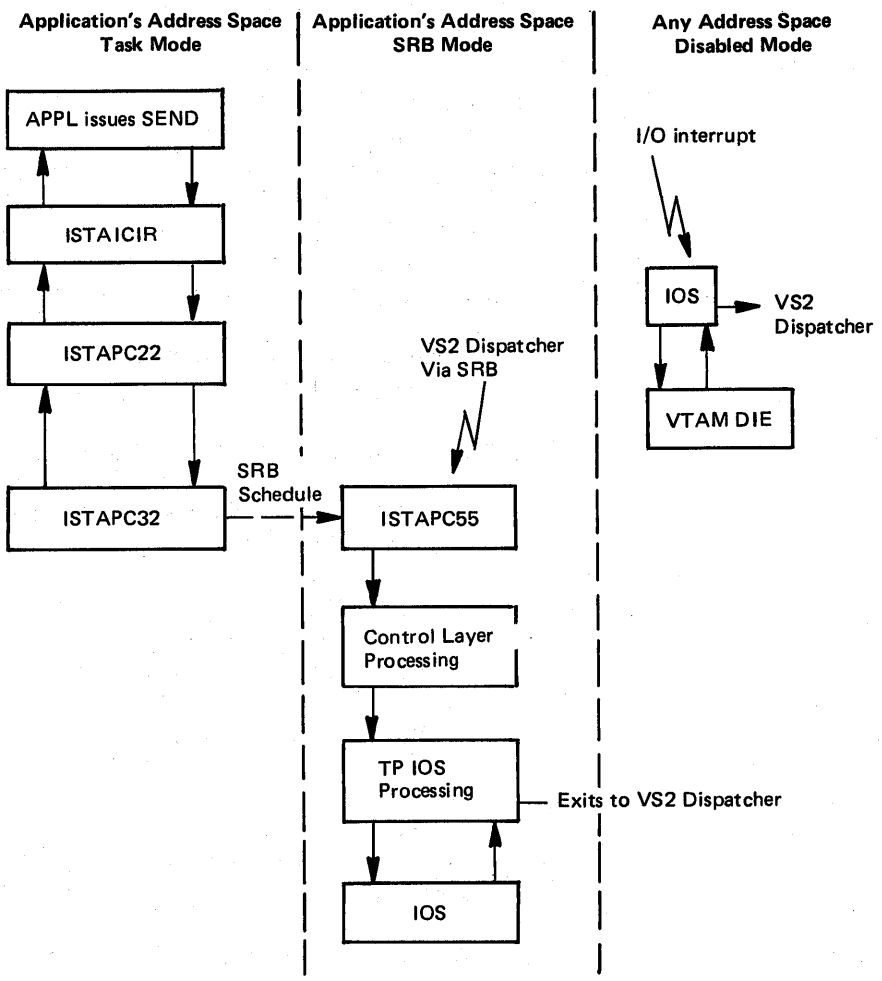


Figure A-5. VTAM SEND Process Flow

VTAM Process (continued)

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.

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.

VTAM Process (continued)

30. ISTAPC31 recognizes that there is nothing more to do and calls ISTAPC58.

31. ISTAPC58 frees the CRA and 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.

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:

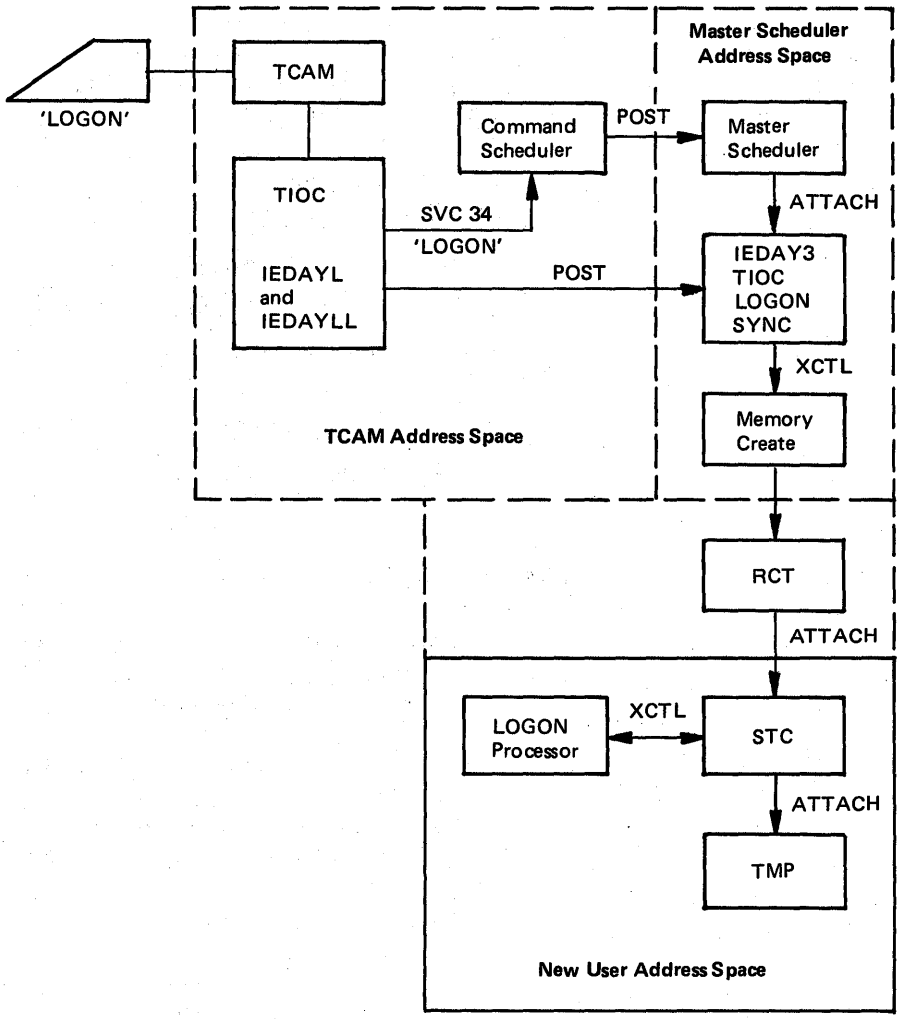
1. The SYS1.PARMLIB member IKJPRMxx is read to determine the TIOC buffer size and number, the maximum number of time sharing users allowed to be logged on at one time, and thresholds for the maximum number of TIOC buffers a single user can use at one time.

