LY28-1735-0
File No. S370-36

Program Product

MVS/Extended Architecture System Logic Library: Recovery Termination Management

MVS/System Product:

JES3 Version 2	**5665-291**
JES3 Version 2	**5740-XC6**

This publication supports MVS/System Product Version 2 Release 2.0, and contains information that was formerly presented in MVS/Extended Architecture System Logic Library Volume 11, LY28-1246-2, which applies to MVS/System Product Version 2 Release 1.7. See the Summary of Amendments for more information.

First Edition (June, 1987)

This edition applies to Version 2 Release 2.0 of MVS/System Product 5665-291 or 5740-XC6 and to all subsequent releases until otherwise indicated in new editions or technical newsletters. Changes are made periodically to the information herein; before using this publication in connection with the operation of IBM systems, consult the latest IBM System/370 Bibliography, GC20-0001, for the editions that are applicable and current.

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

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

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

PREFACE

The MVS/Extended Architecture System Logic Library is intended for people who debug or modify the MVS control program. It describes the logic of most MVS control program functions that are performed after master scheduler initialization completes. For detailed information about the MVS control program prior to this point, refer to MVS/Extended Architecture System Initialization Logic. For general information about the MVS control program and the relationships among the components that make up the MVS control program, refer to the MVS/Extended Architecture Overview. To obtain the names of publications that describe some of the components not in the System Logic Library, refer to the section Corequisite Reading in the Master Preface in MVS/Extended Architecture System Logic Library: Master Table of Contents and Index.

HOW THE LIBRARY IS ORGANIZED

SET OF BOOKS

The System Logic Library consists of a set of books. Two of the books provide information that is relevant to the entire set of books:

1. The MVS/Extended Architecture System Logic Library: Master Table of Contents and Index contains the master preface, the master table of contents, and the master index for the other books in the set.
2. The MVS/Extended Architecture System Logic Library: Module Descriptions contains module descriptions for all of the modules in the components documented in the System Logic Library and an index.

Each of the other books (referred to as component books) in the set contains its own table of contents and index, and describes the logic of one of the components in the MVS control program.

ORGANIZATION OF THE COMPONENTS

Most component books contain information about one component in the MVS control program. However, some component books (such as System Logic Library: Initiator/Terminator) contain more than one component if the components are closely related, frequently referenced at the same time, and not so large that they require a book of their own.

A three or four character mnemonic is associated with each component book and is used in all diagram and page numbers in that book. For example, the mnemonic ASM is associated with the book MVS/Extended Architecture System Logic Library: Auxiliary Storage Management. All diagrams in this book are identified as Diagram ASM-n, and all pages as ASM-n, where n represents the specific diagram or page number. Whenever possible, the existing component acronym is used as the mnemonic for the component book. The Table of Book Titles in the Master Preface in MVS/Extended Architecture System Logic Library: Master Table of Contents and Index lists the book titles, the components included in each book (if a book contains more than one component), the mnemonics for the books, and the order number for each book.

HOW TO USE THE LIBRARY

To help you use this library efficiently, the following topics cover

- How to find information using book titles and the master index
- What types of information are provided for each component
- How to obtain further information about other books in the System Logic Library

FINDING INFORMATION USING THE BOOK TITLES

As you become familiar with the book titles, MVS component names and mnemonics, and the book contents, you will be able to use the System Logic Library as you would an encyclopedia and go directly to the book that you need. We recommend that you group the books in alphabetical order for easy reference, or, if you are familiar with MVS, that you to group the books by related functions.

The Table of Book Titles in the Master Preface in MVS/Extended Architecture System Logic Library: Master Table of Contents and Index contains a list of book titles and mnemonics. It provides a quick reference to all the books, and their corresponding components, in the System Logic Library.

FINDING INFORMATION USING THE MASTER INDEX

If you are not sure which book contains the information you are looking for, you can locate the book and the page on which the information appears by using the master index in System Logic Library: Master Table of Contents and Index. For the component books, the page number in an index entry consists of the mnemonic for the component and the page number; for System Logic Library: Module Descriptions, the page number consists of the mnemonic "MOD" and the page number.

For example:

ASM-12 refers to MVS/Extended Architecture System Logic Library: Auxiliary Storage Management, page ASM-12.

MOD-245 refers to MVS/Extended Architecture System Logic Library: Module Descriptions, page MOD-245.

INFORMATION PROVIDED FOR MOST COMPONENTS

The following information is provided for most of the components described in the System Logic Library.

1. An introduction that summarizes the component's function
2. Control block overview figures that show significant fields and the chaining structure of the component's control blocks
3. Process flow figures that show control flow between the component's object modules
4. Module information that describes the functional organization of a program. This information can be in the form of:
   - Method-of-Operation diagrams and extended descriptions.
   - Automatically-generated prose. The automated module information is generated from the module prologue and the code itself. It consists of three parts: module description, module operation summary, and diagnostic aids.

5. Module descriptions that describe the operation of the modules (the module descriptions are contained in System Logic Library: Module Descriptions)

Some component books also include diagnostic techniques information following the Introduction.

FURTHER INFORMATION

For more information about the System Logic Library, including the order numbers of the books in the System Logic Library, see the Master Preface in MVS/Extended Architecture System Logic Library: Master Table of Contents and Index.

CONTENTS

FIGURES

Summary of Amendments
for LY28-1735-0
for MVS/System Product Version 2 Release 2.0

This publication is new for MVS System Product Version 2 Release 2.0. It contains information that was reorganized from the Recovery Termination Management (RTM) section in MVS/XA System Logic Library Volume 11, LY28-1246-2, which applies to MVS/XA System Product Version 2 Release 1.7.

This publication contains changes to support MVS/System Product Version 2 Release 2.0. The changes include:

- Method of Operation diagrams for the following new modules:

  IEAVTREF
  IEAVTRRR
  IEAVTR1F
  IEAVTR1G
  IEAVTR1I
  IEAVTR1N
  IEAVTR1R
  IEAVTR1S
  IEAVTR1X
  IEAVTR1O

- The following changed modules:

  IEAVTREM
  IEAVTRER
  IEAVTRET
  IEAVTRTD
  IEAVTRTS
  IEAVTRTV
  IEAVTR1A
  IEAVTR2A

- Module IEAVTSIN has been changed to module IEAVTR1N.

- Minor technical and editorial changes throughout the publication.

INTRODUCTION

Recovery termination management (RTM) cleans up systems resources when a task or address space terminates. Specifically, RTM performs normal and abnormal task termination, performs normal and abnormal address space termination, causes dumps to be written, records errors, and provides for recovery of supervisory routines by routing control to functional recovery routines. RTM provides these functions for both system and problem program routines.

Logically, RTM consists of four interrelated groups of functions that perform RTM services:

- RTM1: Attempts recovery after a request for an RTM service from supervisory routines. The CALLRTM macro instruction gives control to RTM1. RTM1 resides in the extended nucleus.
- RTM2: Performs normal and abnormal task termination for both system and problem program routines. The ABEND macro instruction (SVC 13) requests these RTM2 services. RTM2 resides in the extended link pack area (ELPA).
- Address space termination: Provides normal and abnormal address space termination for supervisory routines. The CALLRTM macro instruction is used to request this service. Address space termination resides in the extended nucleus and ELPA.
- RTM support functions: Provides error recording SLIP (serviceability level indication processing), and SPIE/ESPIE (specify program interruption exit/extended specify program interruption exit) processing.

ADDRESSING AND RESIDENCY OF RTM MODULES

All RTM modules execute in 31-bit addressing mode and reside above the 16mb line except:

IEAVNPA6	— AMODE 24, RMODE 24
IEAVNPD6	— AMODE 31, RMODE 24
IEAVTES6	— AMODE 24, RMODE 24
IEAVTRGR	— AMODE ANY, RMODE 24
IEAVTRGS	— AMODE ANY, RMODE 24
IEAVTRG1	— AMODE ANY, RMODE 24
IEAVTRG2	— AMODE 24, RMODE 24
IEAVTRML	— AMODE 24, RMODE 24
IEAVTSFR	— AMODE ANY, RMODE 24
IEAVTSLC	— AMODE ANY, RMODE 24

RTM1 FUNCTIONS

RTM1, which is part of the nucleus, consists of the following modules:

IEAVTRG1	Addressing mode interface routine
IEAVTRMC	RMGRCML preprocessor
IEAVTRRR	RTM1 FRR routines
IEAVTR1F	RTM1 FRR routing pre-processor
IEAVTR1G	RTM1 GTF processing module
IEAVTR1I	RTM1 general SDWA initialization
IEAVTR1N	RTM1 FRR stack initialization
IEAVTR1R	RTM1 record interface module
IEAVTR1S	RTM1 SDWA allocation module
IEAVTR1X	RTM1 CMSET interface module
IEAVTRS0	RTM1 mainline SLIH mode processing
IEAVTRTD	RTM1 subroutines

IEAVTRTF	RTM1 super FRR retry routine
IEAVTRTM	RTM1 mainline
IEAVTRTR	RTM1 recovery routines
IEAVTRTS	FRR control router
IEAVTR1C	Routing to FRRs
IEAVTRTV	RTM PSACSTK verification module
IEAVTRT1	RTM1 entry point and exit processor
IEAVTR1A	RTM1 failing instruction processor
IEAVTSR1	ITERM processor

RTM1 attempts recovery from hardware and software errors for routines protected by FRRs (functional recovery routines) defined by the routine to terminate those tasks or address spaces, via SVC 13, that cannot recover). To achieve recovery, RTM1 routes control to the FRRs when program checks, machine checks, STERM errors (paging I/O errors), invalid SVCs, or restarts occur.

RTM1 functions are divided into three logical categories:

- Second level interruption handler (SLIH) mode. RTM1 acts as second level interruption handler for the interrupt handlers when they detect errors. (See the section "Supervisor Control" for a description of the five interruption handlers).
- Service mode. RTM1 provides the interface for address space or task termination when entered in service mode.
- Hardware error mode. RTM1 functions as an extension of MCH (machine check handler) after a hardware-type error occurs.

SLIH MODE PROCESSING

RTM1, when in SLIH mode, schedules recovery for errors in system-mode functions, and initiates recovery for errors in task-mode processing. System mode recovery involves routing control to functional recovery routines (FRRs) and requesting error recording.

To implement recovery for system-mode functions, RTM1 routes control to the FRRs defined on FRR stacks for specific paths through the supervisor. (The M.O. diagram in "IEAVTR1C - Service Module for IEAVTRTS" on page RTM-301 fully defines the FRR stacks and the paths through the supervisor that they protect.) The system-mode functions use the SETFRR macro instruction (a macro instruction that places the address of the FRR on the stack) to make the FRR known to the system at initialization time. When an error occurs, RTM1 routes control to the FRRs, thus allowing a recovery path through system-mode functions.

SERVICE MODE PROCESSING

RTM1, when in service-mode processing, directs RTM's recovery and/or termination processing to a specific event, program, task, or address space other than the currently executing path. (Service requests often consist of scheduling entries into other services of RTM to complete the request.) Address space termination, requested via a CALLRTM TYPE=MEMTERM macro instruction, activates the resident address space termination controller and queues the ASCB of the address space to be terminated on a termination queue.

For task termination, requested by a CALLRTM TYPE=ABTERM macro instruction, RTM1 establishes an interface to RTM2. This interface differs for tasks in the current, or executing, address space, or for tasks in another address space. For ABTERM of a task in the current address space, RTM1 sets the request block (RB) resume PSW to point to the address of an SVC 13 instruction, which will be executed first when the task is redispatched. For ABTERM of a task in another address space,

RTM1 first reschedules itself as an SRB (service request block) in the address space executing the task to be terminated. Thus it appears that the CALLRTM TYPE=ABTERM request was issued by a task in the same address space. RTM1 uses this interface to give control to RTM2 as an RB issuing an SVC 13 instruction. RTM2 performs the actual recovery termination processing.

CALLRTM TYPE=ABTERM can also cause reentry into RTM1 if an EUT FRR is on the stack.

The STERM service (for page I/O errors) differs for unlocked tasks or for locked tasks and SRBs. For unlocked tasks, RTM1 sets an RB to point to an SVC 13 instruction, thereby giving control to RTM2 to execute a task termination. For locked tasks or SRBS, RTM1 establishes an interface to allow FRRS to gain control. RTM1 does this by causing the task or SRB to invalidly issue an SVC. This effects a re-entry into RTM1 in SLIH mode: RTM1 then routes control to FRRs defined for the path that failed. Figure 3 on page RTM-23 illustrates STERM processing, and refers to method of operation diagrams that describe the processing.

HARDWARE ERROR MODE

RTM1, when in hardware error mode, logically operates as a subroutine of the machine check handler (MCH). RTM1 performs software repair, gathers data about the error, and records the error. When MCH cannot recover from the error, RTM1 sets up an MCH re-entry to attempt software repair. Figure 4 on page RTM-25 illustrates how RTM1 handles a hardware error.

RTM2 FUNCTIONS

RTM2, which resides in the extended link pack area (ELPA), is entered via SVC 13. Mainline processing for RTM2 comprises the following modules:

IEAVTRTC — controller
IEAVTRTE — exit handler
IEAVTRT2 — initialization

Other important RTM2 modules are:

IEAVTAS1 — pre-exit processing
IEAVTAS2 — post-exit processing
IEAVTAS3 — control recovery
IEAVTMMT — address space purge
IEAVTMRM — RTM's address space termination resource manager
IEAVTRML — installation resource manager list
IEAVTR2A — RTM2 failing instruction processor
IEAVTSKT — task termination purges

RTM2 terminates tasks and controls the cleanup of their associated resources and control blocks. RTM2 handles normal task termination and termination of tasks that cannot complete their processing because of an error. Resource managers, routines called by RTM2, clean up the resources and control blocks associated with a task or address space to complete termination. The component owning the resource provides the resource manager.

RTM2 performs abnormal termination, which can be requested directly or indirectly. The request is direct when a system or user program issues an ABEND macro instruction to terminate the current task. The request is indirect when scheduled by RTM1. The SVC 13 instruction, which is executed the next time the task to be terminated is dispatched, causes supervisor-assisted linkage to ABEND.

NORMAL TERMINATION

When the last program to be executed for a task ends, it returns control to the EXIT routine. EXIT gives control to RTM2 to perform normal end-of-task processing. Figure 5 on page RTM-27 shows the steps that occur for normal task termination. (The Task Management section describes EXIT and EXIT prolog processing in detail.)

ABNORMAL TERMINATION

Abnormal termination occurs because of an unrecoverable error, such as an I/O error or program check. It can also be initiated by a system or user program that detects an abnormal condition that could cause program damage or incorrect results. The task whose program or I/O operation has malfunctioned is abnormally terminated because continued execution would waste system resources. Abnormal termination frees the resources for use by other tasks.

Abnormal termination allows two options: task and step termination. These are normally user options, specified by an operand of the ABEND macro instruction.

For abnormal termination, RTM2 provides the following services:

- Retry of a terminating task, if possible.
- Allow tasks that cannot retry to process special exits.
- Display a snapshot of storage.
- Wait for subtask termination to complete.
- Purge subtask resources.
- Convert ABEND requests to the jobstep level.

Figure 6 on page RTM-29 shows how RTM2 handles an abnormal termination.

Retry Terminating Tasks

RTM2 permits tasks scheduled for termination to bypass termination and resume processing if they have created exits for this function.

These exits receive control from RTM2 prior to termination completing. (This facility complements the FRR facility in RTM1.) The exits might attempt to recover the task being terminated; if successful, RTM2 does not terminate the task. If the exit does not recover the task, task termination continues. Figure 7 on page RTM-31 shows retry.

Term Exits

Whereas RTM2 allows retry during most task terminations, certain conditions (for example, CANCEL requests, ancestor task abnormally terminating, and timer expiration) cannot be retried. However, a special feature of ESTAE/ESTAI exits, called the TERM option, can be used to give control to an ESTAE or ESTAI exit during these situations. (The user indicates this by specifying TERM=YES when the ESTAE or ESTAI is issued.) During normal error recovery processing for a task, these exits function in exactly the same way as exits created without the TERM option. But for a situation that cannot be retried, these specially marked exits are given control so that a user can clean up resources, write records, print messages, or perform any other function before RTM2 completes the termination. Retry, even though requested, is not permitted by RTM2. Figure 8 on page RTM-33 shows how RTM2 processes a CANCEL request and routes control to term exits.

It is now possible to issue the DETACH macro from within a term exit.

Storage Dump

When the DUMP option is specified on the ABEND, CALLRTM, or SETRP macro, RTM2 will create an ABEND dump, via SNAP, for all tasks in the failing task tree.

Control Block Formatter

IEAVTFMT formats the available RTM control blocks associated with the TCB. Print dump (PRDMP), interactive problem control system (IPCS), and SNAP call the RTM control block formatter as a TCB exit in the following manner:

- PRDMP specifies the SUMMARY control statement with the FORMAT parameter.
- IPCS specifies the SUMMARY subcommand with the FORMAT keyword.
- SNAP specifies the SUM or ERR option.

Wait for Subtask Termination

RTM2 waits for subtasks within RTM2 processing to complete before terminating all the other subtasks in the task tree. RTM2 can stack, or wait, for up to four subtasks to be processed at one time. (This does not apply to CANCEL requests.)

Purge Subtasks

To terminate the tasks in a failing task tree, RTM2 removes, via DETACH, each subtask. DETACH then abnormally terminates, via CALLRTM TYPE=ABTERM, any that has not yet completed processing.

Convert to Step

When a caller requests ABEND (SVC 13) with the STEP option, RTM2 completely terminates the failing task and any of its subtasks. Then before giving control to EXIT prolog, RTM2 issues a CALLRTM TYPE=ABTERM request for the job step task.

ADDRESS SPACE TERMINATION

Address space termination can be requested by certain system functions. For example, real storage management might decide to terminate an address space because of a swap-in failure for the LSQA. Normally, however, RTM2 requests address space termination after task termination of the region control task.

Address space termination begins after RTM1 invokes the address space termination controller by scheduling the address space termination SRB to post it. The address space termination controller determines the address space being terminated and dequeues the ASCB. The address space termination controller then attaches the address space termination task to complete the termination. The termination will be complete after all the resources associated with the address space have been purged by the address space termination controller and RTM2. Figure 9 on page RTM-34 shows the control flow of an address space termination.

RTM SUPPORT FUNCTIONS

RTM provides functions that allow users to establish their own recovery protection, and system functions that enhance system serviceability and reliability. RTM gives control to these services as part of its main processing, but none of these are integral to RTM.

RTM support services consist of the following:

- STAE (specify task abnormal conditions) and ESTAE (extended STAE) services. STAE and ESTAE services create SCBs (STAE control blocks) to represent user-written abnormal condition exits. RTM will give control to these exits during termination processing. STAE services are not supported in 31-bit mode.
- SETFRR. This is a macro instruction that places an FRR (functional recovery routine) on the correct FRR stack. RTM routines route control to FRRs after an error occurs.
- Initializing FRR stacks. This service creates FRR stacks during system initialization, and changes FRR stacks in response to CONFIG processor commands.
- Recording. RTM uses recording to record errors and records created during recovery or termination processing.
- SLIP command. To obtain diagnostic information, SLIP intercepts software errors prior to recovery routines receiving control.

STAE SERVICES

The STAE services create SCBs that represent caller-requested abnormal exits. STAE services, requested via an SVC 60 instruction, create four types of SCBs:

- ESTAE SCBs.
- ESTAI SCBs.
- STAE SCBs.
- STAI SCBs.

SETFRR

The SETFRR macro instruction places an FRR on the appropriate FRR stack. This is the mechanism used by routines requiring recovery protection.

INITIALIZING FRR STACKS

During initialization, this function initializes the FRR stacks used by the system, and places pointers to these stacks in the recovery stack vector table (RSVT) of the PSA. The RSVT does not have sufficient room for the RTM and ACR stacks. Therefore, the addresses of these stacks are placed in other PSA fields. The CONFIG processor command can use this function. The FRR stacks initialized by this function are:

- SVC-I/O-dispatcher stack, used by supervisor control routines.
- Machine check stack, used by the machine check handler after a machine check occurs.
- Program check stack, used by the program check handler after a program check occurs.
- The three external interrupt handler stacks, used by the external interrupt handler to process three levels of recursion. (See the section "Supervisor Control" for a

description of the external interrupt handler and its use of the FRR stacks.)