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).

Terminal User Issues LOGON



Note: Details of this process are shown in part 2 of this figure.

Figure A-6. Overview of Logon Processing (Part 1 of 2)

LOGON Scheduling

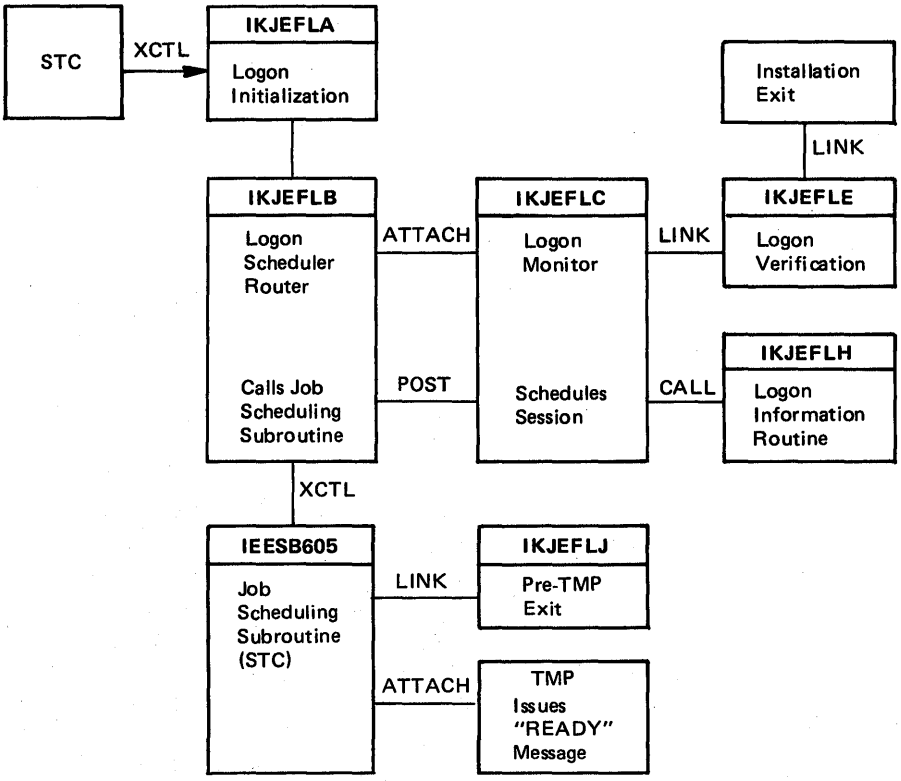


Figure A-6. Overview of Logon Processing (Part 2 of 2)

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).

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.

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).

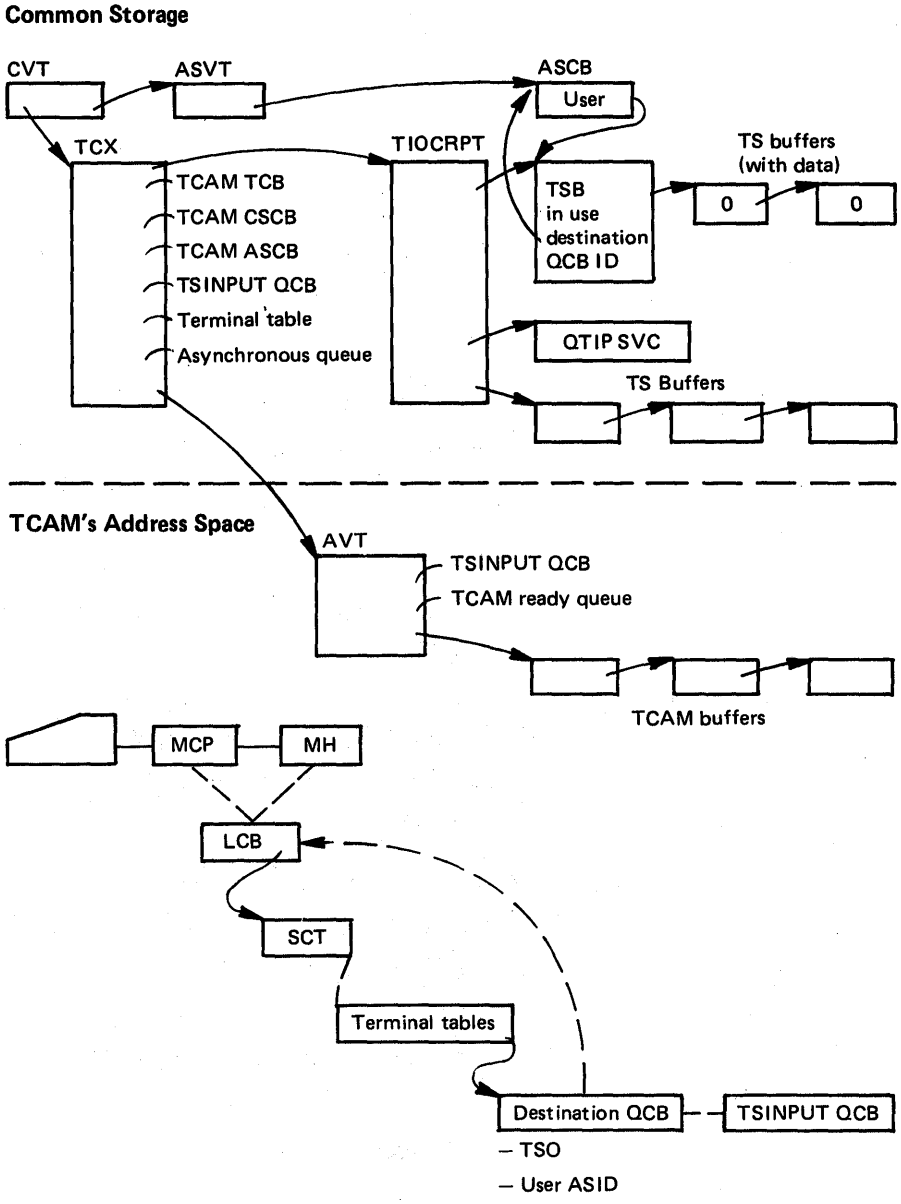


Figure A-7. TCAM Organization After a TSO Logon

Logon Initialization (IKJEFLA) (7): Logon initialization uses the address of the ASCB as input and does the following:

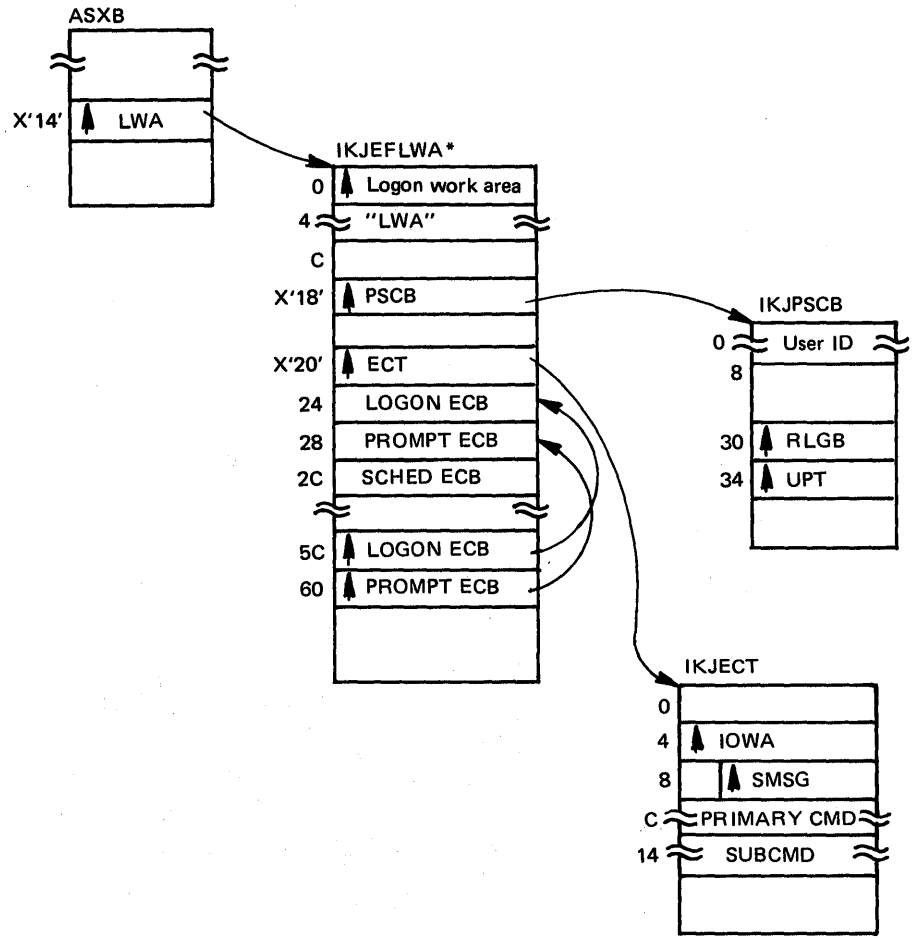
- Ensures SYS1.UADS and SYS1.BRODCAST data sets are allocated.
- Gets the LWA (logon work area) from the LSQA. (See Figure A-8.)
- Puts the LWA address in the ASXB.
- Gets the JSEL (job scheduling entrance list) from the LSQA.
- Puts the CSCB and ASCB addresses in the JSEL.
- Gets the JSXL (job scheduling 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.

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 the job is:
  - swappable
  - terminal job
  - cancellable
  - TSO



*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

TSO (continued)

- Gets the local and CMS locks.
- Puts the CSCB address in the ASCB.
- Frees the local and CMS locks.
- Calls MGCR to remove the old CSCB from the chain.
- Puts the new CSCB pointer in the JSCB and JSEL.
- Calls MGCR 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.
- 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.

TSO (continued)

- 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.