- Restart interrupt handler stack, used by the restart interrupt handler.
- RTM stack, used by the RTM function when it uses FRR recovery.
- ACR stack, used by the ACR function during CPU recovery processing.
- Normal stack, used by supervisor control routines processing on behalf of problem programs that use supervisor services.

RECORDING SERVICES

The recording facility schedules asynchronous I/O either to SYS1.LOGREC or to the operator. The facility consists of two principal routines — the nucleus-resident recording request routine (IEAVTRER) and the recording task (IEAVTRET) in the master scheduler address space. Requests for recording by disabled routines are accepted and buffered by the nucleus routine, which in turn posts the recording task via an SRB. The recording tasks write the queued records to SYS1.LOGREC by issuing SVC 76 or to the operator by issuing SVC 35.

THE SLIP COMMAND

Serviceability level indication processing (SLIP) is a debugging facility used for obtaining diagnostic information. There are times when an SVC dump or ABEND dump does not give the user adequate information about an error. For example, the recovery process or independent system activity might alter the failing environment before the dump can be scheduled. To avoid this situation, SLIP can be used to selectively intercept software errors that are handled by RTM. However, many MVS/XA problems cannot be resolved using only data collected at the time of the error. Therefore, SLIP also provides program event recording (PER) support to allow the user to obtain diagnostic information only when a situation of interest occurs. Thus, a system operator or authorized TSO user can issue the SLIP command to establish one of two types of SLIP traps, non-PER and PER:

- Non-PER traps specify error conditions which the action specified on the trap is to be taken.
- By using the PER hardware, PER traps specify instruction fetch, storage alteration, or successful branch events that are to be monitored within a range of virtual addresses. When events of the selected type occur, system conditions specified on the trap are compared with current system conditions. If they match, the action specified on the trap is taken.

Possible actions include scheduling an SVC dump, placing the system in a wait state, suppressing dumps, writing a GTF trace record, or opting to take no action (the IGNORE option). After taking the specified action, PER traps may also specify that recovery processing be forced in the interrupted program. The SLIP command allows the user to establish more than one SLIP trap, and to selectively enable and disable the traps. However, only one PER trap with an action other than IGNORE can be enabled at a time. Controls may be specified with the trap that automatically disable it when user-specified conditions exist.

The SLIP command processor (IEECB905) sets, modifies, or deletes one or all of the SLIP control element (SCE) SLIP traps. It receives control via the ATTACH macro when IEEVWAIT processes the CSCB entry built by SVC 34 as a result of a SLIP command. (The section "Command Processing" contains more details on the SLIP command processor.)

The user can display detailed information or summary information about all the SLIP traps by using the DISPLAY SLIP command. The DISPLAY SLIP command processor (IEECB907) receives control via the ATTACH macro when IEEVWAIT processes the CSCB entry built by the DISPLAY command as a result of a DISPLAY SLIP request. The DISPLAY SLIP command processor also receives control via BALR from the SLIP command processor (IEECB905) to display the requested options and any defaults for an incomplete (that is, no END parameter) SLIP command. (The section "Command Processing" contains more details on the DISPLAY SLIP command.)

SPIE/ESPIE PROCESSING

Requestors can use the SPIE/ESPIE service to allow a task to regain control after certain program interruptions. The SPIE/ESPIE service routine receives control from the SVC FLIH after a SPIE or ESPIE request occurs. SPIE/ESPIE constructs an SCA (SPIE control area) that contains information that enables a task to regain control after a program interruption. (See the section "Supervisor Control" for a description of the interruption types.) SPIE/ESPIE constructs the SCA and sets indicators in the TCB of the requestor.

RTM work areas can be valuable in identifying failing components when the system malfunctions.

RTM work areas and suggestions for using them to diagnose failures appear in the Diagnostic Techniques publication under the topic "Use of Recovery Work Areas for Problem Analysis."

The following section contains diagnostic information for the SLIP portion of RTM.

SLIP PROCESSOR DEBUGGING AIDS

A considerable amount of recovery processing is built into the SLIP function, some portion of that recovery is executed if an error occurs during SLIP processing. Consequently, when trying to debug the SLIP function, you should have an idea of what the recovery processing is attempting to do. This section discusses the recovery philosophy and provides details for the major SLIP functions.

SLIP COMMAND PROCESSOR RECOVERY

Module IEECB906 provides recovery processing for the SLIP command processor (primarily IEECB905). Most errors encountered in the command processor affect only the command that is issued and not the rest of the system. However, if a PER trap is involved, an error in the command processor could potentially affect the system.

If a PER trap is being disabled or deleted and an error is encountered, IEECB906 disables the non-IGNORE PER trap and schedules IEAVTGLB to deactivate PER. If a PER trap is being set or enabled and error occurs after SHDRPER has been updated but before IEAVTGLB has been scheduled, IEECB906 tries to schedule IEAVTGLB to activate PER. Additionally, whenever an error occurs, the command processor recovery routine checks to make sure the double-threaded SCE chain is properly chained. Forward and backward pointers found to be in error are repaired if possible. If an error occurs during recovery for the SLIP command processor, SLIP recovery does not return to mainline processing but requests percolation in the event of an error.

Diagnostic information concerning errors that occur in the command processor is available in a software LOGREC record and a dump. The ESTAE parameter list (mapped by IEEZB906) is part of the LOGREC record. The ESTAE parameter list and the SHDR data area along with other information are available in a dump for the error.

SLIP PROCESSOR RECOVERY

Recovery for the SLIP processor is designed to handle both expected and unexpected errors.

Errors which are considered "expected" are:

- A page fault occurs while examining or retrieving the instruction that caused a PER interrupt.
- A page fault occurs while retrieving user-defined data.
- A page fault occurs while processing in IEAVTADR.

When the above error conditions are recognized, the SLIP processor attempts to retry at an appropriate point. In

general, the retry allows normal trap processing to continue. You may eventually receive an indication that an error has occurred while examining a trap (for example, the data unavailable counter has been increased.) SYS1.LOGREC recording does not occur for these expected errors.

When an unexpected error occurs, SLIP processor recovery gathers information concerning the error, cleans up any resources being used by the SLIP processor, and then retries at a point that will terminate processing for the event that caused the SLIP processor to receive control. Diagnostic information concerning the error can be found in the dump taken by the SLIP processor recovery routine (IEAVTSLR). The summary dump usually contains:

- The FRR parameter list (mapped by IHASLFP in module IEAVTSLP or IEAVTSLR).

  Note: The FRR parameter list is also recorded as part of the software LOGREC record for the error. Bits in the AUDITWRD portion of the FRR parameter list provide an indication of what portion of the SLIP processor encountered the error.

- The SHDR data area.
- The SCE/SCVA data areas being processed at the time of the error.
- The SLIP parameter list (IHASLPL).
- SLIP work areas.
- The SLIP register save area.
- The SCE/SCVA data areas representing the enabled non-IGNORE PER trap (if they exist).

Further information concerning the error is included in the software LOGREC record for the error.

PER ACTIVATION/DEACTIVATION RECOVERY

The PER activation/deactivation function is performed primarily by SLIP modules IEAVTGLB, IEAVTSIG, IEAVTLCL, and IEAVTJBN. In general, if an error is encountered at any point in the PER activation/deactivation process, these modules try to deactivate PER completely. Recovery processing for these modules is described in the following topics.

IEAVTGLB Recovery

If an error is encountered by IEAVTGLB, the recovery for this module gathers information concerning the error, frees the resources held by the mainline code, disables the non-IGNORE PER trap, and then retries at a point in the module that attempts to completely deactivate PER. Diagnostic information concerning the error is recorded in a software LOGREC record and a dump. The information available in the summary dump includes some or all of the following (depending on when the error occurred).

- The FRR parameter list (mapped by FRRWA in module IEAVTGLB).

  Note: The FRR parameter list is also recorded as part of the software LOGREC record for the error.

- The CVT data area.
- The SHDR data area.
- The SCE/SCVA data areas for the non-IGNORE PER trap.
- The model PSA data area.

- The PCCAVT data area.
- The ASCB being processed by IEAVTGLB.
- The name of the job running in the address space being processed by IEAVTGLB.
- The PCCA data area.
- The PER control registers (9, 10 and 11).

If a recursive error is encountered by IEAVTGLB, message IEA414I is sent to the operator and percolation is requested.

IEAVTLCL Recovery

If an error is encountered by IEAVTLCL, the recovery for this module sets tasks dispatchable in the address space, gathers information concerning the error, frees the resources held by the mainline code, and then percolates the error. Diagnostic information concerning the error is available in a software LOGREC record and a dump. The information in the summary dump includes some or all of the following (depending on when the error occurred).

- The FRR parameter list (mapped by FRRPARMS in module IEAVTLCL).
- The CVT data area.
- The SHDR data area.
- The SCE/SCVA data areas for the non-IGNORE PER trap.
- The ASCB for the address space in which IEAVTLCL was running when the error occurred.
- The name of the job in the address space.

IEAVTJBN Recovery

If an error is encountered by IEAVTJBN, the recovery for this module gathers information concerning the error, notifies the SLIP user that the status of PER in the system is uncertain (via message IEA422I), and then returns to mainline processing where control is returned to the caller of IEAVTJBN. Diagnostic information concerning the error is available in a software LOGREC record and a dump.

Control Blocks Used by SLIP

The following control blocks contain key information that can be used to debug problems in SLIP routines.

- System Control Blocks
  - Address space control block (ASCB)
  - Logical configuration communication area (LCCA)
  - Prefixed save area (PSA)
  - Request block (RB)
  - Task control block (TCB)
- SLIP Control Blocks
  - SLIP control element (SCE)
  - SLIP control element variable area (SCVA)
  - SLIP header (SHDR)
  - SLIP TSO element (STE)

Control Block	Information for Debugging SLIP
ASCB	ASCBPER bit: 1—PER is active in the address space. 0—PER is inactive.
LCCA	LCCAPPSW field: PSW LCCAPERC field: Interruption code LCCASLIP field: Pointer to SLIP storage area IEAVTPER splits this area into: a parameter list, work area, and register save area before calling SLIP.
PSA	External, SVC, I/O new PSW's: each has a bit that reflects PER status. PSASLIP bit: SLIP recursion control.
RB	RBOPSW field: PSW save area. PSW has a bit that reflects PER status.
TCB	Non-dispatchability bit for SLIP: used when PER is being activated or deactivated.
SCE	SCEDSABL bit: 1—SLIP is disabled. 0—SLIP trap is enabled. SCEMATCH bit: 1—Trap method specified conditions at least once since it was enabled.
SVCA	SCVAMLNO field: If MATCHLIM was specified or defaulted, indicates the number of times the trap matched specified conditions since it was enabled. SCVADAUN field: If data was specified, indicates the number of times data was unavailable for comparison for the trap.
SHDR	The SHDR provides the anchor for the chain of SCE/SCVA control blocks. It is pointed to by CVTRMS. The SHDRFWD field points to the first SCE on the chain and the SHDRBKWD field points to the last SCE. SHDRPFC field: 0—No enabled SLIP traps. 1—SLIP and associated routines are page fixed ; no processing is taking place on behalf of any trap. 2 or more—SLIP or portions of IEVTGLB are running.

Control Block	Information for Debugging SLIP
SHDR	SHDRSRBR bit: 1—IEVTGLB needs to be scheduled. This bit is usually turned on when IEAVGLB tries to get a resource (primarily SHDRSEQ) to perfrom some service but the resource is not available. When on, it indicates that IEABTGLB will be scheduled to perform the service later. The SLIP command processor (IEEBC905) may also set this bit and examine this bit when the sequence word is released. SHDRPER field: Points to enabled non-IGNORE PER trap or is zero. SHDRSEQ word: Used as a lock to serialize access to the SCE chain for: • The SLIP command processor (IEECB905). • PER activation/deactivation routine (IEAVTGLB). • Local PER activation/deactivation (IEECB907). The contents of the word indicate the owner of the word as follows: • 'CMD'— IEECB905 • 'DSP'— IEECB907 • 'GLB'— IEAVTGLB • 'Lxx'— IEAVTCLC (where xx indicates the ASID in which IEAVTLCL is running.)
STE	The STE is used to communicate between the SLIP command processor running in the master scheduler address space and a TSO user who issued the SLIP command. The STE is created when the TSO user issues the SLIP command and is deleted when SLIP command processing is completed. The STE chain is pointed to by the RTCTSTE field in the RTCT.

CONTROL BLOCK OVERVIEW

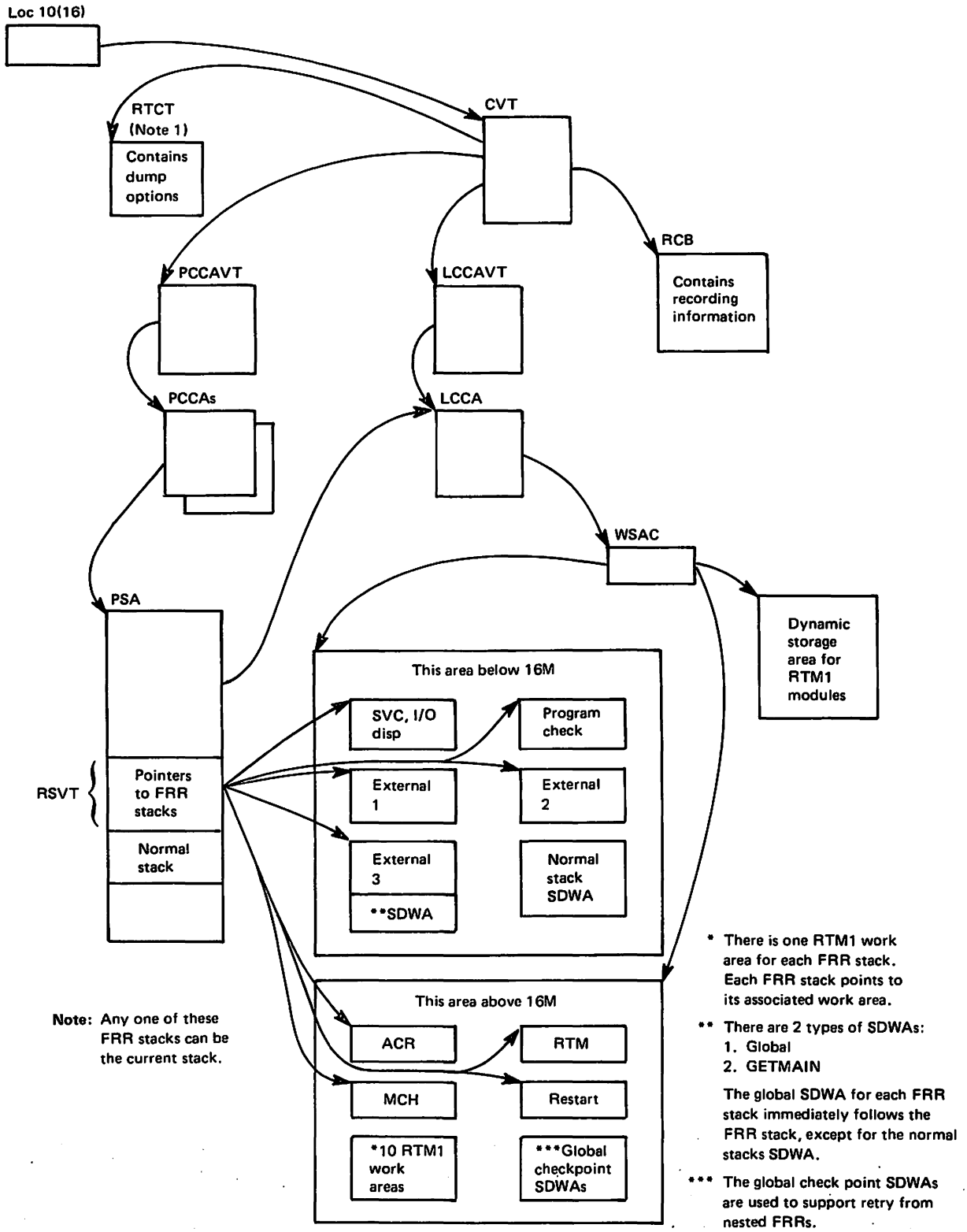
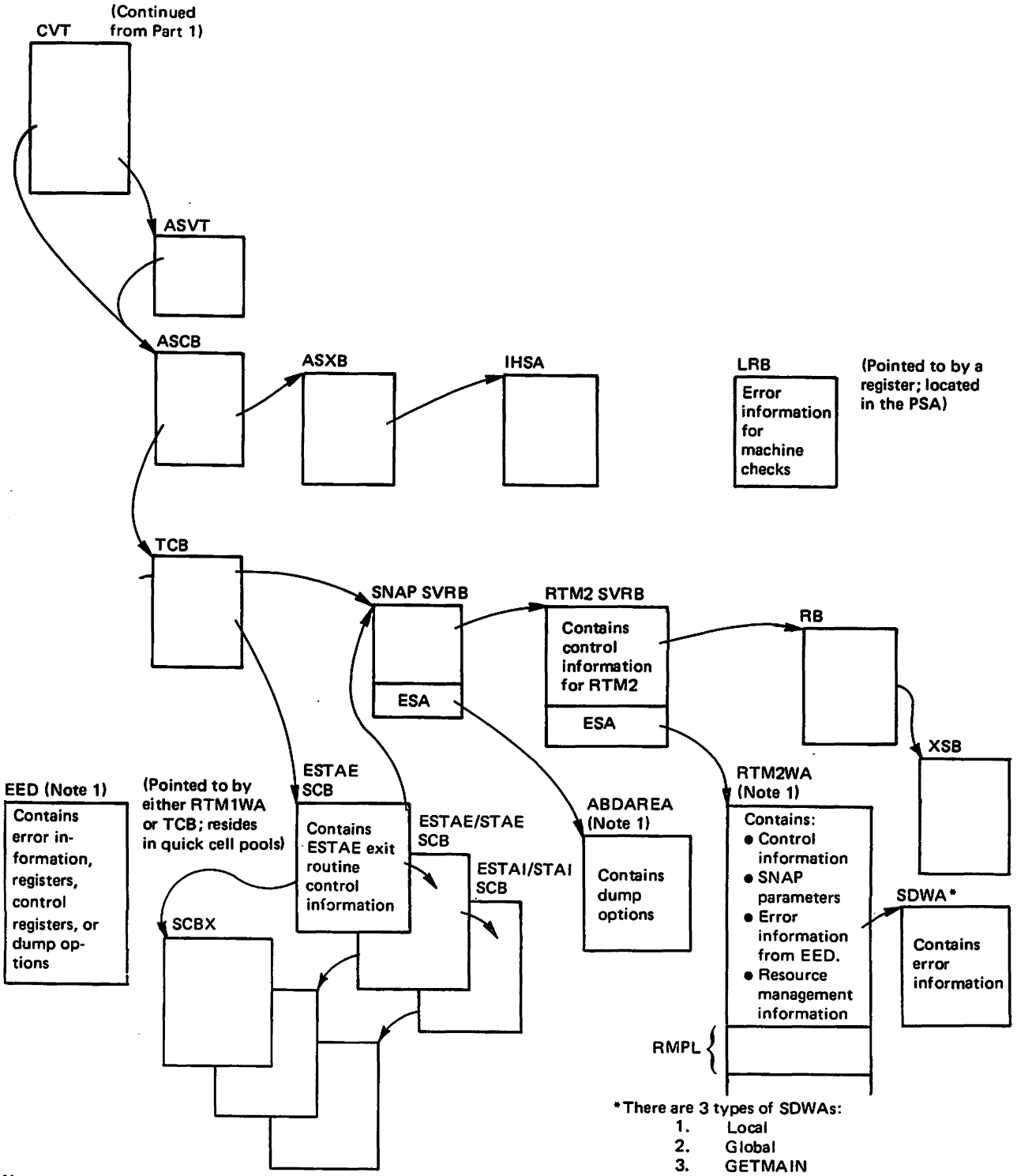


Figure 1 (Part 1 of 2). Recovery Termination Management Control Block Overview



Notes:

1. The RTCT, EED, and ABDAREA contain information that is moved into the RTM2WA. Information in the RTM2WA is moved into the SDWA.