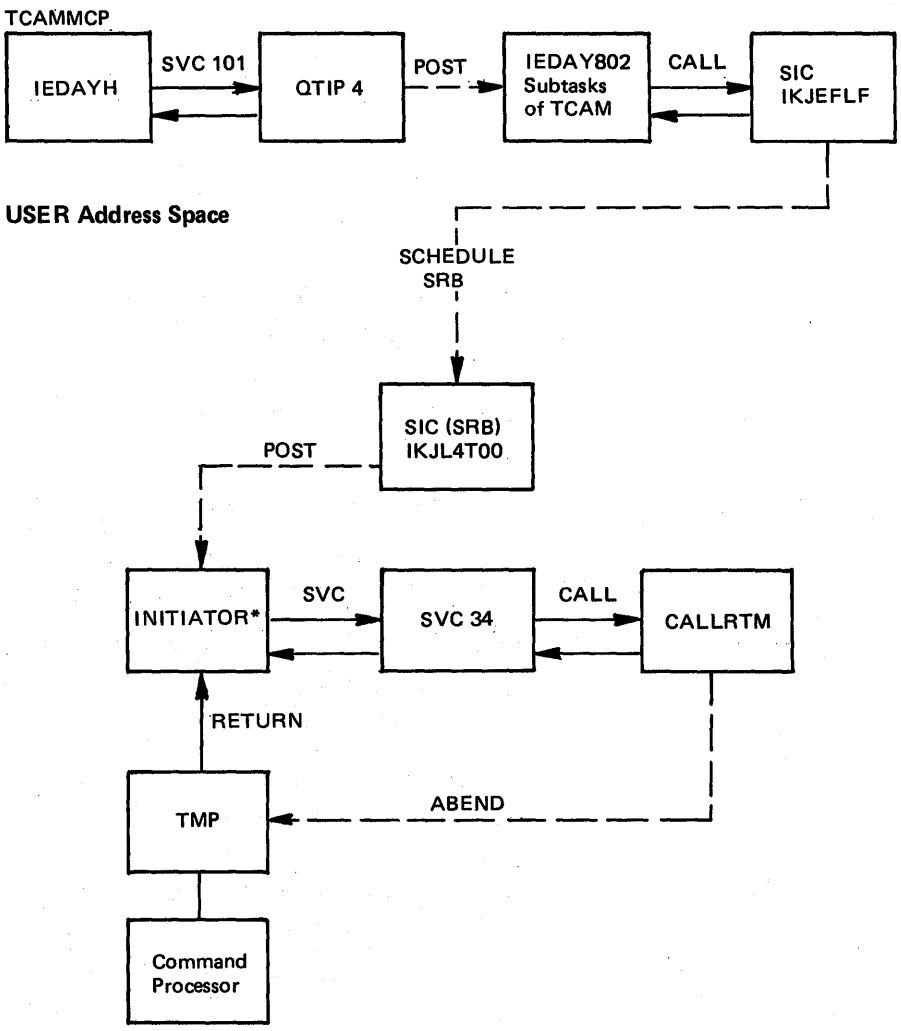
IEDAYH (Part of TCAM MCP):

- 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 limit.
  - 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 Dump 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-minute 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 PARM= 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 '$?' (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 '?' 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.):

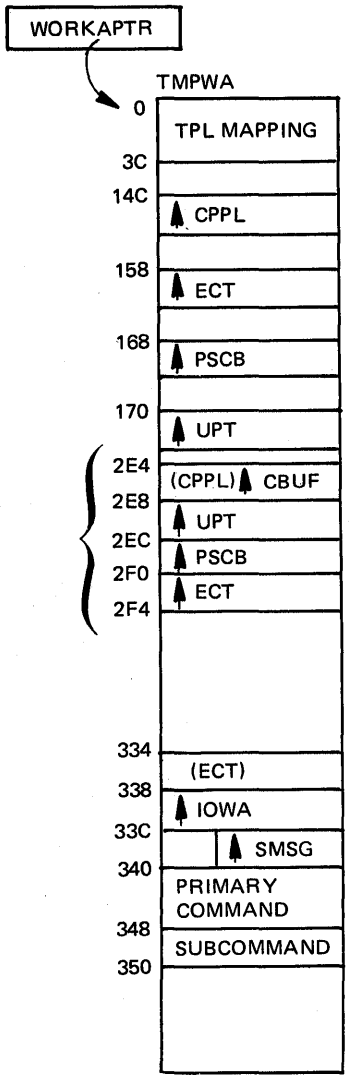
IKJTMPWA (TMP Work Area)

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.

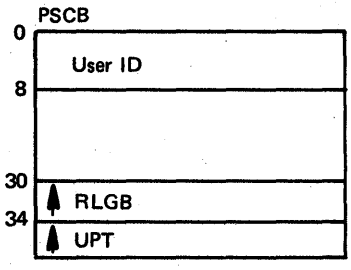
IKJPSCB (Protected Step Control Block)

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.)



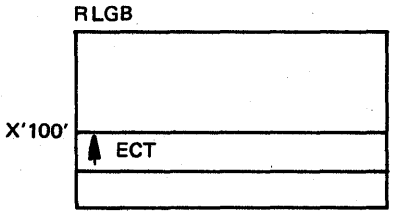
IKJRLGB (Re-Logon Buffer)

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.



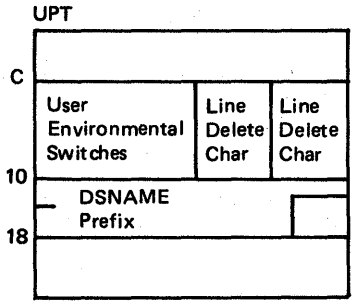
IKJUPT (User Profile Table)

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 IBM's 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 (*OS/VS2 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 on 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 on 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					IKJ56554I
		STAI				IKJ56554I
EDIT	ESTAE		√			See Note 1
LOGON	ESTAE		√	√		IKJ56452I
		ESTAI		√		IKJ56451I
						IKJ56452I
						IKJ56406I
OPERATOR	STAE		√			IKJ55004I
		STAI	√			IKJ55004I
OUTPUT	ESTAE			See Note 2	See Note 3	IKJ56318I
		STAI				
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 Input/Output 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 (SVC 93):* The TMP and command processors issue this SVC to move data from 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.