Figure 1 (Part 2 of 2). Recovery Termination Management Control Block Overview

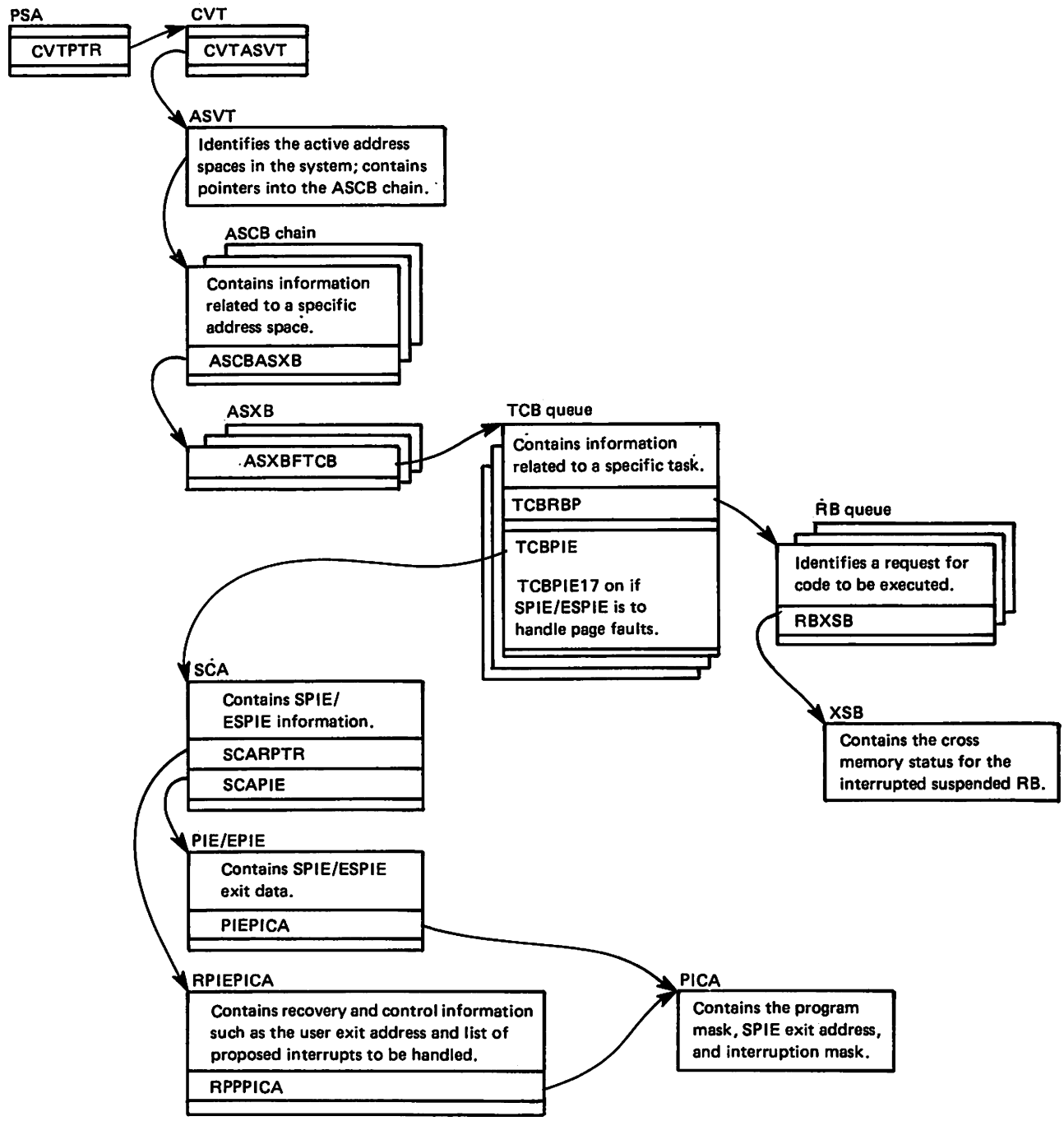


Figure 2. SPIE/ESPIE Control Block Overview

PROCESS FLOW

This section contains the following figures:

Figure 3 on page RTM-23: STERM Error Processing

This figure shows the scope of supervisor control, IOS, RSM and RTM involvement in processing an STERM error. The double error (SVCERR) caused by RTM1 shows how an RTM1 service request (STERM) establishes the proper RTM1 re-entry interface so that recovery routines can be processed.

The figure shows how an I/O error during a page-in request is processed by RTM. For this example an SRB routine has been used. However, similar action is given for locally locked tasks and normal tasks.

Data flow is as follows. The operating environment of program A — that is, the registers, control registers 3 and 4, PSW and recovery stack (step 1) is stored into an SSRB on page interrupts (step 2). When the error is detected by the paging supervisor (step 4) the SSRB is passed to RTM. RTM copies the registers, control registers 3 and 4, and the PSW from the SSRB into its own data area — the EED (step 5) and alters the SSRB fields so that it will issue an ABEND when redispatched. The page reset routine puts the SSRB on the dispatching queue (step 4). The dispatcher dequeues the SSRB (step 6), copies the stack contents (saved in step 1) into the normal stack (re-establishing program A's recovery) and loads the registers and PSW from the SSRB (modified by RTM in step 5 to cause an ABEND). As a result of the ABEND, RTM is re-entered (step 8) and passes the original register, control registers 3 and 4, and PSW from the EED into an SDWA (step 10) so that the FRR for program A is presented with the environmental information at the time it was first interrupted for a page fault.

Figure 4 on page RTM-25: Hardware Error Processing

This figure depicts 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 alludes to the fact that software recovery will continue in task mode, because in this example the FRR does not recover the error.

The use of EEDs allows the LOGREC buffer to be available for further possible machine checks and is the mechanism of passing information to RTM1 and RTM2. The information in the global SDWA used by RTM1 recovery was obtained from the EEDs. RTM2 will obtain an SDWA but will also use EEDs as its source of error data to be passed to the recovery routines.

The RTM processor related work save area (WSACRTMK) is used by RTM1 to alter the general purpose registers and the PSW that MCH will reload — thereby determining whether MCH will resume the interrupted process (soft error), or re-enter RTM1 for software recovery (hard error).

Figure 5 on page RTM-27: The Process of Normal Task Termination

EXIT and parts of RTM2 comprise this function. The figure indicates how EXIT is entered and re-entered to complete task termination. It also provides a perspective of RTM2 functions related to normal termination of a task.

Figure 6 on page RTM-29: Abnormal End-of-Task

This figure shows 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.

Figure 7 on page RTM-31: Retry

This figure shows the flow through RTM2 when processing a potentially recoverable error. The recovery exit is supplied environmental data that describes the error (for example, the completion code, register contents, PSW, and system state at the time of the error), to aid in diagnosing the error. To effect retry, the resume PSW in each 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. All the RBs between the retry RB and the RTM2 RB have their resume PSW set to either EXIT prolog or SVC 3. To ensure running in the home address space, the RBOPSW S-bit is set to 0, and the primary and secondary address spaces in the XSB are set to the home address space. When RTM2 eventually returns to the system, supervisor assisted linkage will cause the retry address in the retry RB to be given control.

Figure 8 on page RTM-33: Cancel

This figure illustrates 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=X'x22' where x is any value). 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, termination exits are given control.

Figure 9 on page RTM-34: The Process of Terminating an Address Space

The process of terminating an address space (memory) is one which cannot be isolated to one task, module or logical unit of code. This figure shows the control flow and data flow of this process. The multiple dispatches, tasks, and address spaces involved would otherwise be hidden elements.

Figure 10 on page RTM-36: SRB to Task Percolation

This figure shows the flow of control through RTM when rescheduling an SRB. Error information is saved in EEDs or SPIs before the SRB is rescheduled. An SVC 13 (ABEND) placed in the RBOPSW identifies this SRB as a re-entry function to RTM1.

Figure 11 on page RTM-38: Removal of a SPI

This figure shows the process of removing a SPI (serial percolation information) control block from the SPI queue. Each SPI queued from a TCB represents a percolation from an SRB's recovery. At the time of the percolation, the related task was in recovery and the last FRR to get control for the SRB requested serialization. (See Figure 10 on page RTM-36).

Figures 12 and 13 show the flow and basic functions of RTM1 and RTM2.

Figures 14, 15, 16, and 17 show the module flow for address space termination, RTM service routines, SLIP action processing, and SPIE/ESPIE.

Figure 18 shows the module flow for formatting RTM's control blocks.