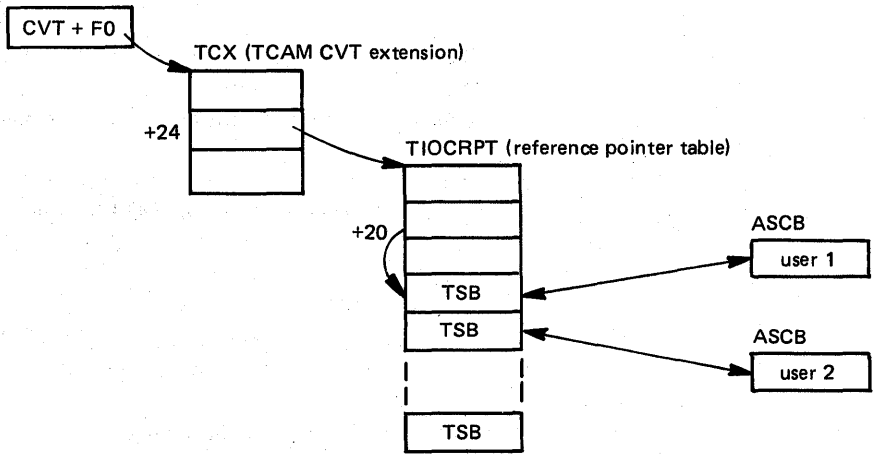
e. The TCAM buffer is routed to the buffer disposition routine for final processing.

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 *OS/VS2 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.

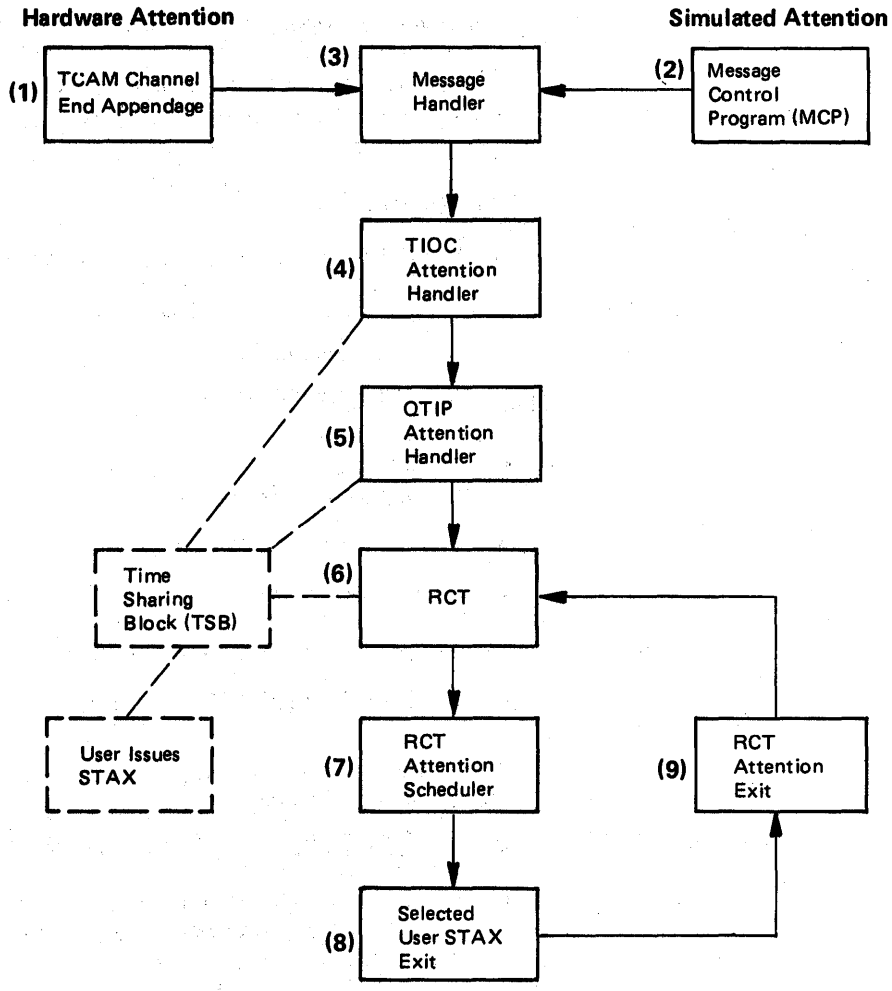


Figure A-13. TSO Attention Flow

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).

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 he detects:

- That the system has stopped in a solid wait state with a wait state code.
- What appears to him to be a system loop.
- That 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" will 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 runs as a task in the master scheduler's address space, usually represented by the third TCB in the formatted portion of the stand-alone dump; it 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 always looks like: X'00020000 00000xxx'

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.

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 the status could have been stored any time and not necessarily just before the dump.

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 although the stored status is put in the PSA +X'100' and the registers are put at PSA + X'160-1FF', the 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 the other processor to store status. The STORE STATUS command always stores in the normal PSA at location zero. This means that the normal PSA will contain the registers and PSW from the other 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 normal 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 = 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. PSA + X'101' = 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

Check the PSA for a low storage overlay. Critical fields are the CVT pointer at X'10', the new PSW locations at location X'58-78' and at location X'00', and the trace table pointer at location X'54'.

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. These locations never contain valid data.

If the PSW reflects the wait bit and does not have a zero address and if the STORE STATUS registers are zero, check location X'300'. Is it equal to the wait state PSW ? If so, it is possible some task scheduled a bad SRB. Examine the trace table for the SRB dispatch. Register 0's position in the trace table is a pointer to the SRB. The previous address space before the SRB dispatch is the possible scheduler of the SRB. Another possibility is an overlaid RB or LCCA. What does the last entry in the trace table reflect – SRB or task dispatch ? Make sure that the trace table was not stopped by the dump task. Check for an X'80' in the high order byte of the CPUID field.

Loops can be either disabled or enabled. The best way of determining which has occurred is by noting the address of the loop if the operator recorded it before taking the SADMP.

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. ***Caution:*** If the only addresses the operator furnished 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 the other 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 the other 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.

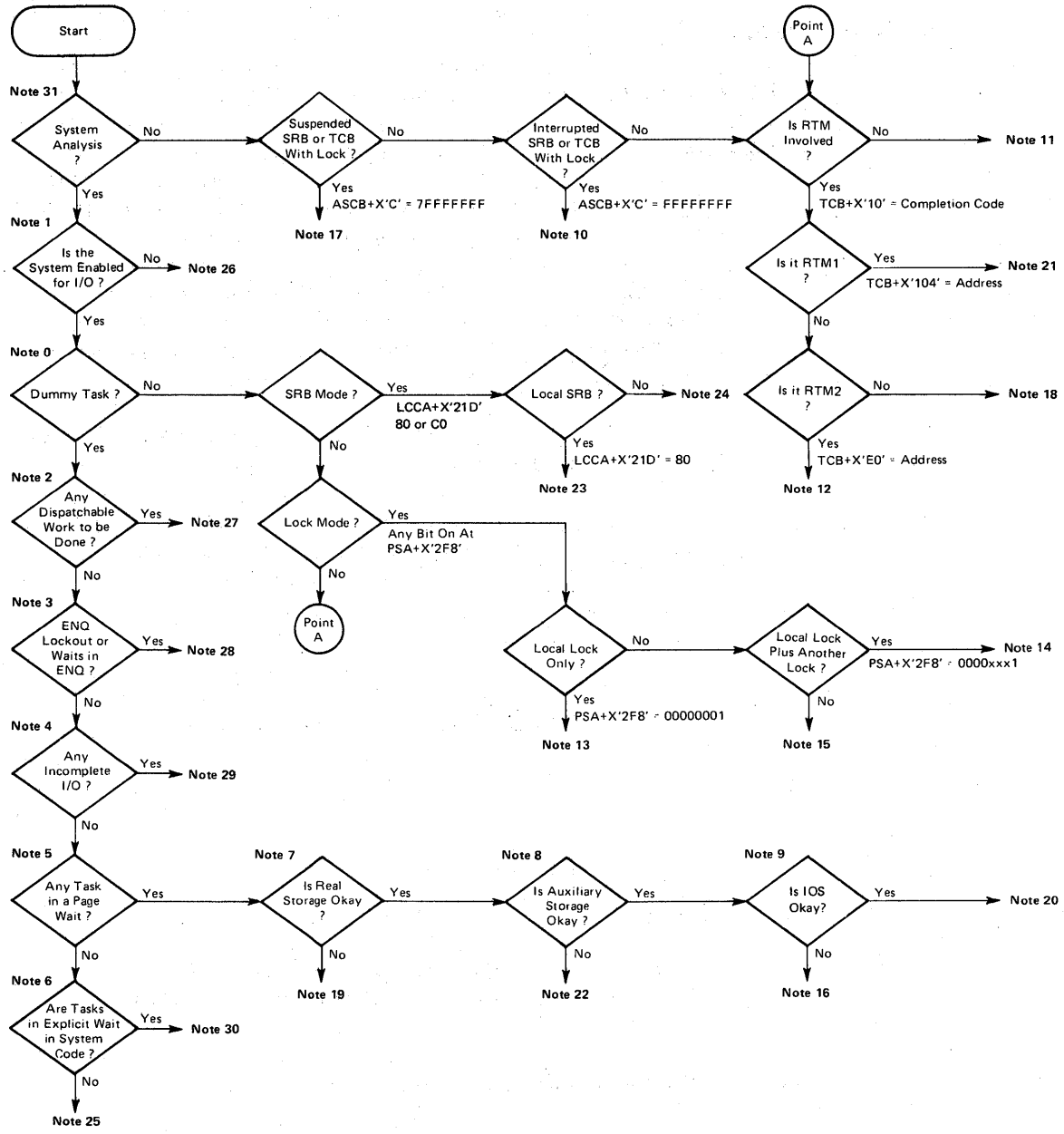


Figure B-1. Stand-alone Dump Analysis Flowchart

Analysis Procedure

The following explanations correlate to the "Notes" in Figure B-1.

Note 0 – Dummy Task ?

The enabled wait generally occurs as a result of the lack of some critical system resource.

If the PRINT statement of PRDMP is used properly (see the chapter "Additional Data Gathering Techniques" in Section 2), the message "CURRENT TASK = WAIT TASK" appears in the formatted portion of the dump.

PSA + X'218/21C' is the new/old TCB pointer. PSA + X'220/224' is the new/old ASCB pointer.

The ASCB with an ASID=0 is the dummy ASCB. If the dummy task is the current task, go to the TCB summary before going to the next block and check whether any task has an error completion code. If any TCBs are abending, continue at Point A and start with the top three address spaces if they have a completion code.

If the ASCB is the dummy ASCB but the TCB new/old pointers are zero, then take the "no" path and check for SRB mode.

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+C=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+80 = 00000000 or FFFFFFFF
   ASCB+66, 67, 72, 73 = 00
   ASCB+7C = some value or
   ASCB +1C = the service priority list has an SRB queued.

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 service manager queue and the global service priority list.

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 7FFFFFFF in the lockword keeps the dispatcher from dispatching. Most of the flags show the reason for not dispatching.

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 to print the QCBs and check for exclusive enqueues.

2. The CVT+X'280' points to the first major QCB.
   Major QCB+8 points to the first minor QCB.
   Minor QCB+0 points to the next minor QCB.
   Minor QCB+8 points to the first QEL.

3. Any QEL reflecting exclusive control or reserve status 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+1C = wait count
RB-8 ≠ 40 (FLAG1)

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+4 = available frame count
   PVT+X'24' = free PCB count
   PVT+X'140' = deferred for lack of free page frames
   PVT+X'148' = requests sent to ASM

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 auziliary 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'50' = the number of started I/O requests that have not completed
4. ASMVT+X'54' = indicates whether the ASM's SRB for ILRPTM (ASM PART monitor) has been scheduled

If the number of paging I/O requests completed is equal to the number of paging I/O requests received, the ASM has no outstanding work. However, if these counts differ, check the other status indicators for the following:

1. If I/O requests have been started but not completed, determine what has happened to the I/O.
2. If ASM's SRB for ILRPTM has been scheduled, determine what the dispatcher has done with the SRB.