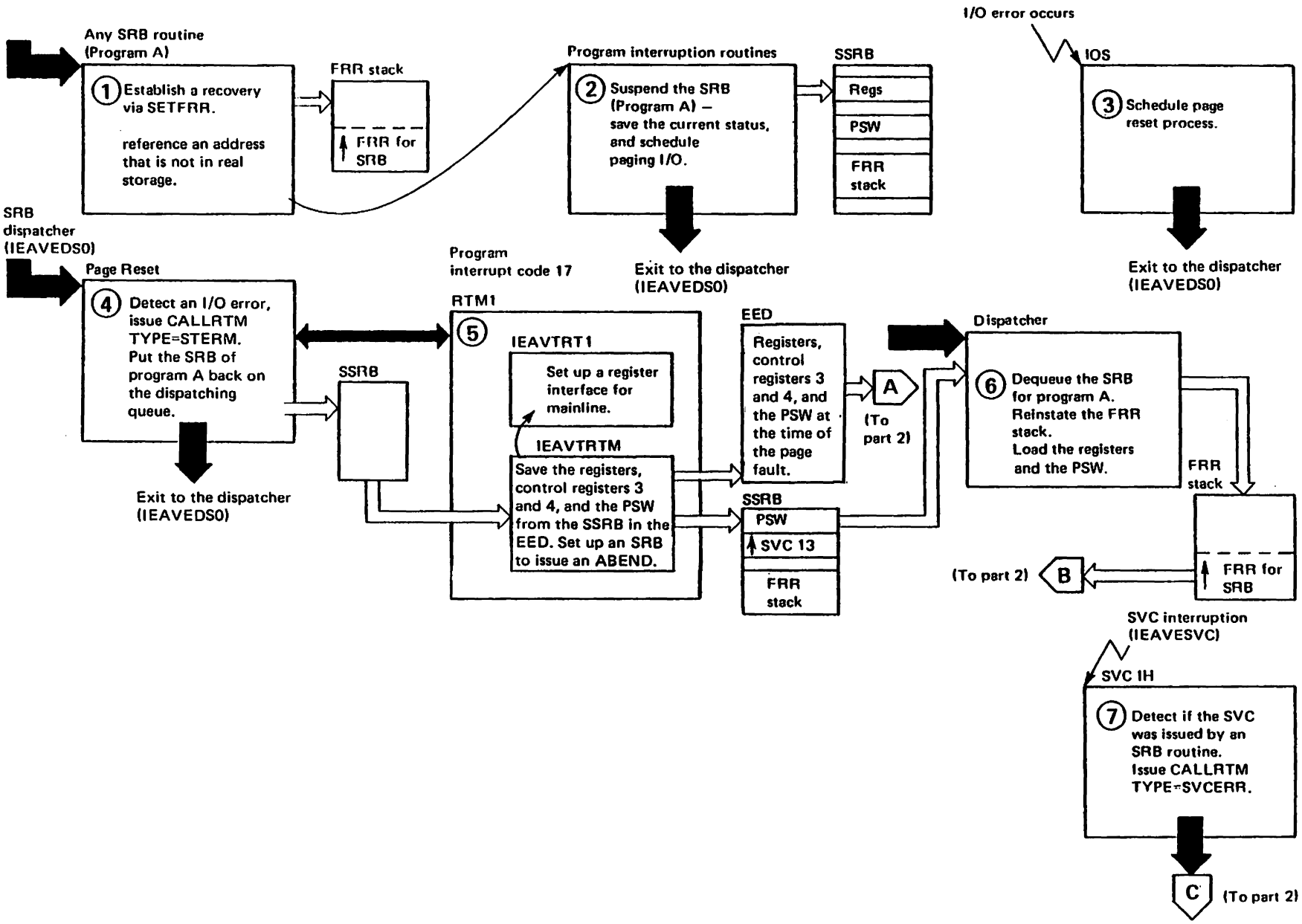


Figure 3 (Part 1 of 2). STERM Error Processing

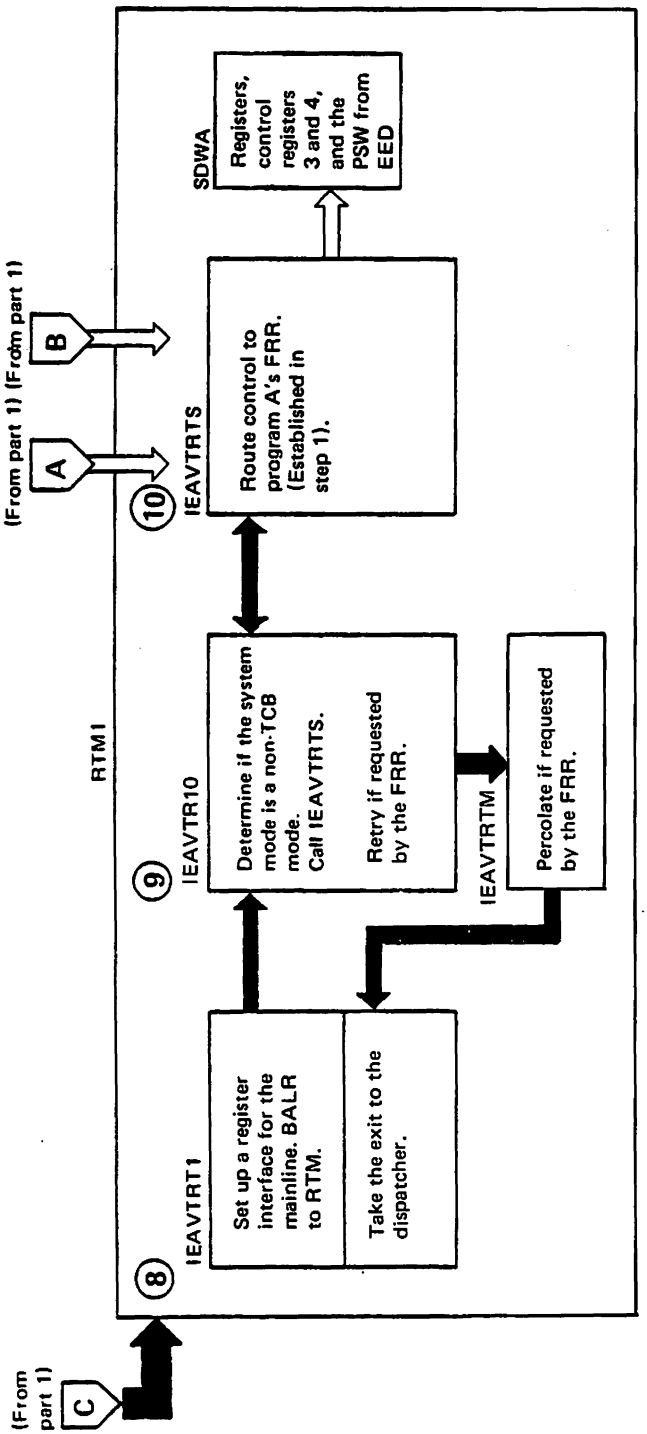


Figure 3 (Part 2 of 2). STERM Error Processing

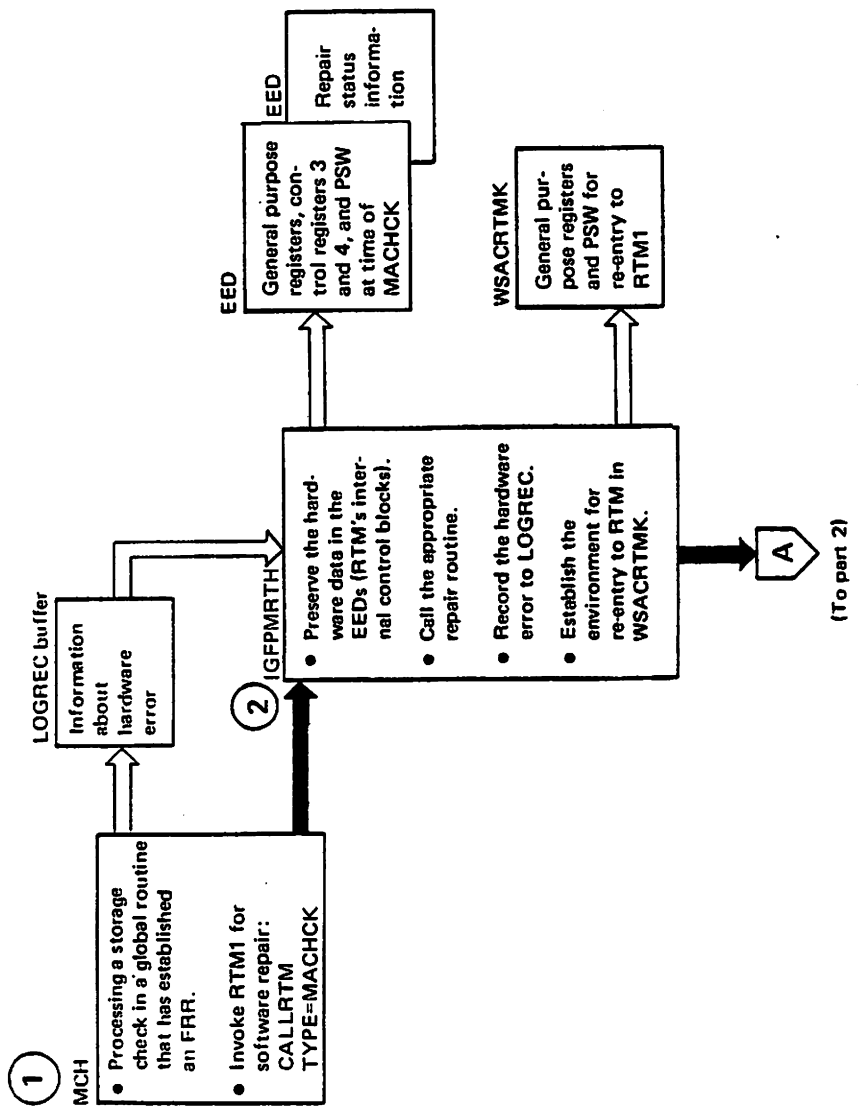


Figure 4 (Part 1 of 2). Hardware Error Processing

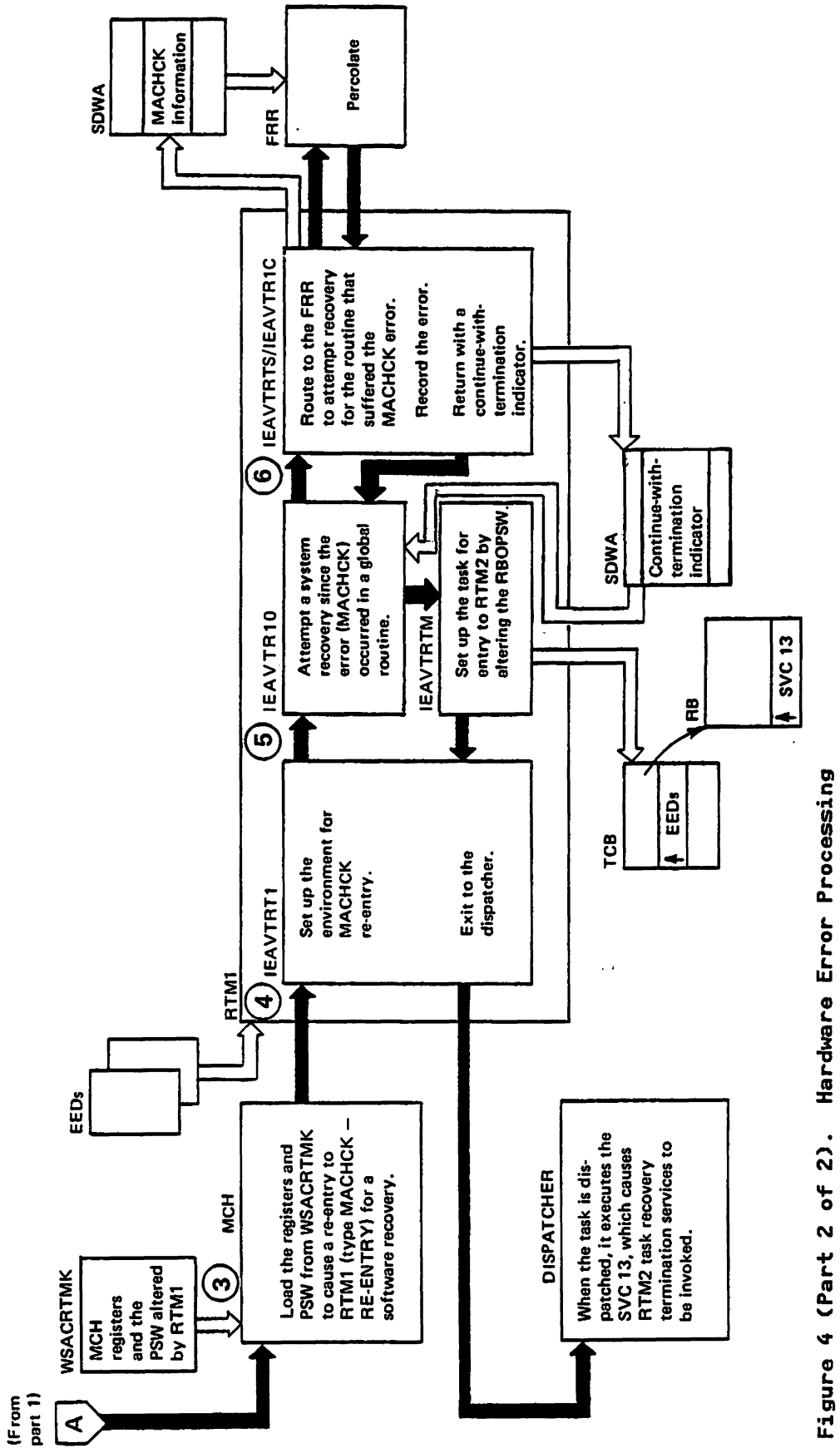


Figure 4 (Part 2 of 2). Hardware Error Processing

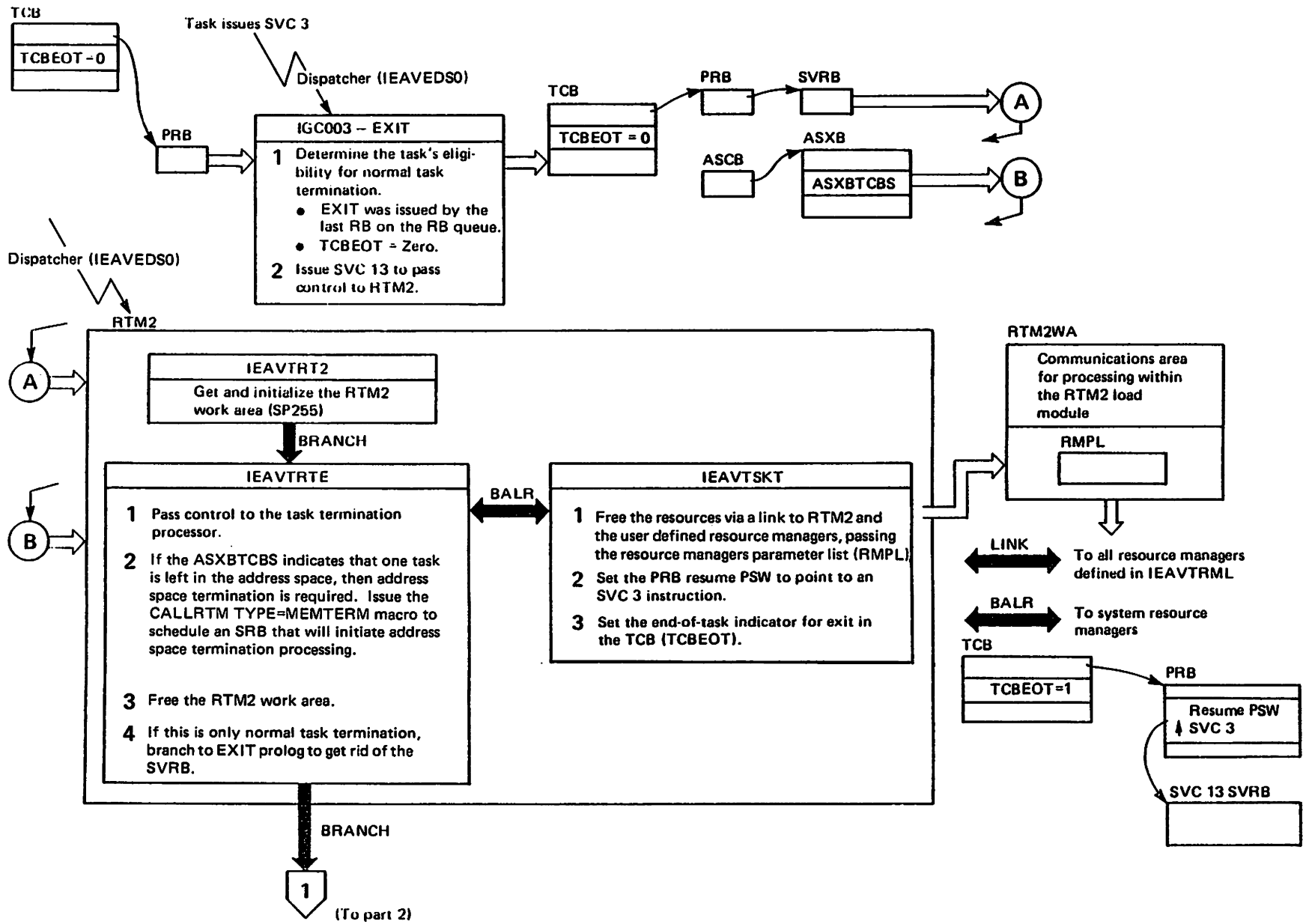


Figure 5 (Part 1 of 2). The Process of Normal Task Termination

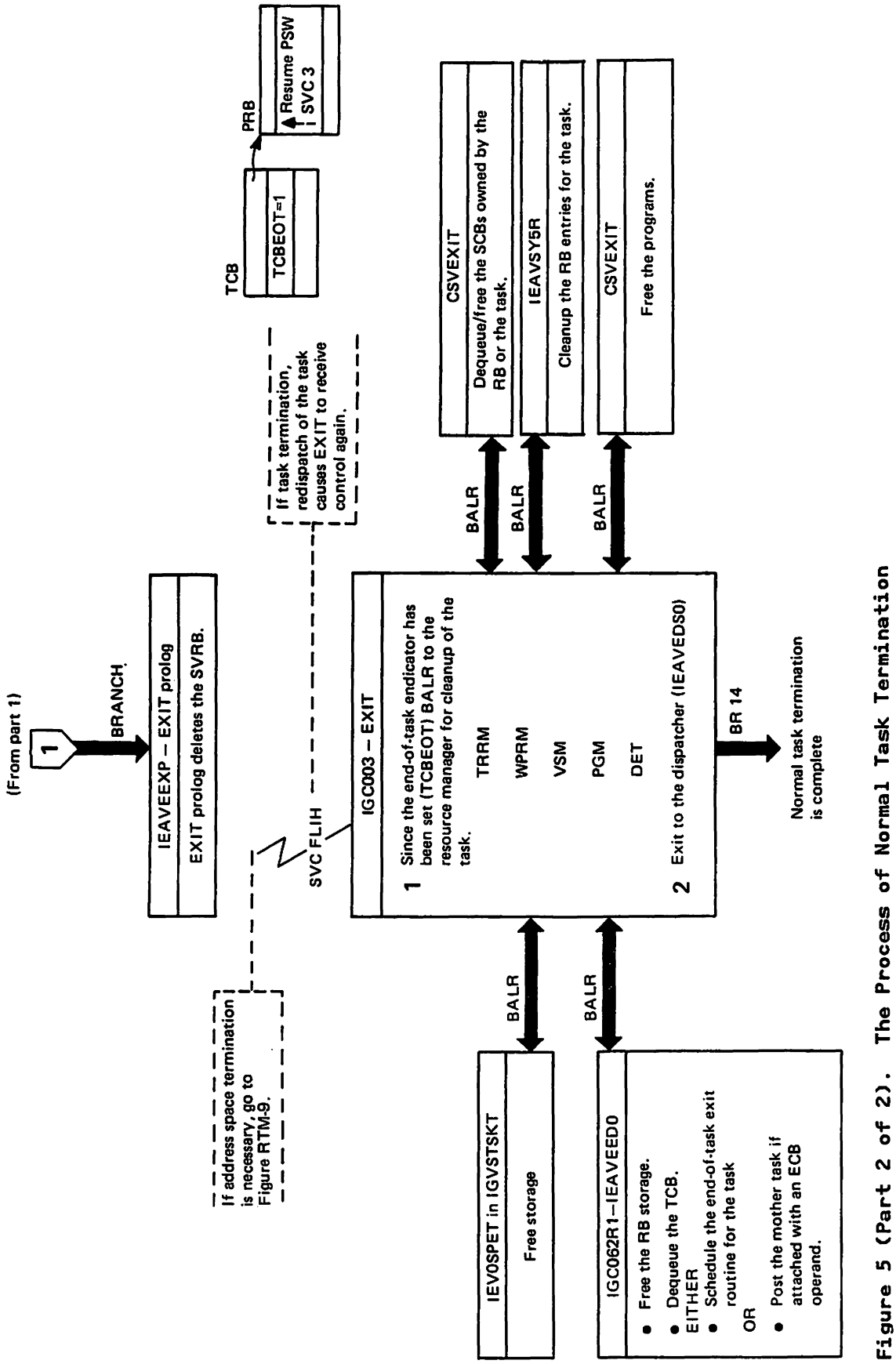


Figure 5 (Part 2 of 2). The Process of Normal Task Termination

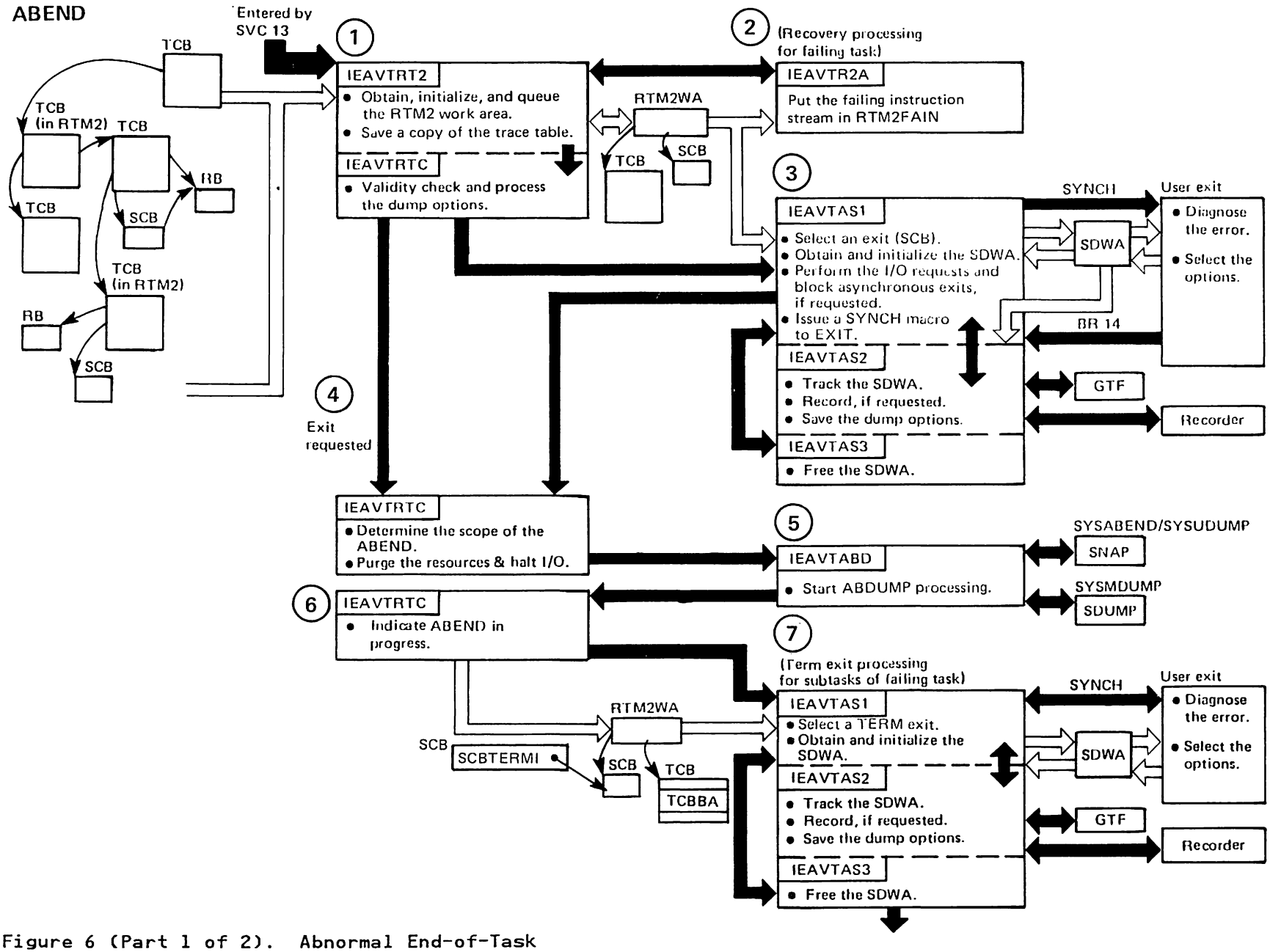