Note 9 – Is IOS Okay ?

If the number of started I/O requests that have not completed (ASMVT+X'50') is zero, then IOS has completely processed all the I/O that ASM has started.

Note 10 – Interrupted SRB or TCB ?

The condition that caused the SRB to be suspended has been resolved. The suspended SRB (SSRB) is queued on the SPL at the non-quiesceable level.

The condition that caused the TCB holding the local lock or local and and CMS locks 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 and CMS locks has been interrupted by a higher priority task. The save area used for re-dispatching 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 stand-alone dump reflects the same current task, 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.

If this is a system dump, the TCB belongs to a non-abending sister, mother, or daughter task. Find the task that has a completion code (by checking the TCB summary) and continue at Point A.

Note 12 – RTM2, Yes.

The most important place to find information about abend codes is *OS/VS Message Library: VS2 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'). If there is no space in LSQA to build the work area, SQA is used.

Normally 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'C' contains the CPU ID. The current TCB+X'110' 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 (ASCB+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, the other processor could be causing the loop by not freeing a spin lock that it is currently holding. *Note:* The failure to free or obtain a lock can be caused by the lockword being overlaid on either an MP or UP.

If both 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.

Within the lock manager code, register 12 contains the bit mask for the locks-held table in the PSA (PSA+X'2F8'). Register 11 can contain the address to the lockword itself and register 14 contains the return address of the requestor.

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 contains the CPU ID. The current TCB+X'110' contains the CPU ID. The loop is within this task if only the local and CMS locks are held. 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+6C) – if the task abended while holding local and CMS locks
- 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'254'. For the normal stack, the global SDWA immediately follows the RESTART super stack SDWA.

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.

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 or a request for a CMS lock when it is being held by another processor. 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 CMS lock request, the SSRB is on the CMS lock-suspended queue, which can be located in IEANUC01 at label CMSSRBF. (See AMBLIST of IEANUC01.)

A locked TCB can be suspended for the same reasons as an SRB. The save area is the IHSA (described in the *Debugging Handbook*). The IHSA is valid during a page fault if the corresponding PCB+8 flag is on. The IHSA is valid for a CMS lock suspend if the ASCB is on the CMS lock suspend queue in IEANUC01 at label CMSASBF.

The TCB can be suspended because of a page fault while holding the local lock and the CMS lock. A difference would be that the ASCB+X'67' flag for the CMS lock is turned on. See the chapter "Dispatcher" in Section 5 and the chapter "System Execution Modes and Status Saving" in Section 2 of this manual.

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 is 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 under RT1W 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 life span, 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' or X'C34' indicates an empty stack. Any address between X'C54' and X'E34' 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. If any entry on the stack points to RTM or ABDUMP's FRR, it is almost certain that system damage has occurred in a SADMP. This is normal in an SVC dump.

If there is an address at either X'C44' or X'C48', there has been an entry into RTM1 and an RTCA (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'C44' or C'48') 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') differs from the count of I/O requests completed (ASMVT+X'2C'), and the number of started I/O requests that have not completed (ASMVT+X'50') 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. AIA–X'28' = part of PCB which contains RSM-related data
2. ASMVT+X'20-24' = queue of AIAs waiting for IOEs
3. The I/O request element (IOE) which points to the AIA is queued to one of the following PART queues:
   PART+X'30-34' = common write queue
   PART+X'38-3C' = spill write queue
   PART+X'40-44' = duplex write queue
   PART+X'48-4C' = local write queue
   each PART entry contains an unsorted read queue (X'C') and a sorted read queue (X'30').
4. Each active IORB (PART entry+X'1C') contains a chain of PCCWs (IORB+X'8'). Each of these PCCWs points to an AIA (PCCW+X'8').
5. 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 the other 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 the other 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 "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 the area of the problem. For example, if Flags4 = X'04', this indicates a VARY or QUIESCE command is in process on an MP system; Flags5 = 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, a SLIP trap (wait state code 01B), 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. (The other 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.

PSA + X'2F8' – PSAHLI. 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.

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 first entry.

The recording buffer (LRB) + 4 = pointer to the last entry.

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: Abbreviations

ABP	–	Actual block processor
ACA	–	ASM control area
ACB	–	Access method control block
ACE	–	ASM control element
ACP	–	Automatic command processing
ACR	–	Alternate CPU recovery
ACT	–	Account control table
ADA	–	Automatic data area
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
ASVT	–	Address space vector table
ASXB	–	Address space extension block
ATA	–	ASM tracking area
AVT	–	TCAM address vector table
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
CFQ	–	Common frame queue
CHAP	–	Change priority
CI	–	Control interval
CIDF	–	Control interval definition field
CMB	–	Console message buffer
CMP	–	Completion field

Abbreviations (continued)

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
CSA – Common storage area
CSCB – Command scheduling control block
CSD – Common system data area
CTGPL – Catalog parameter list
CVT – Communications vector table
| CXSA – Communications extended save area

DAT – Dynamic address translation
DAVV – Direct access volume verification
DCB – Data control block
| DCM – Display control module
DCT – Device control table
DDRCOM – Dynamic device reconfiguration communication table
DE – Directory entry
DEB – Data extent block
DECB – Data event control block
| DIDOCS – Device independent display operators console support
DIE – Disable interrupt exit
DIR – Deferred incident record
DMDT – Domain descriptor table
DMVT – Domain vector table
DQE – Descriptor queue element
DRQ – Data ready queue
DSAB – Data set association block
DSCB – Data set control block
DSPCT – Data set page correspondence table
DVT – Destination vector table