Figure 6 (Part 1 of 2). Abnormal End-of-Task

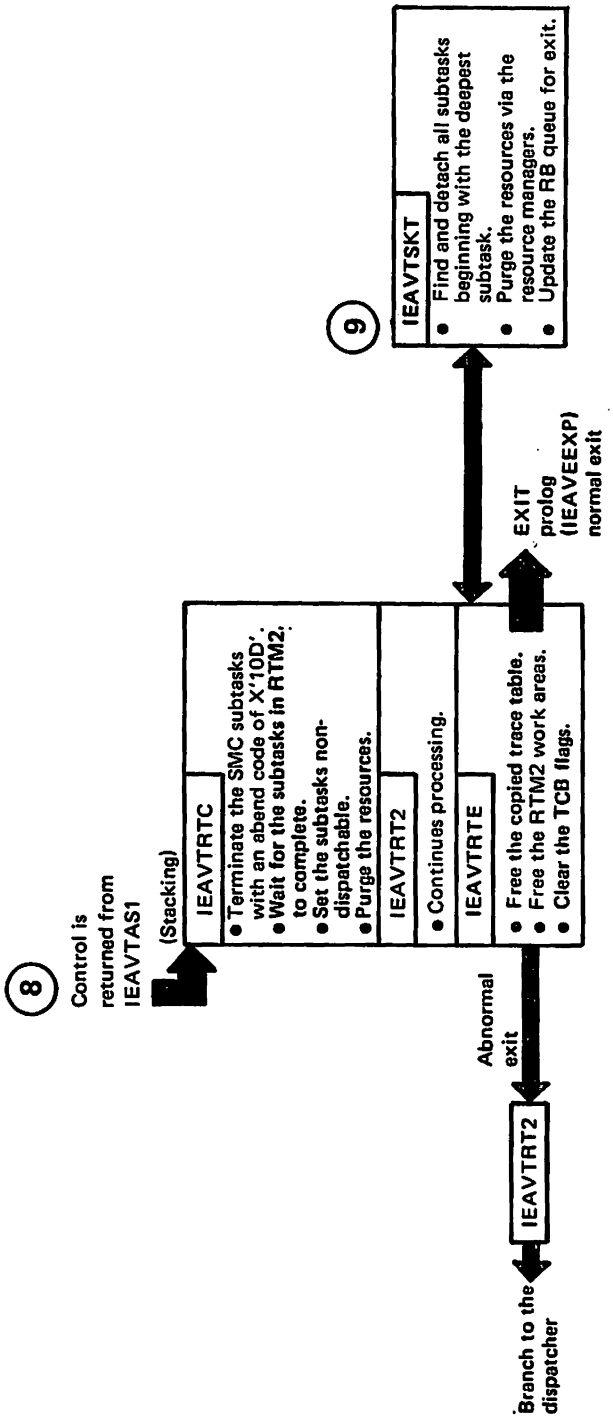


Figure 6 (Part 2 of 2). Abnormal End-of-Task

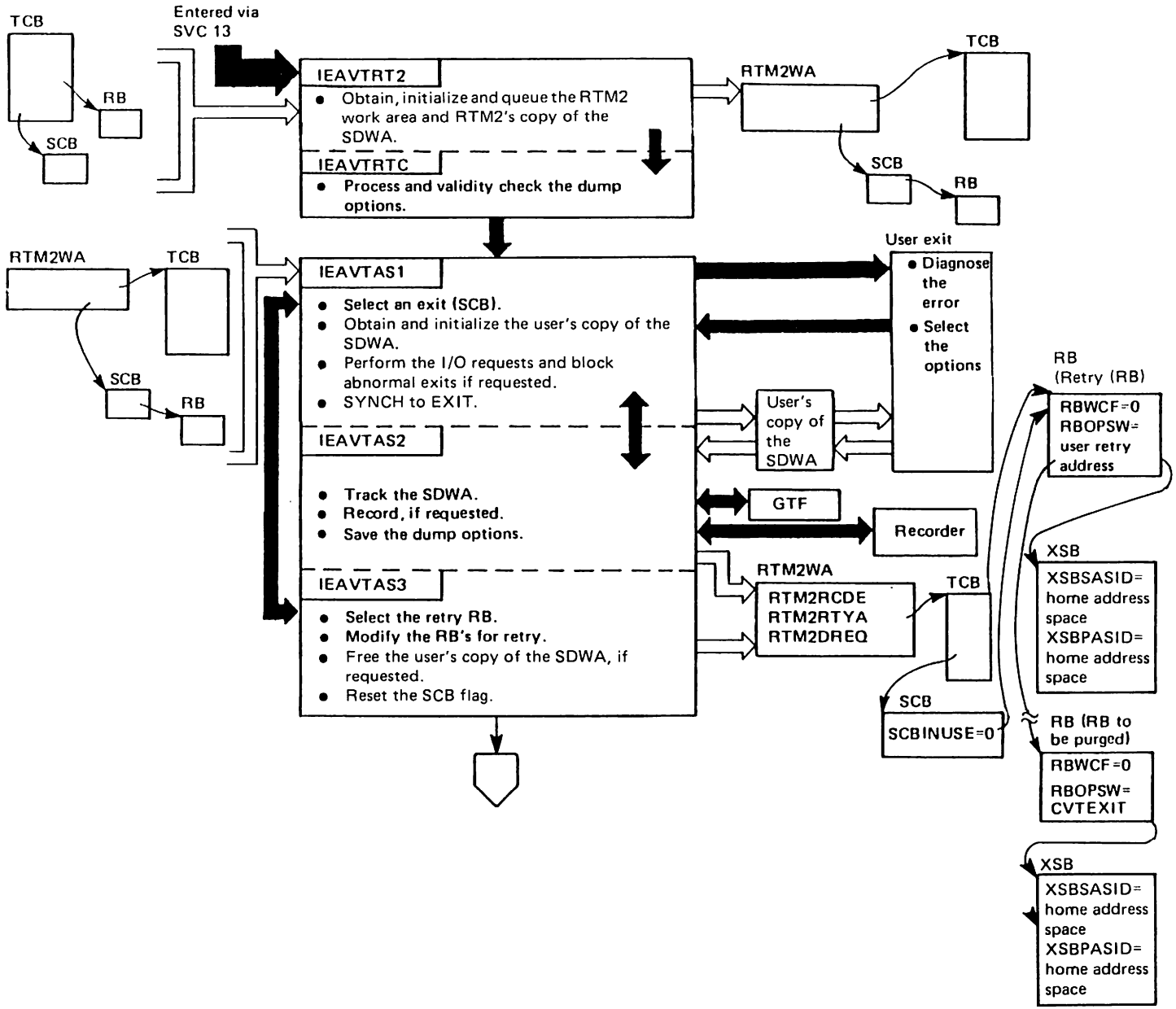


Figure 7 (Part 1 of 2). Retry

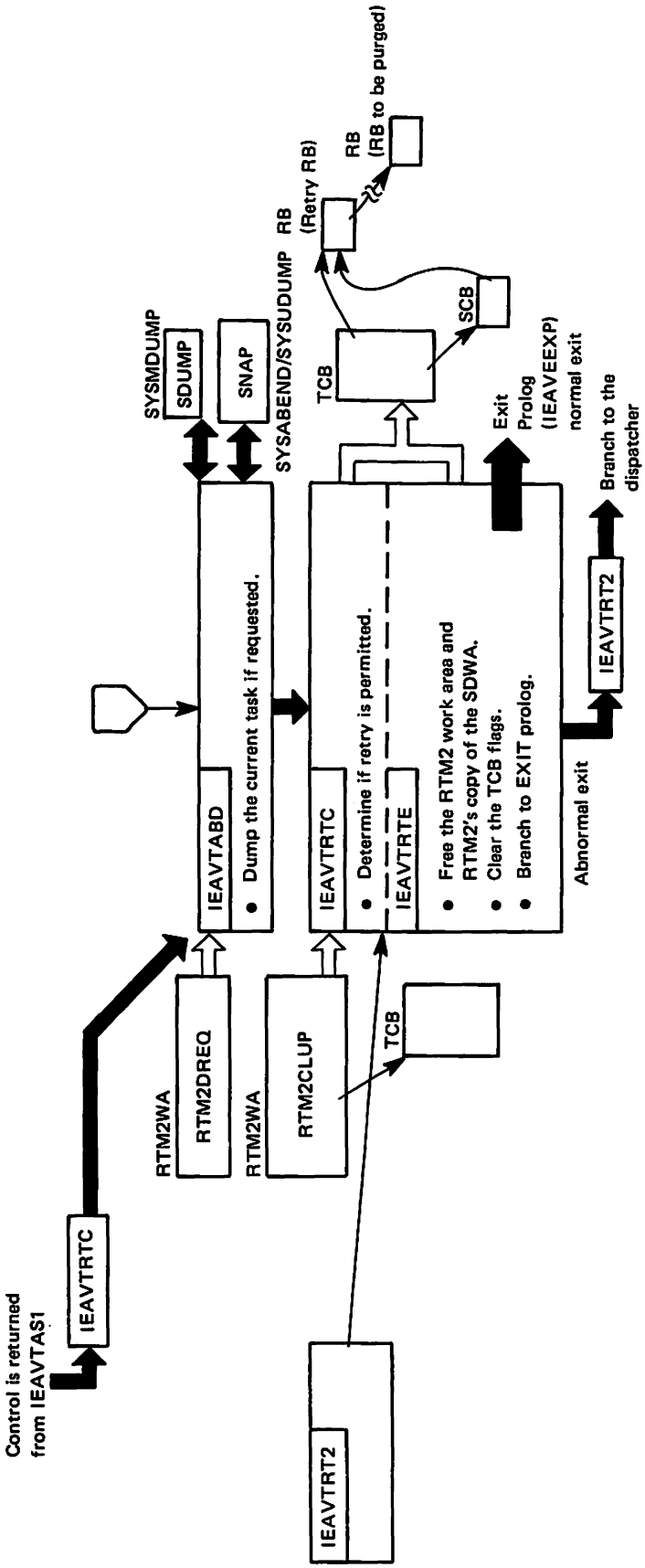


Figure 7 (Part 2 of 2). Retry

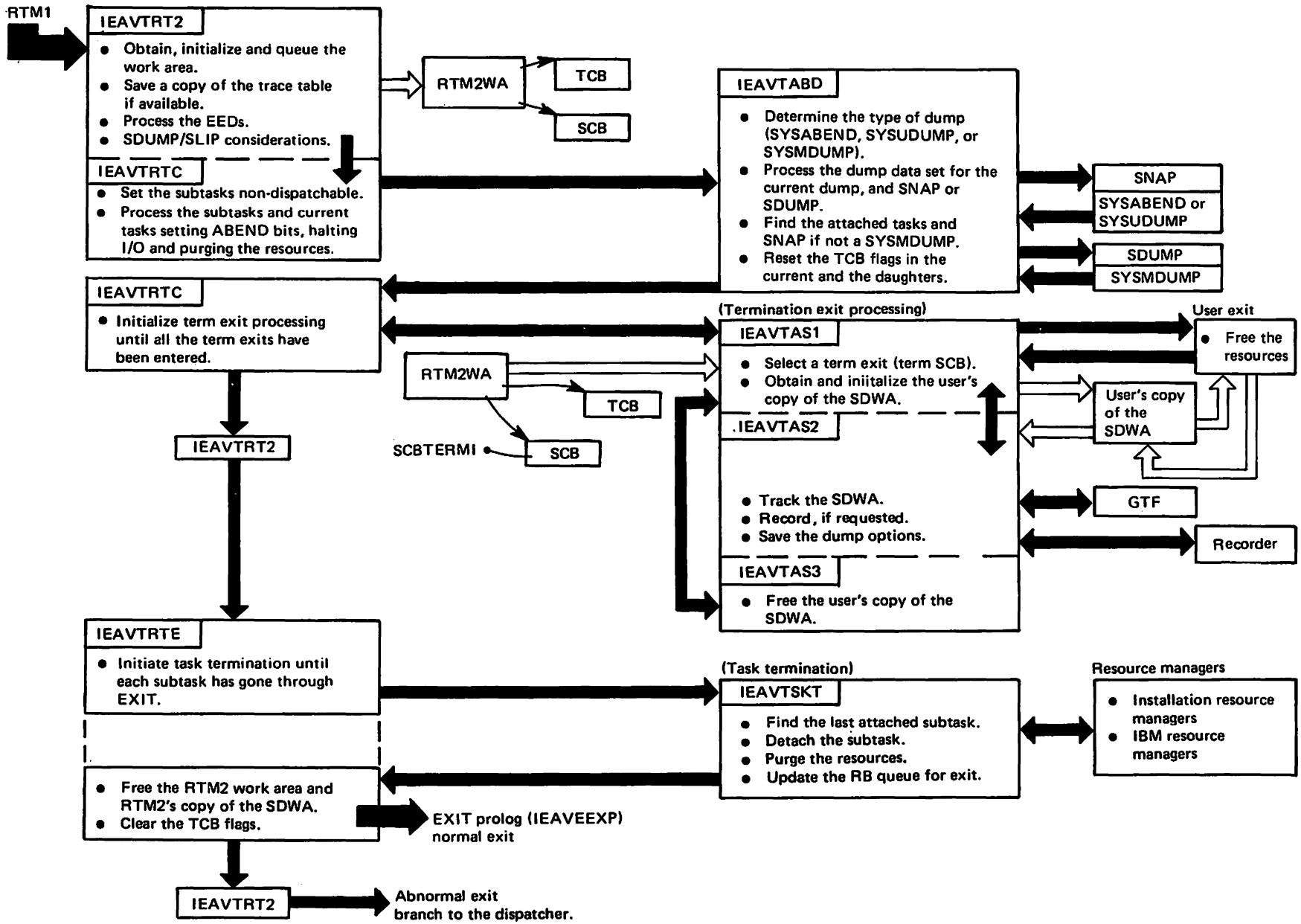


Figure 8. Cancel

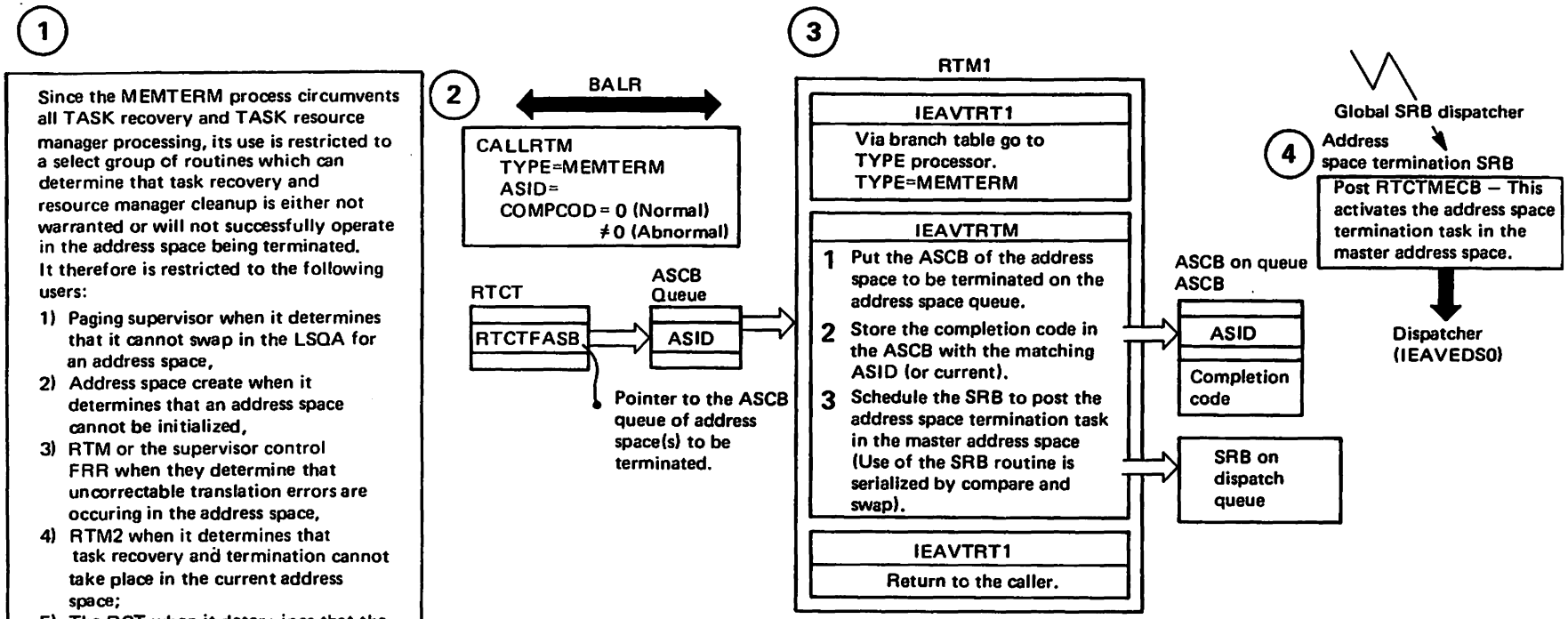


Figure 9 (Part 1 of 2). The Process of Terminating an Address Space

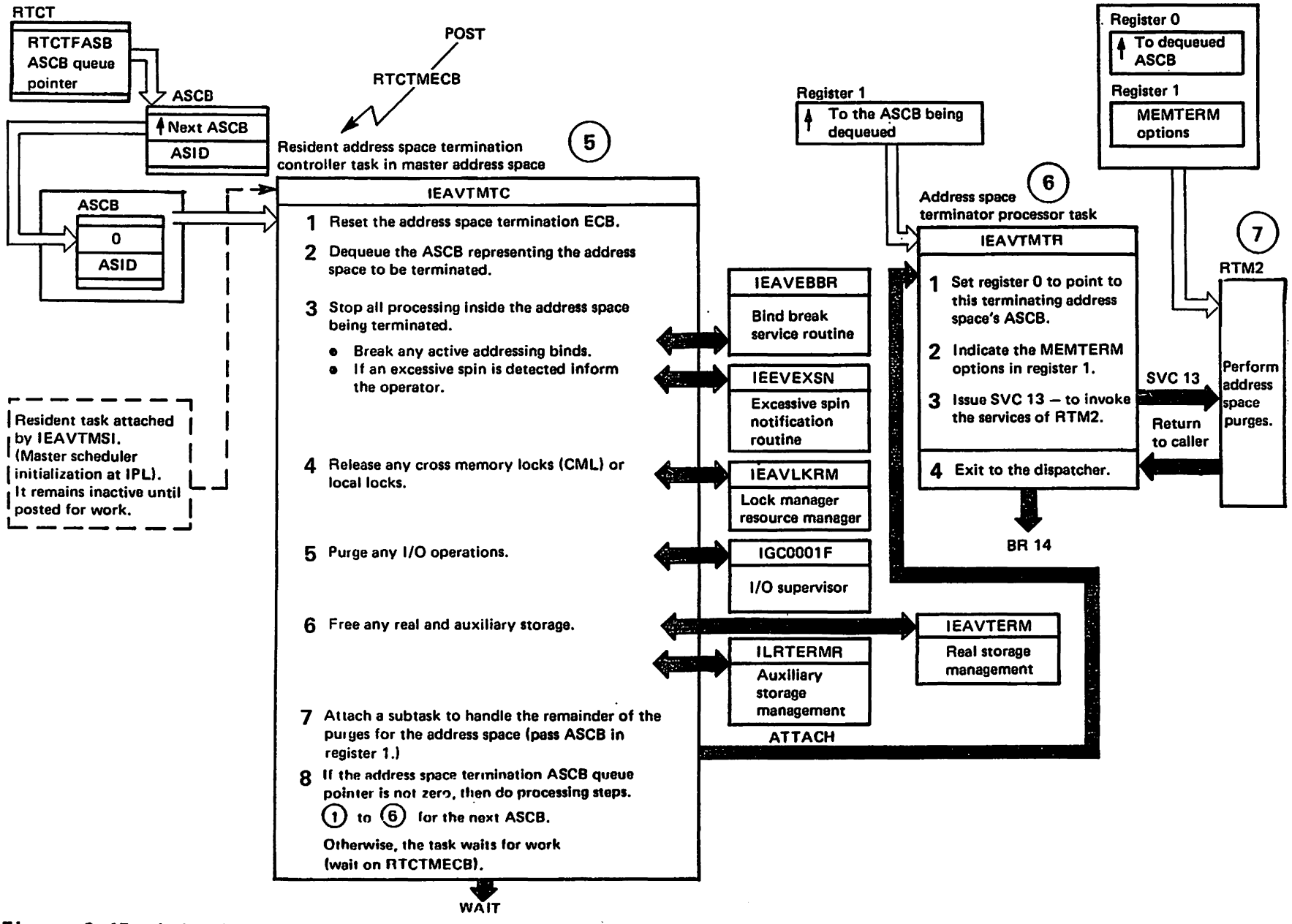


Figure 9 (Part 2 of 2). The Process of Terminating an Address Space

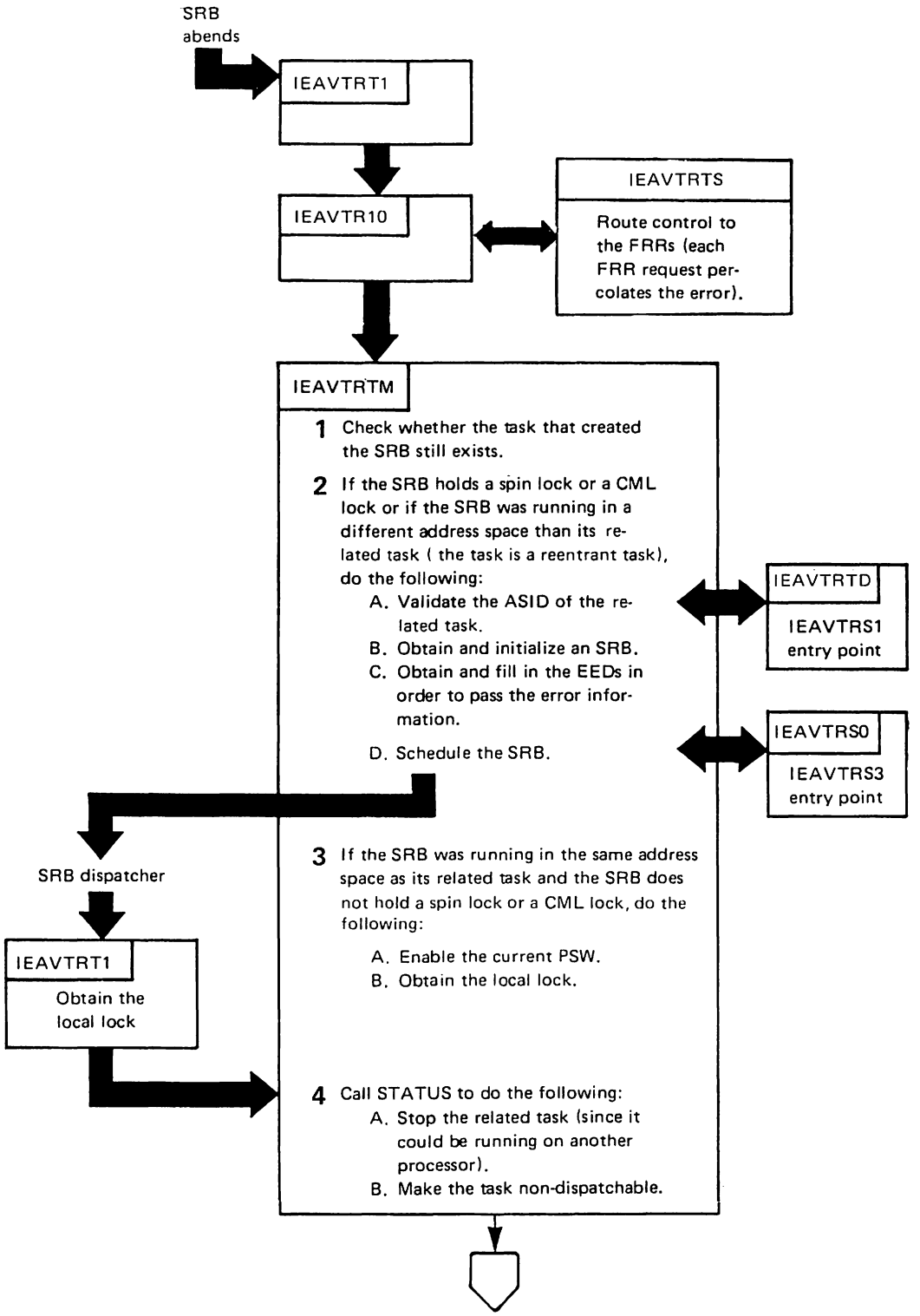


Figure 10 (Part 1 of 2). SRB to Task Percolation

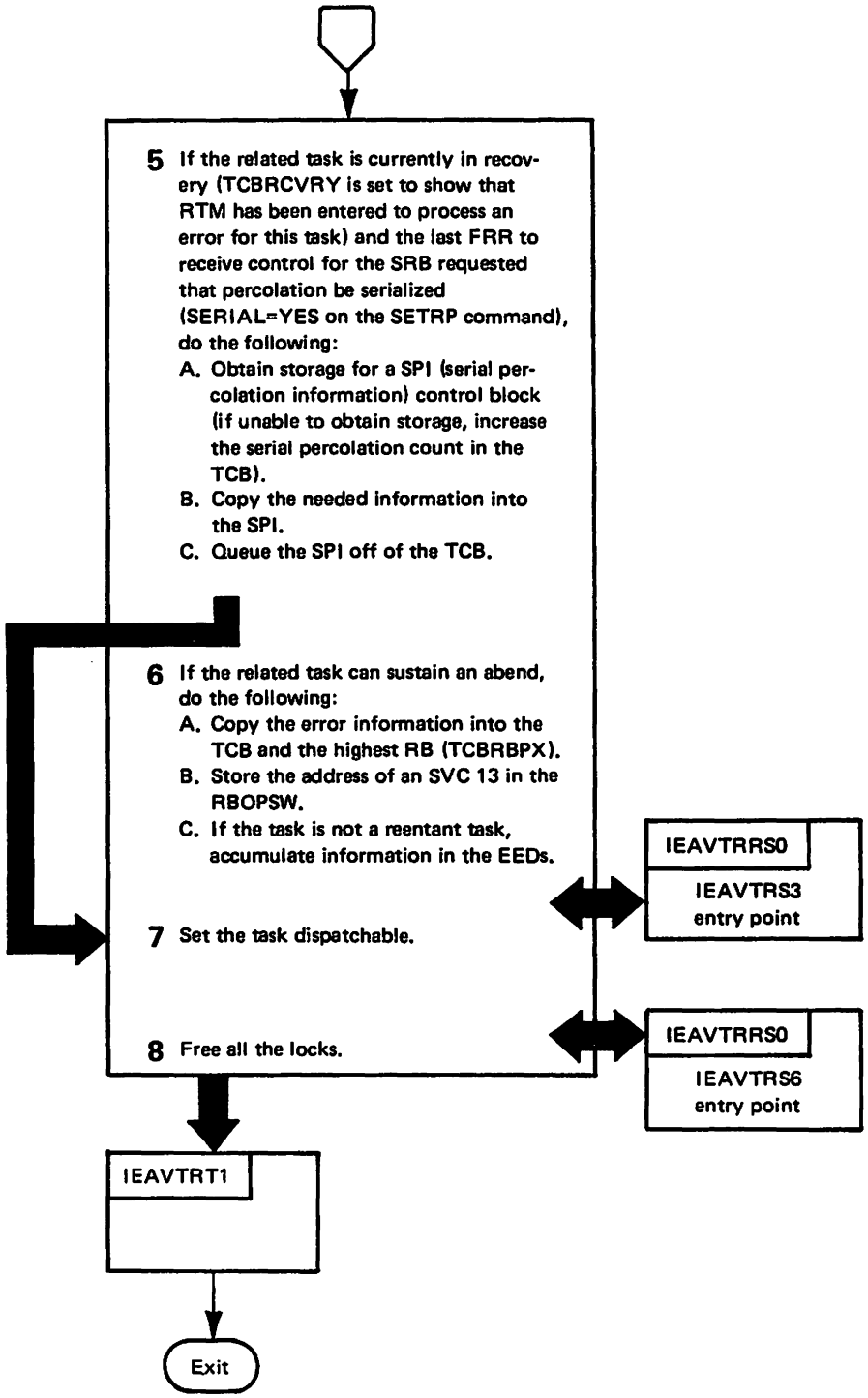


Figure 10 (Part 2 of 2). SRB to Task Percolation

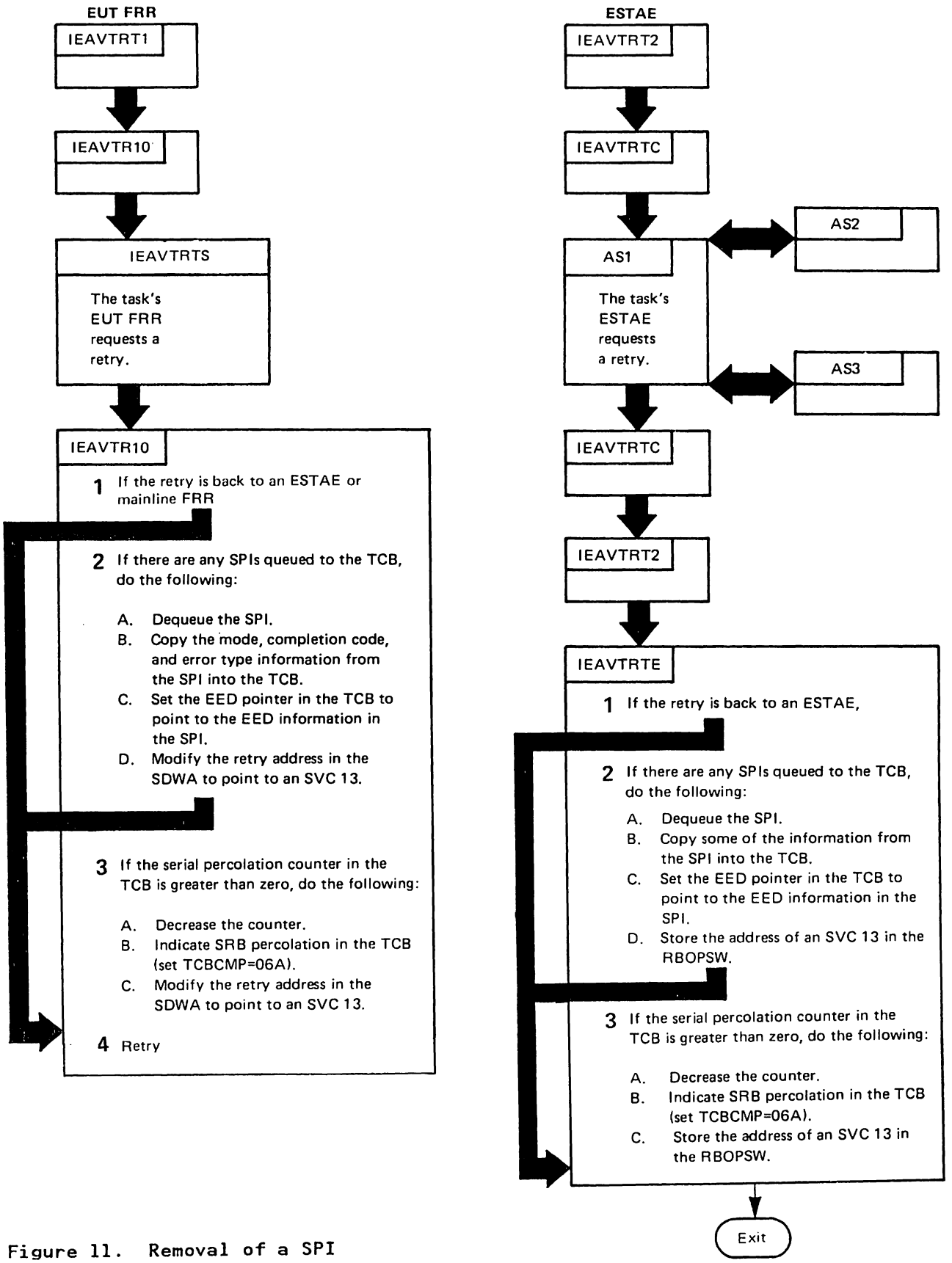


Figure 11. Removal of a SPI

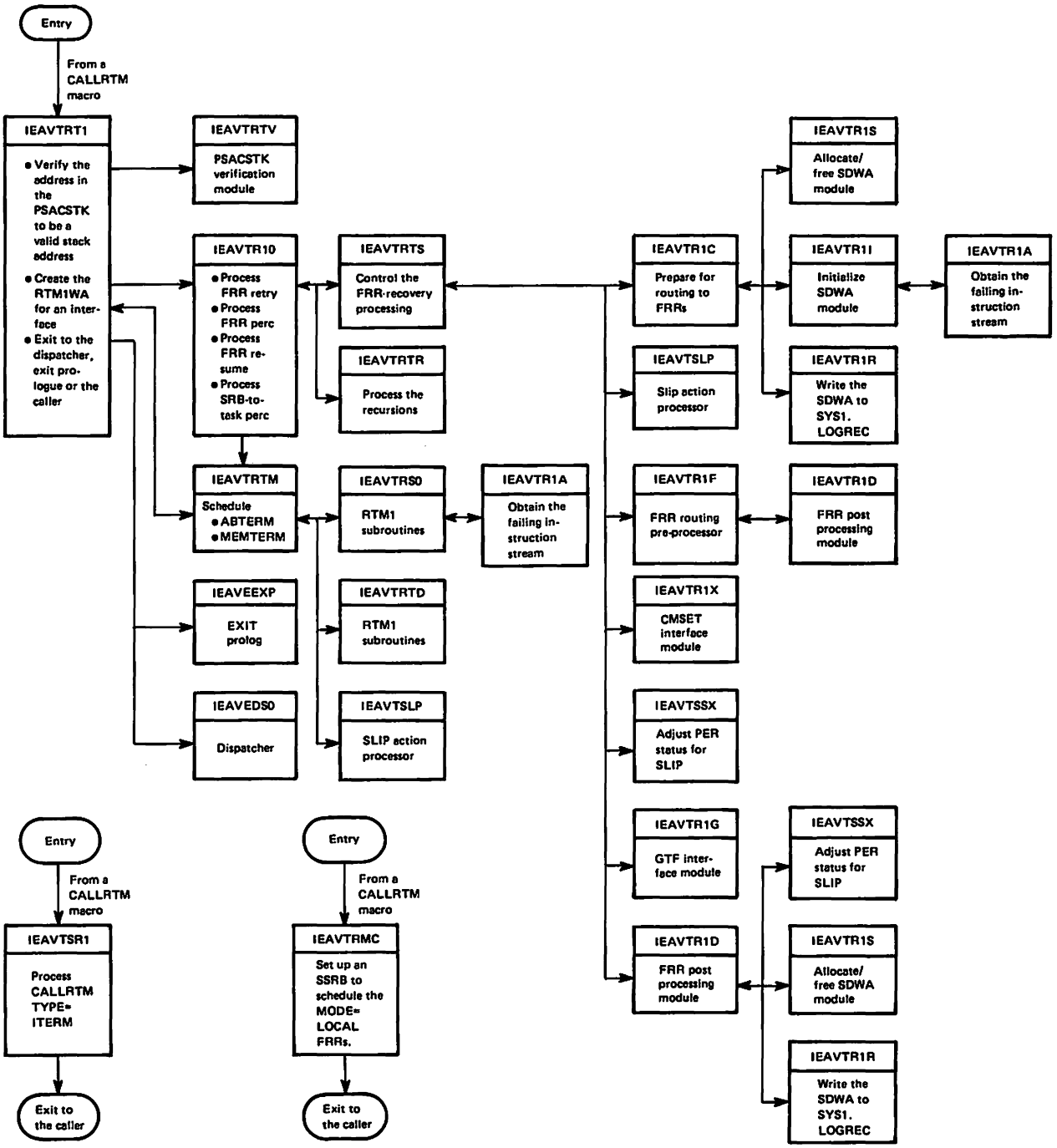


Figure 12. RTM1 Module Flow and Basic Functions Performed

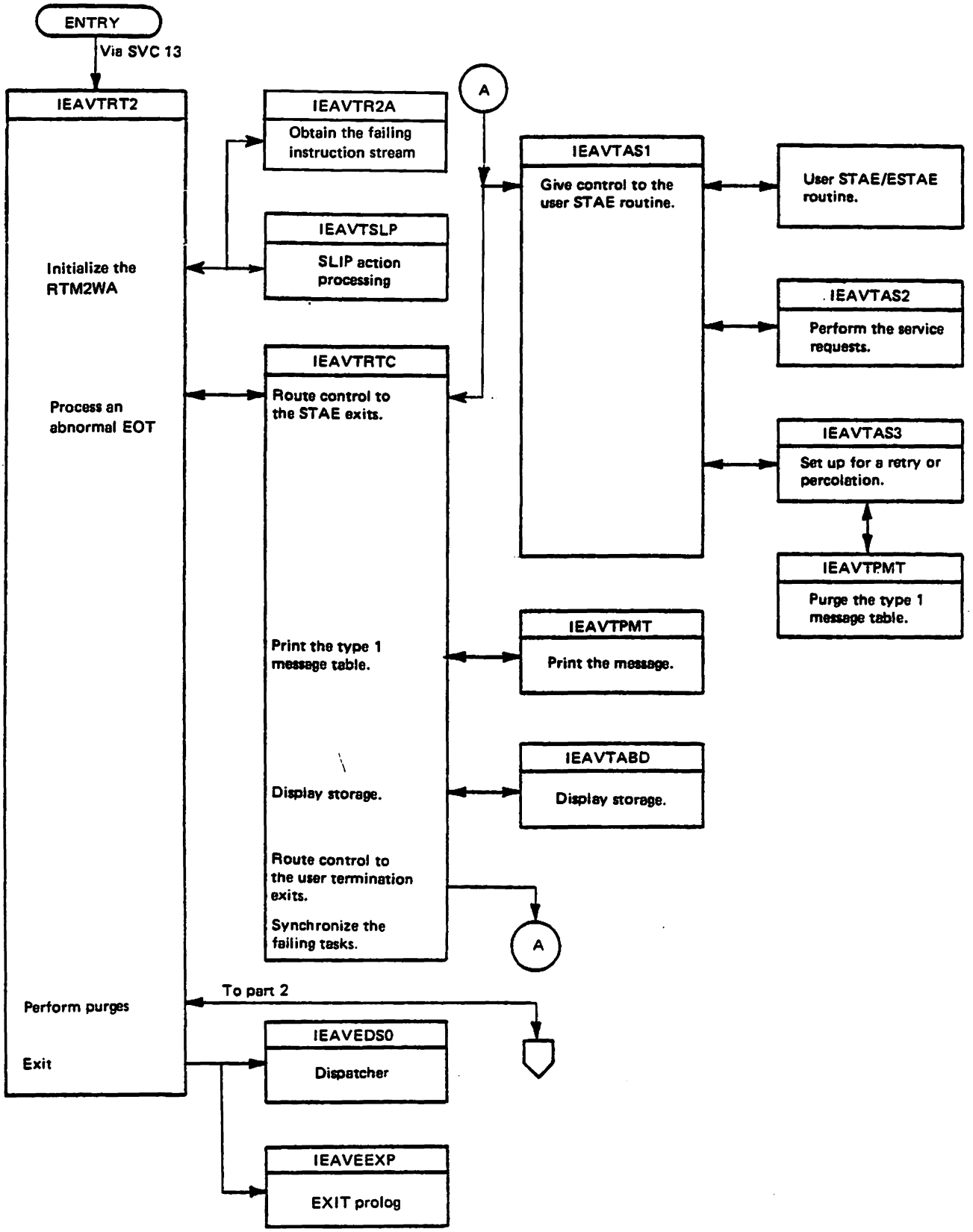


Figure 13 (Part 1 of 2). RTM2 Module Flow and Basic Functions Performed

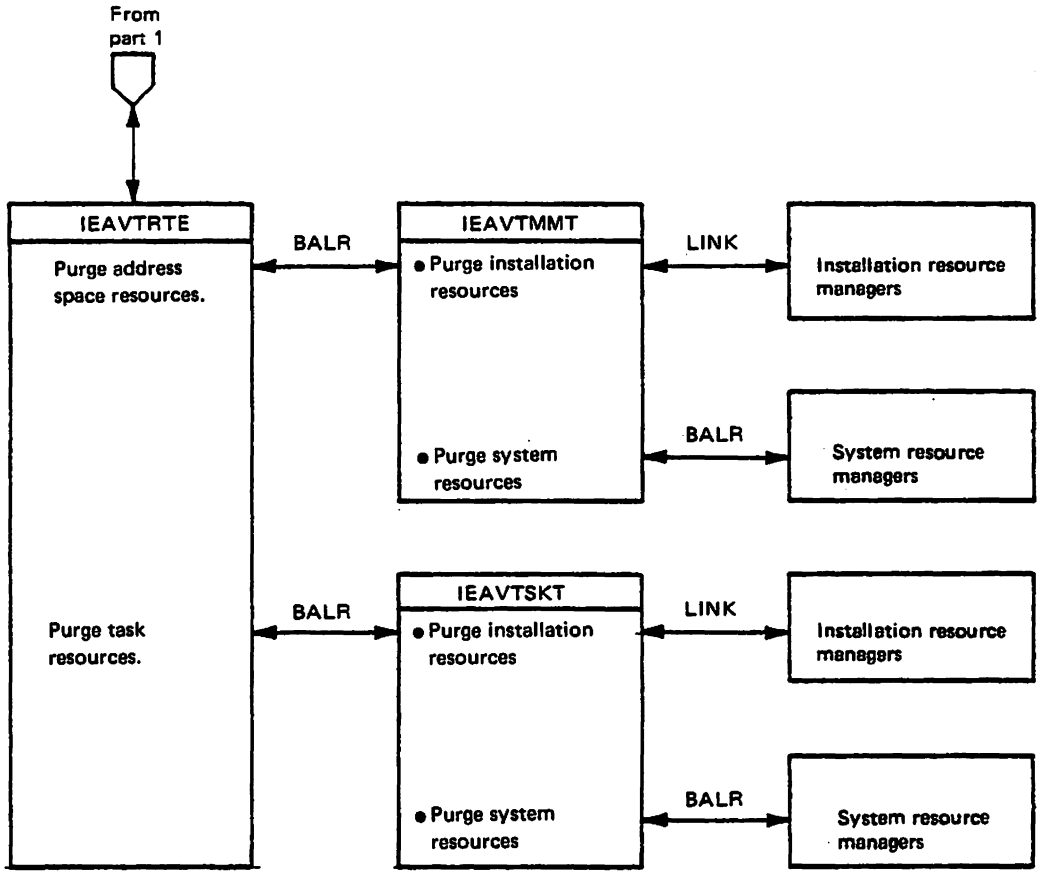


Figure 13 (Part 2 of 2). RTM2 Module Flow and Basic Functions Performed

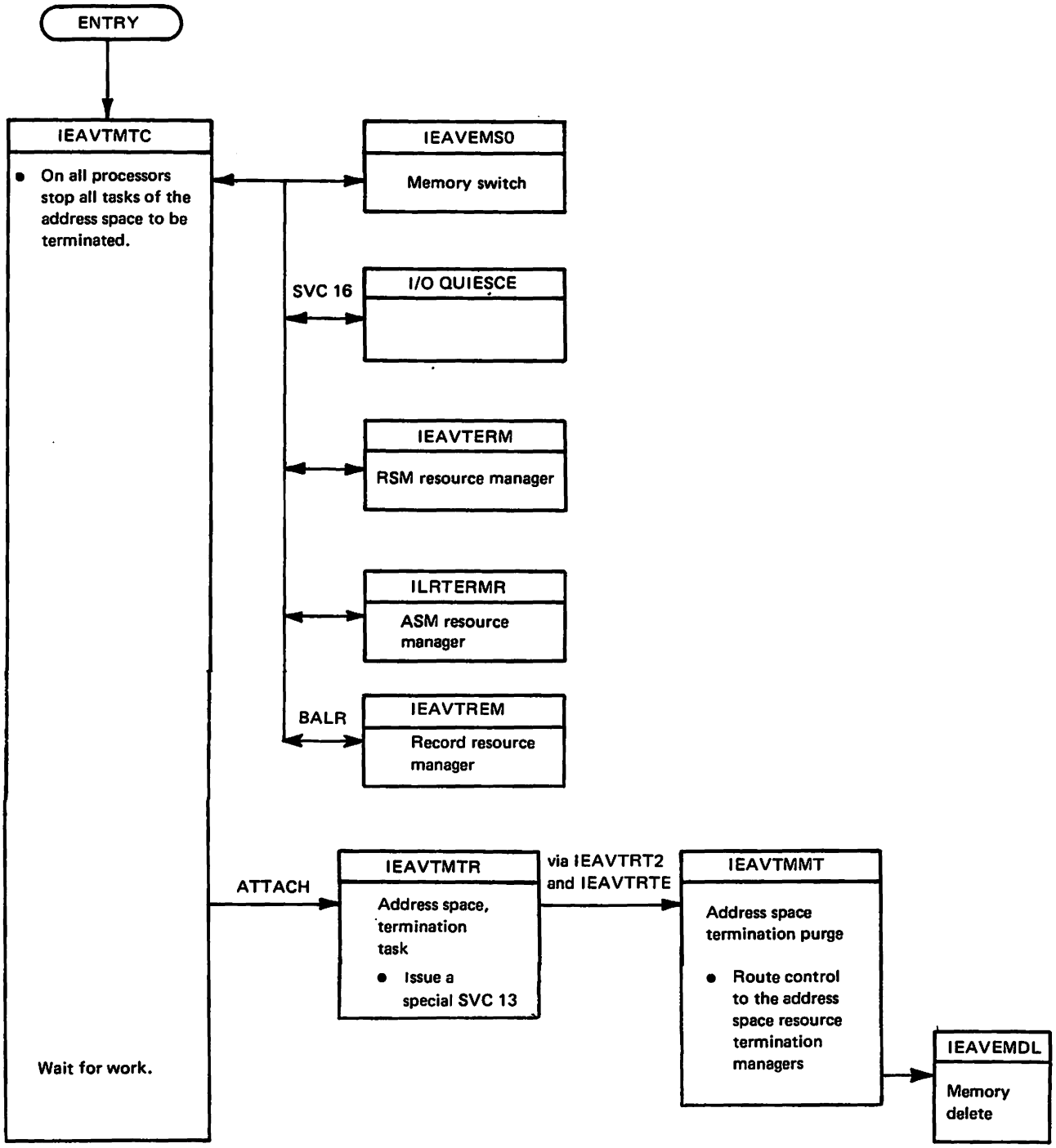