ECB – Event control block
ECC – Error checking and correction
ECT – Environment control table
EDB – Extent descriptor block
EDL – Eligible device list
EED – Extended error descriptor
| EIL – Event indication list
EIP – EXCP intercept processor
EMS – Emergency signal
EOA – End of address
EP – Emulator program
| EPATH – Error path (recovery audit trail area)
EPS – External page storage

Abbreviations (continued)

ERP	–	Error recovery procedures
ERPIB	–	Error recovery procedures interface block
ESTAE	–	Extended STAE
ESTAI	–	Extended STAI
EVNT	–	Event table
EWA	–	Common ERP work area
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
FSB	–	Feedback status block
FVT	–	Field vector table
GDA	–	Global data area
GPR	–	General purpose register
GSMQ	–	Global service manager queue
GSPL	–	Global system priority list
GSR	–	Global shared resource
GTF	–	General Trace Facility
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
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

Abbreviations (continued)

JCL – Job control language
JCT – Job control table
JES – Job Entry Subsystem
JESCT – JES control table
JFCB – Job file control block
JFCBX – Job file control block extension
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 – Logical configuration communication area 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
LPDE – Link pack directory entry
LPID – Logical page identifier
LPME – Logical to physical mapping entry (or) logical page mapping entry
LRB – Logrec buffer
LSID – Logical slot ID
LSMQ – Local service manager queue
LSPL – Local service priority list
LSQA – Local system queue area
LUB – NCP logical unit block
LWA – Logon work area

MCH – Machine check handler
MCIC – Machine check interrupt code
MCP – Message control program
MCS – Multiple console support

Abbreviations (continued)

MFA	–	Malfunction alert
MH	–	Message handler
MIH	–	Missing interrupt handler
MLPA	–	Modified link pack area
MP	–	Multiprocessing
MPST	–	Memory process scheduling table
MSS	–	Mass storage subsystem
MVS	–	Multiple Virtual Storage
MWA	–	Module work area
NCP	–	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
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
PDDB	–	Peripheral data definition block
PDS	–	Partitioned data set
PEP	–	Partitioned emulator program
PER	–	Program event recording
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
PQE	–	Partition queue element
PRB	–	Program request block
PSA	–	Prefixed save area

Abbreviations (continued)

PSAHLHI – PSA highest lock held indicator
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
PWKA – Paging work area

QAB – Queue anchor block
QCB – Queue control block
QEL – Queue element

RACF – Resource Access Control Facility (Program Product)
RB – Request block
RBA – Relative byte address
RBN – Real block number
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
RIM – Resource initialization module
RJE – Remote job entry
RMCT – Resource manager control table
RMF – Resource Management Facility (Program Product)
RMS – Recovery management support
RPH – Request parameter header
RPL – Request parameter list
RQE – Request queue element
RSM – Real storage manager
RSMHD – RSM header
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)
SART – Swap activity reference table
SAST – Subsystem allocation sequence table
SAT – Swap allocation table
SCCW – Swap channel control work area
SCT – Step control table
SDWA – System diagnostic work area
SGT – Segment table
SGTE – Segment table entry
SIC – System initiated cancel
SIGP – Signal processor
SIO – Start input/output

Abbreviations (continued)

SIOT	–	Step I/O table
SLIH	–	Second level interrupt handler
SMF	–	System measurement facility
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
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	–	Started task control
STCB	–	Subtask control block
STM	–	Store Multiple instruction
STOR	–	Segment table origin register
SVC	–	Supervisor call
SVRB	–	Supervisor request block
SWA	–	Scheduler work area
TCAM	–	Telecommunications Access Method
TCB	–	Task control block
TCH	–	Test channel
TCX	–	TCAM
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

Abbreviations (continued)

| 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
VIO – Virtual I/O
VSAM – Virtaul 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
WTQE – Wait queue element

XL – Extent list
XPTE – External page table entry

Index

I.1.10 OS/VS2 System Programming Library: MVS Diagnostic Techniques

OS/VS2 System Programming Library:
MVS Diagnostic Techniques
GC28-0725-2

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. This form may be used to communicate your views about this publication. They will be sent to the author's department for whatever review and action, if any, is deemed appropriate.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

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 comments are:

Clarity Accuracy Completeness Organization Coding Retrieval Legibility

If comments apply to a Selectable Unit, please provide the name of the Selectable Unit _____.

If you wish a reply, give your name and mailing address:

_____

_____

_____

Please use pressure sensitive or other gummed tape to seal this form.

Cut or Fold Along Line

Please circle the description that most closely describes your occupation.

Customer	(Q) Install Mgr.	(U) System Consult.	(X) System Analyst	(Y) System Prog.	(Z) Applica. Prog.	(F) System Oper.	(I) I/O Oper.	(L) Term. Oper.	(O) Other

IBM	(S) System Eng.	(P) Prog. Sys. Rep.	(A) System Analyst	(B) System Prog.	(C) Applica. Prog.	(D) Dev. Prog.	(R) Comp. Prog.	(G) System Oper.	(J) I/O Oper.	(E) Ed. Dev. Rep.	(N) Cust. Eng.	(T) Tech. Staff Rep.

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.)

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 706-2
PO Box 390
Poughkeepsie, New York 12602

Fold and tape | Please Do Not Staple | Fold and tape

IBM

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, N.Y. 10604

IBM World Trade Americas/Far East Corporation
Town of Mount Pleasant, Route 9, North Tarrytown, N.Y., U.S.A. 10591

IBM World Trade Europe/Middle East/Africa Corporation
360 Hamilton Avenue, White Plains, N.Y., U.S.A. 10601