Figure 14. Address Space Termination Module Flow

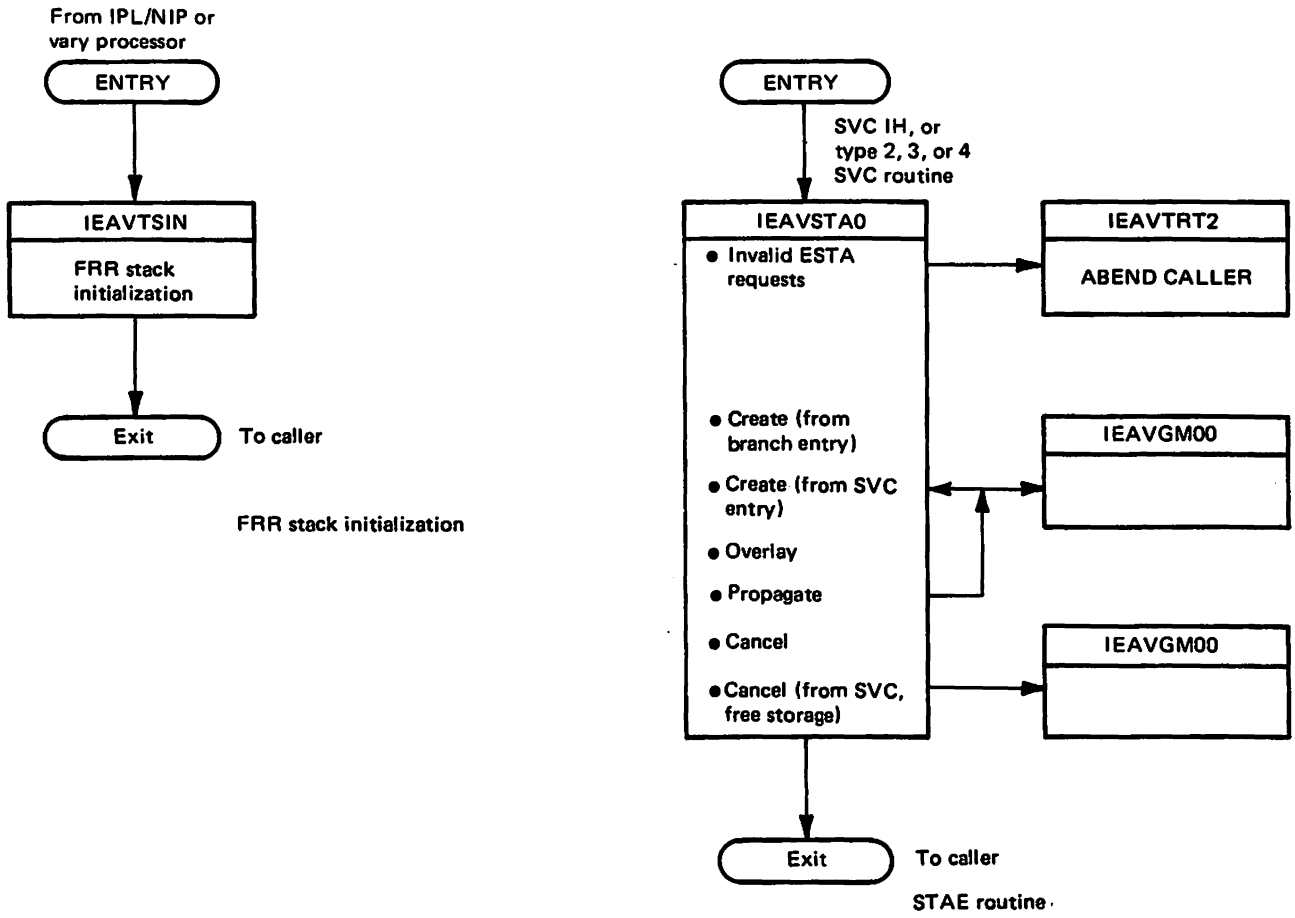


Figure 15 (Part 1 of 3). RTM Services Module Flow

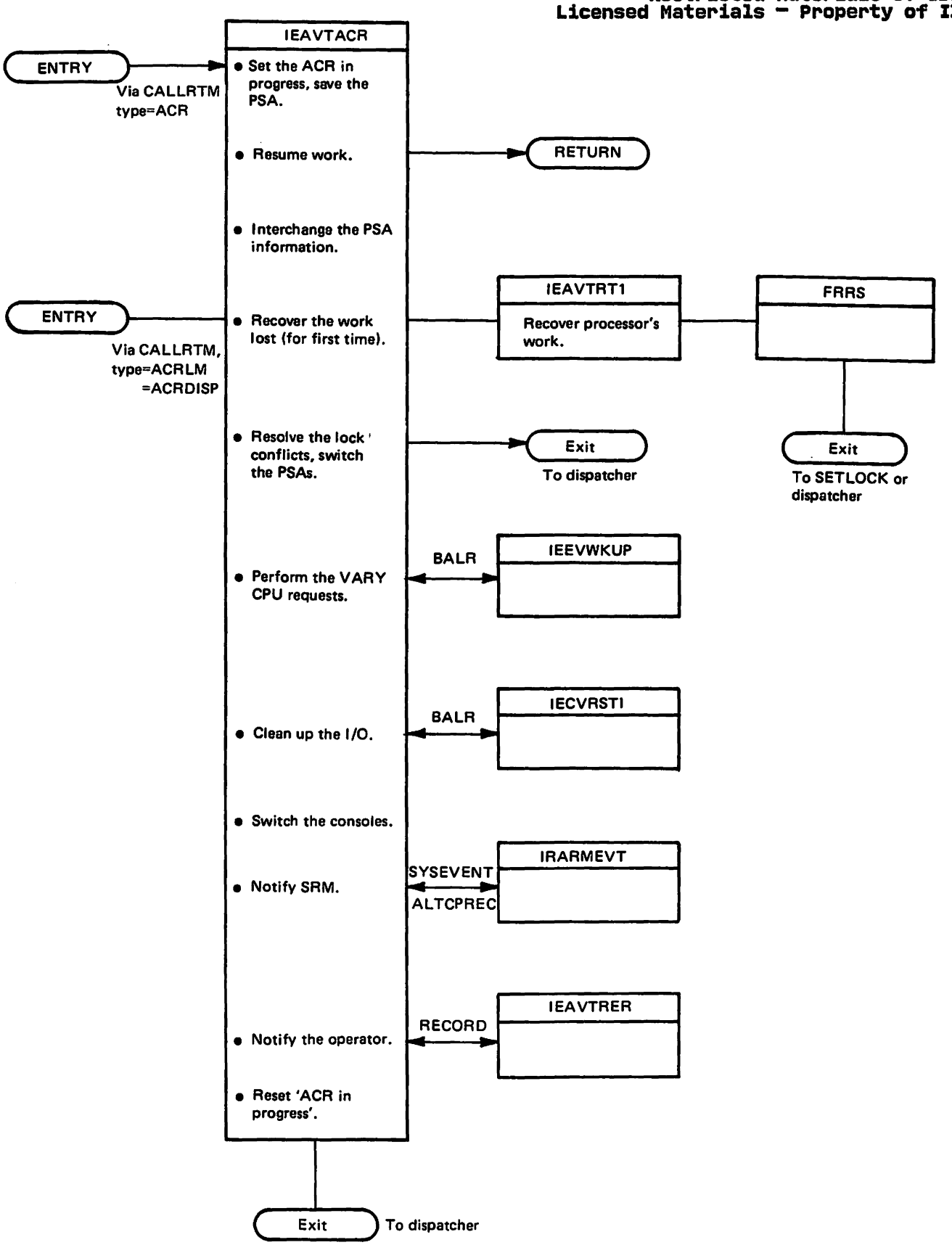


Figure 15 (Part 2 of 3). RTM Services Module Flow

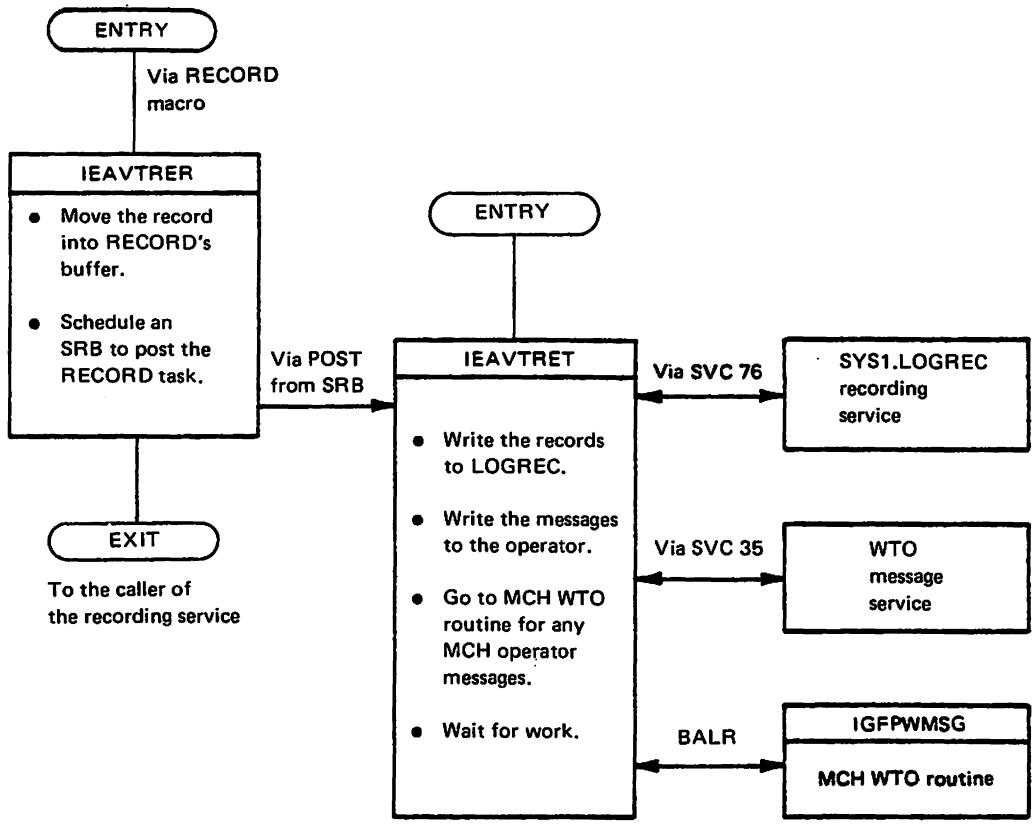


Figure 15 (Part 3 of 3). RTM Services Module Flow

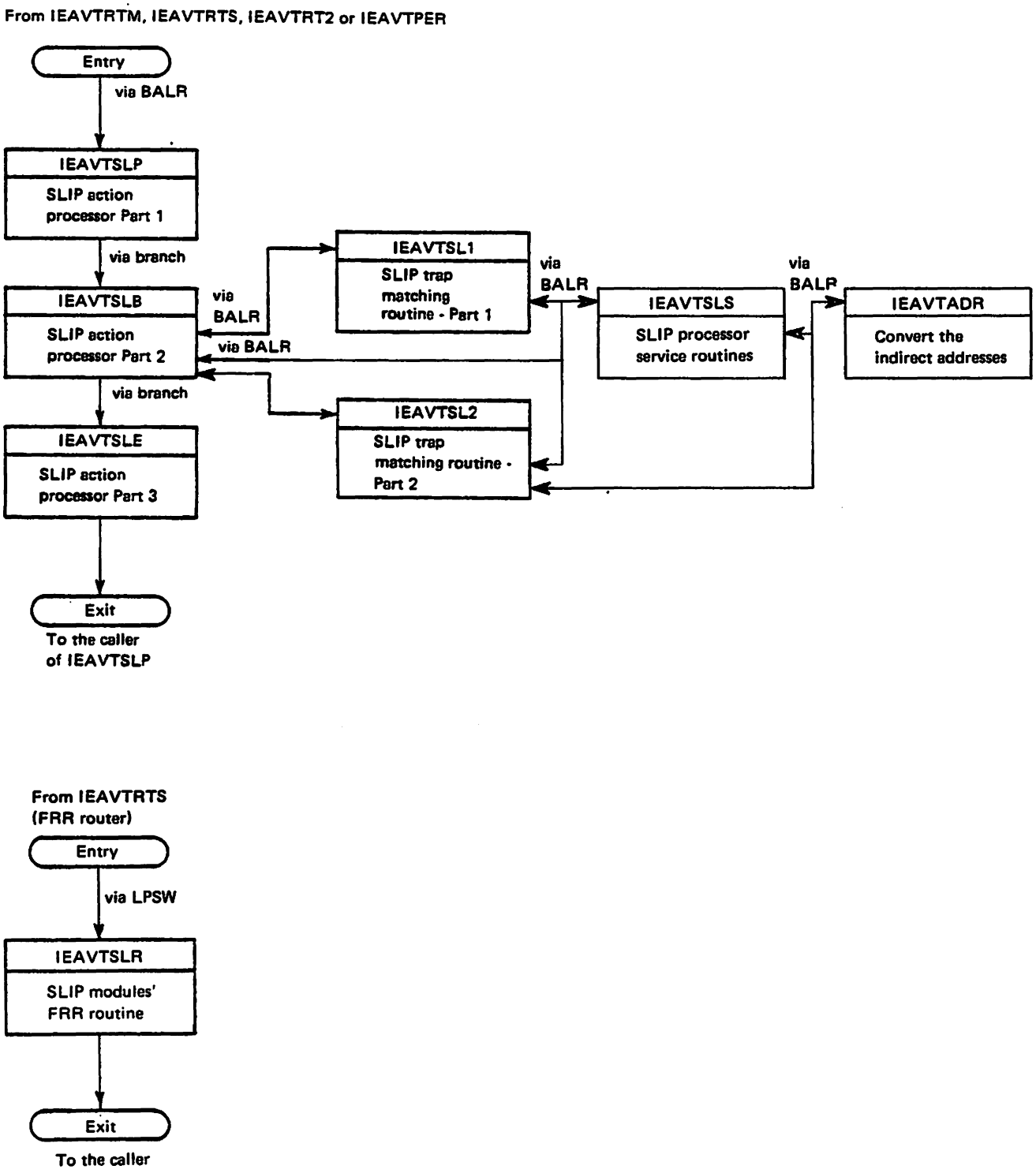


Figure 16. SLIP Action Processing Module Flow

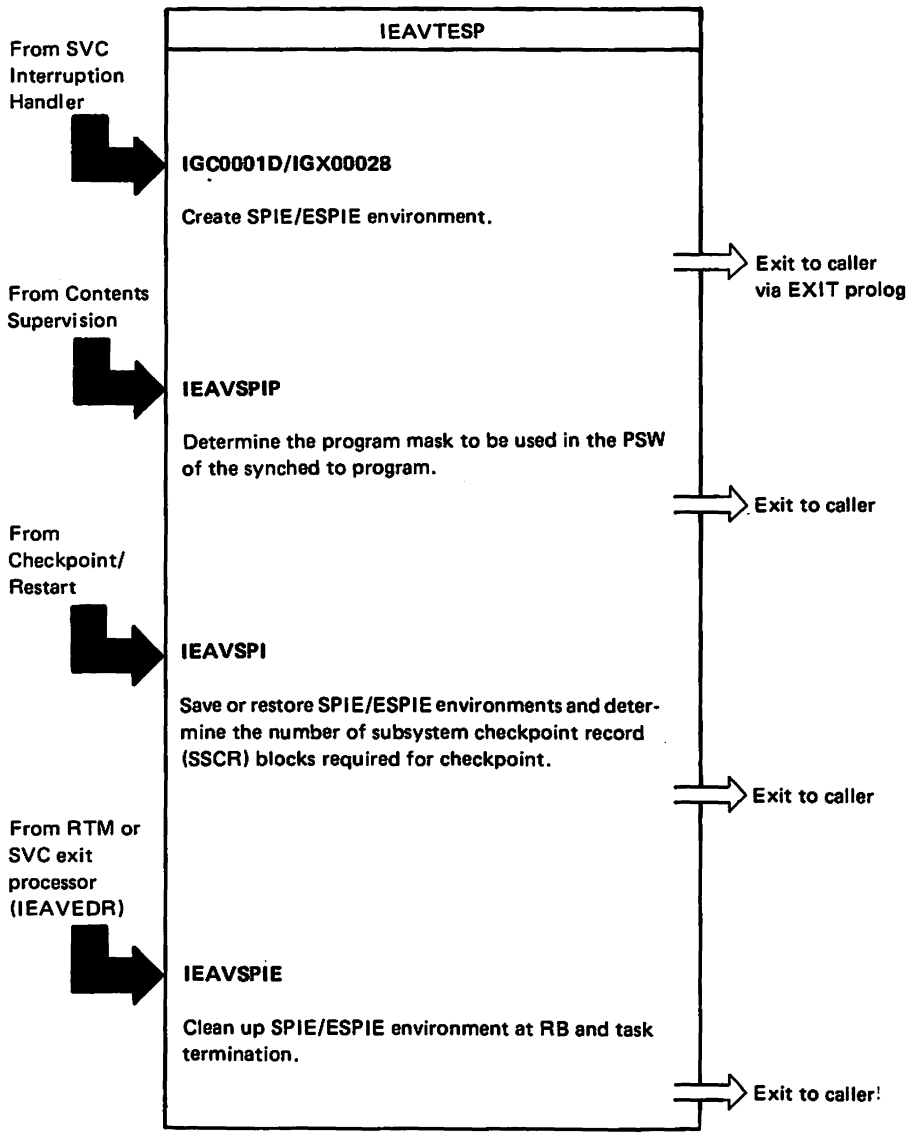


Figure 17 (Part 1 of 2). SPIE/ESPIE Module Flow

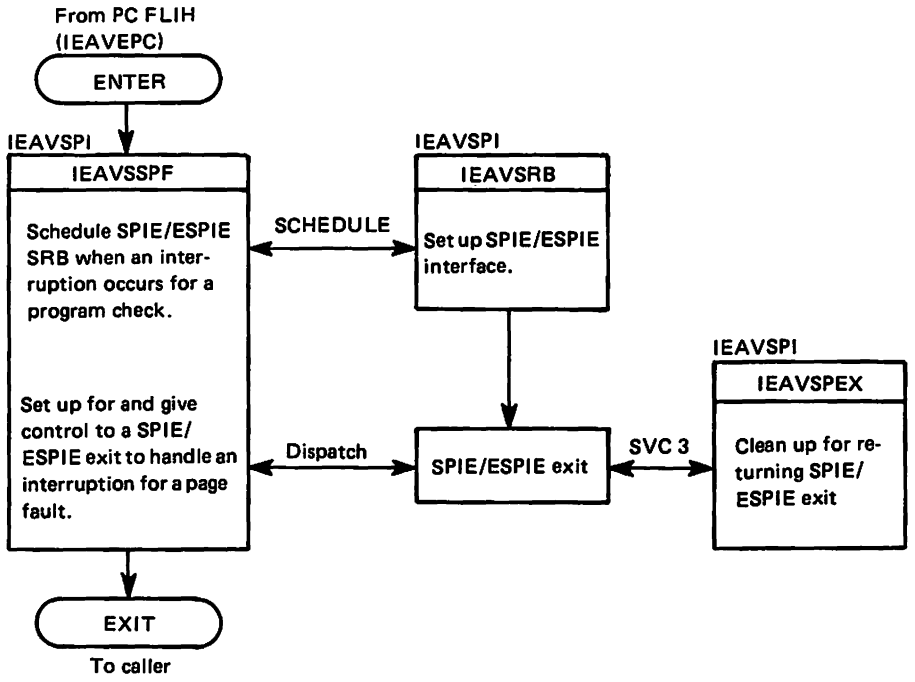
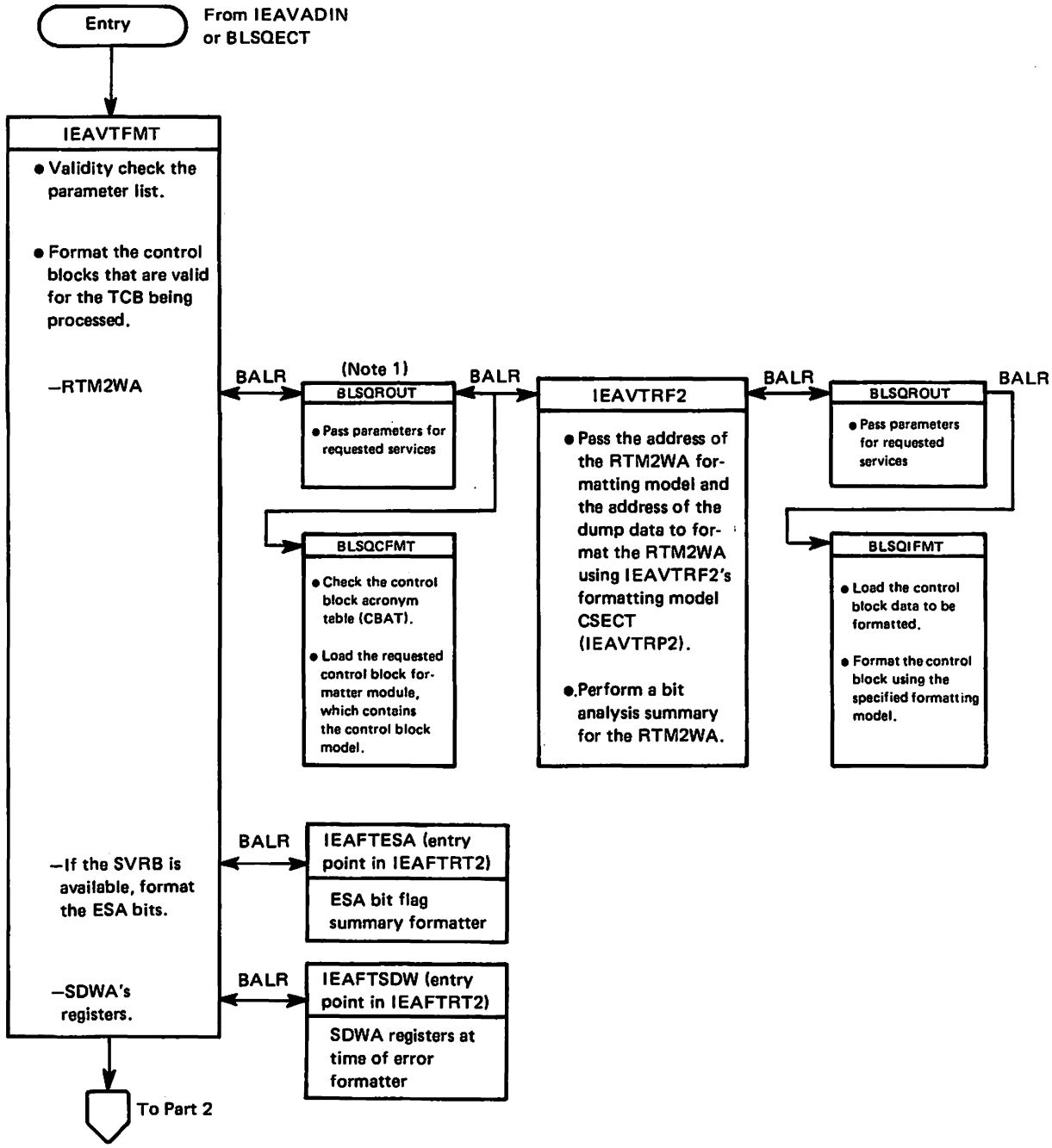


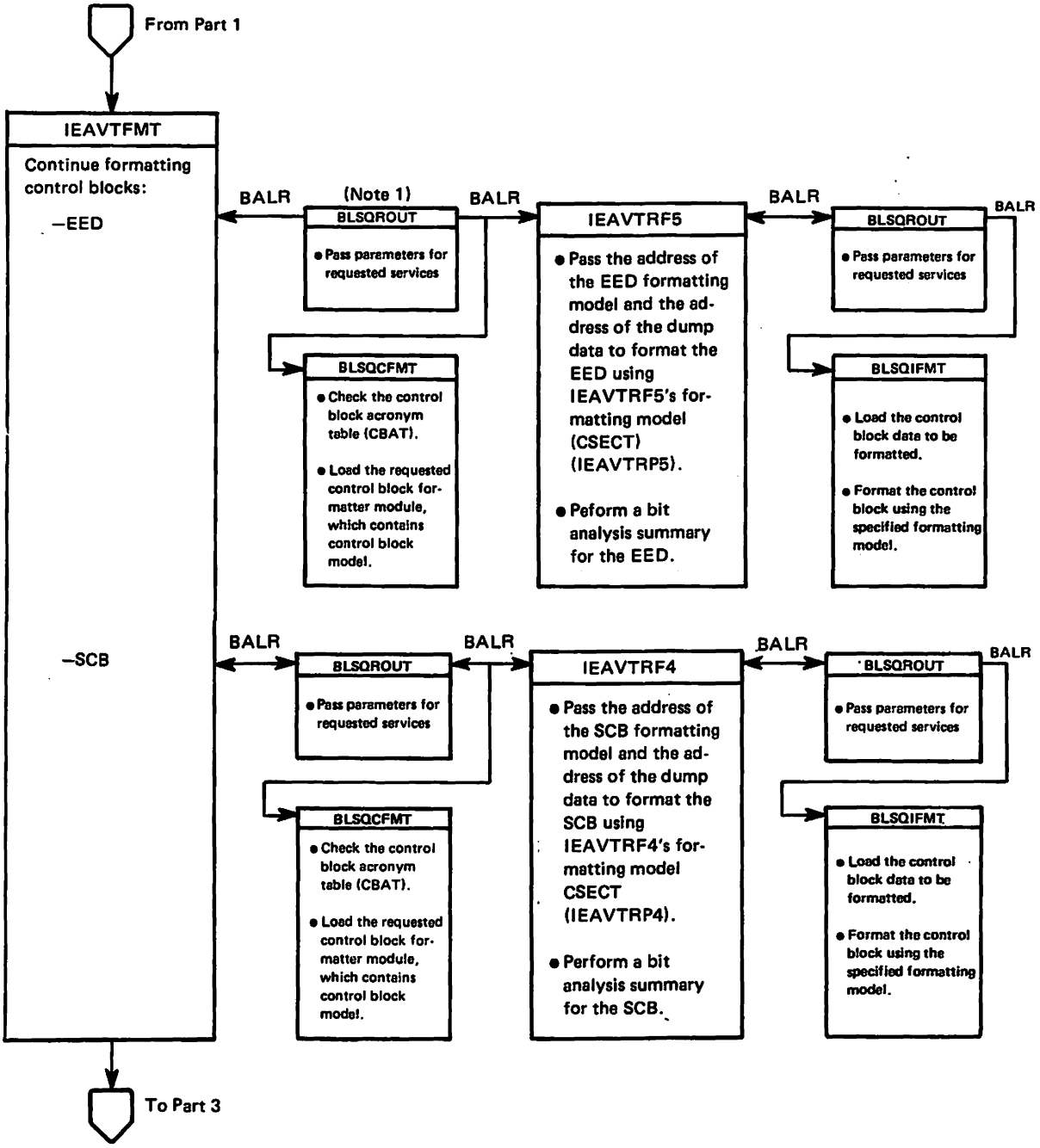
Figure 17 (Part 2 of 2). SPIE/ESPIE Module Flow



Note:

1. The BLS modules are documented in IPCS Logic and Diagnosis.

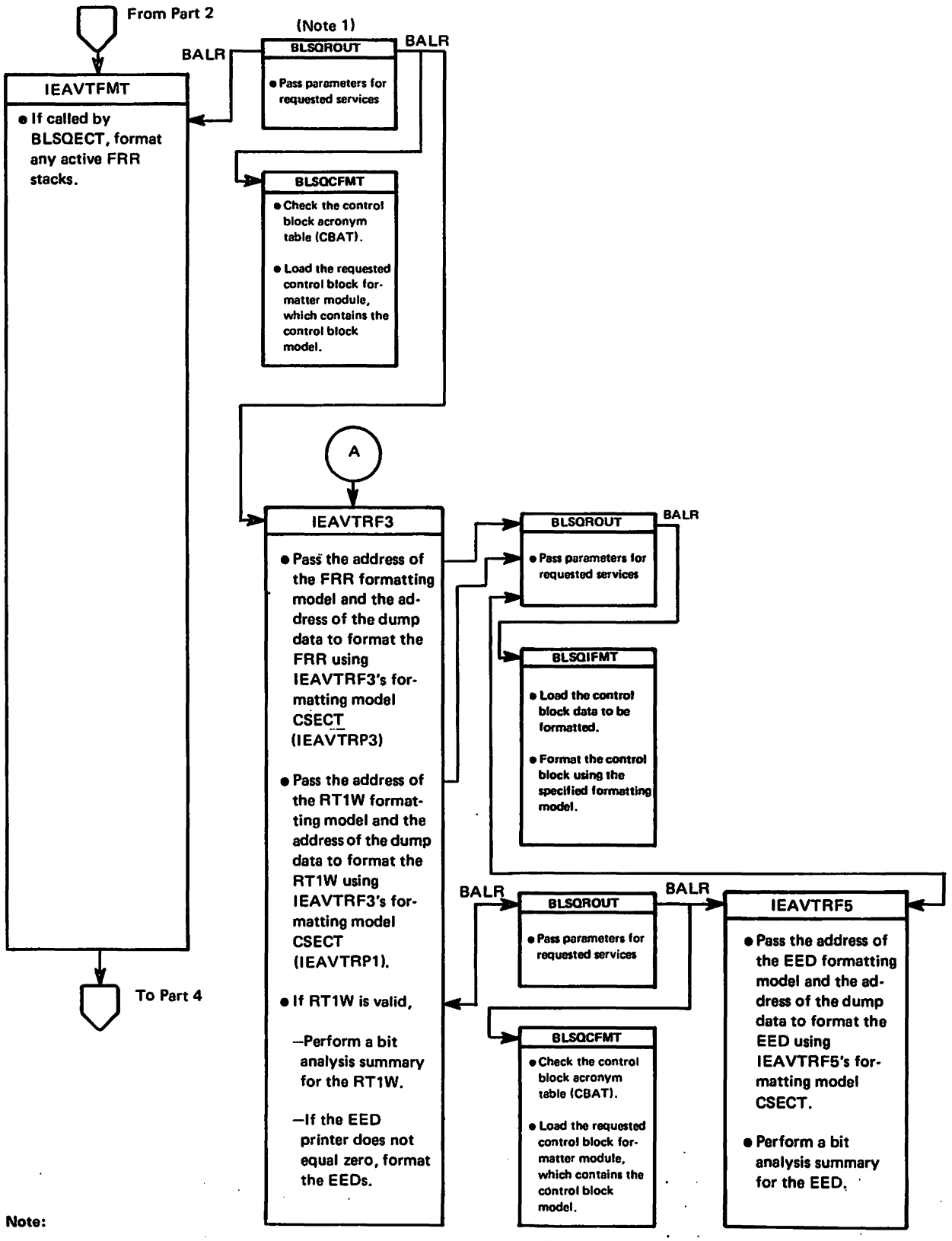
Figure 18 (Part 1 of 4). RTM Control Block Formatter



Note:

1. The BLS modules are documented in IPCS Logic and Diagnosis.

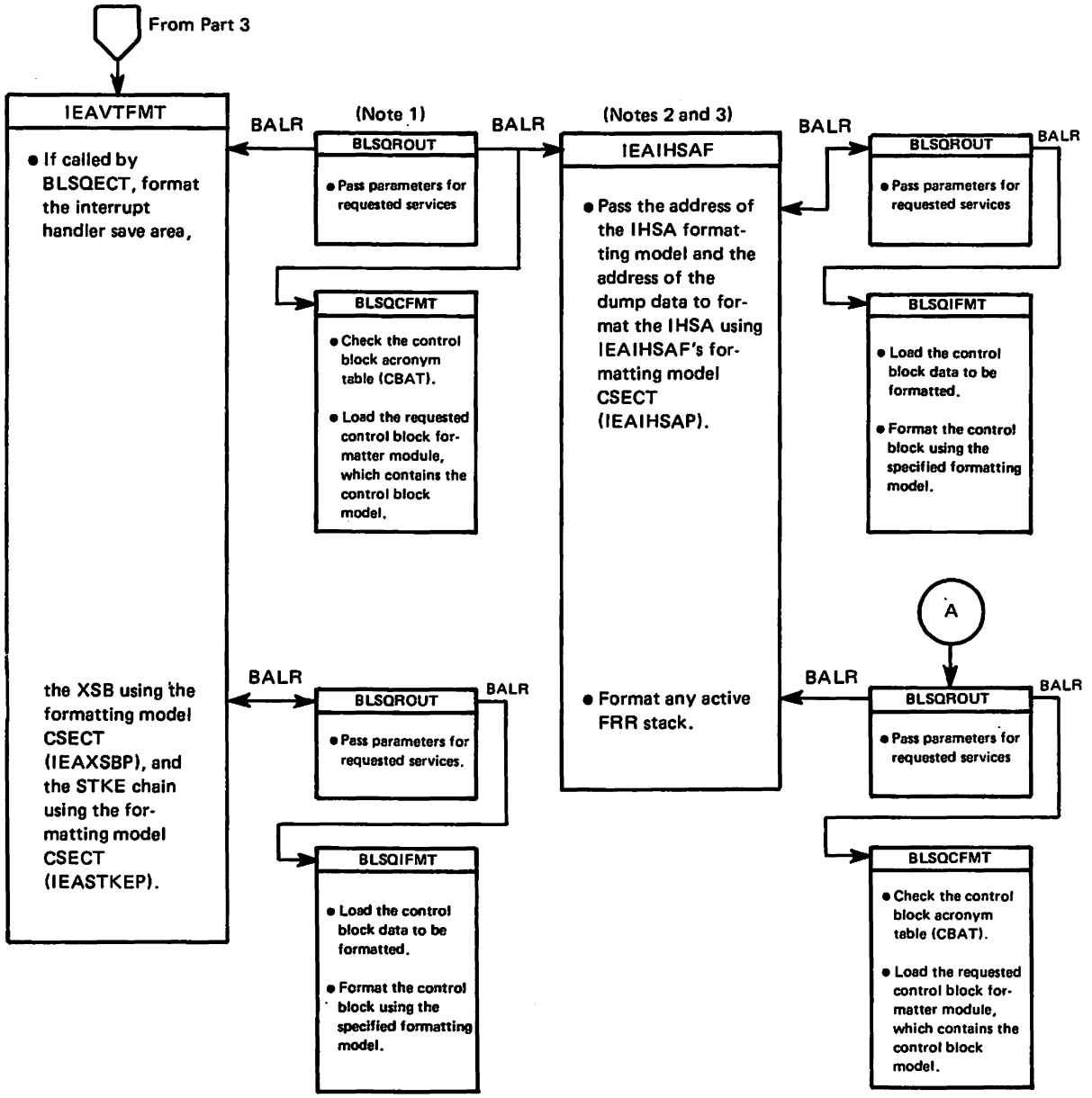
Figure 18 (Part 2 of 4). RTM Control Block Formatter



Note:

1. The BLS modules are documented in IPCS Logic and Diagnosis.

Figure 18 (Part 3 of 4). RTM Control Block Formatter



Notes:

1. The BLS modules are documented in IPCS Logic and Diagnosis.
2. IEAIHSAF is the supervisor module's formatter.
3. SDUMP calls IEAIHSAF directly.

Figure 18 (Part 4 of 4). RTM Control Block Formatter

METHOD OF OPERATION

This section contains logic diagrams for the modules in this component.

The first two diagrams are overviews of RTM1 and RTM2 processing. The remaining diagrams are in alphabetic order by module name, and the diagrams use either hipo format or prologue format.

The following figure shows the symbols used in hipo format logic diagrams. The relative size and the order of fields in control block illustrations do not always represent the actual size and format of the control block.

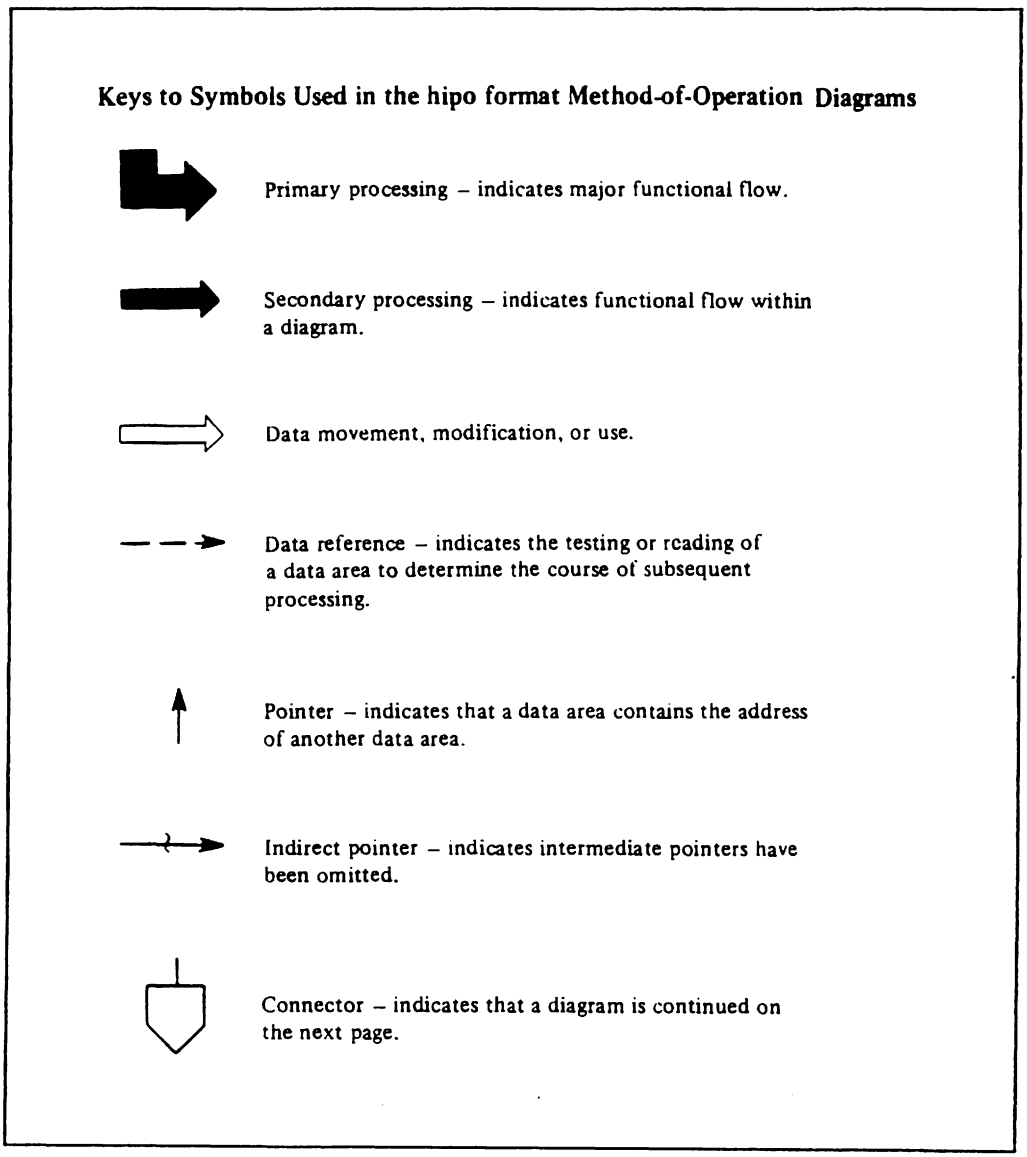


Figure 19. Key to Hipo Logic Diagrams

The prolog format diagrams contain detailed information that is broken down into four different headings. The four headings and the topics they document are:

Module Description, which includes:

- Descriptive name
- Function (of the entire module)
- Entry point names, which includes:
  - Purpose (of the entry point)
  - Linkage
  - Callers
  - Input
  - Output
  - Exit normal
  - Exit error, if any
- External references, which includes:
  - Routines
  - Data areas, if any
  - Control blocks
- Tables
- Serialization

Note: Brief RTM module descriptions are also included in MVS/Extended Architecture System Logic Library: Module Descriptions, which contains module descriptions for all the MVS/Extended Architecture components described in the System Logic Library.

Module Operation, which includes:

- Operation, which explains how the module performs its function.
- Recovery operation, which explains how the module performs any recovery.

Diagnostic aids, which provide information useful for debugging program problems; this includes:

- Entry point names
- Messages
- Abend codes
- Wait state codes
- Return codes for each entry point. Within each entry point, return codes might be further categorized by exit-normal and exit-error.
- Entry register contents for each entry point
- Exit register contents for each entry point

Logic Diagram, which illustrates the processing of the module, the input it uses, the output it producs, and the flow of control. Some modules do not have a logic diagram because the processing is sufficiently explained in the module description, the module operation, and the diagnostic aids sections. Figure 20 on page RTM-55 illustrates the graphic symbols and format used in the logic diagrams.

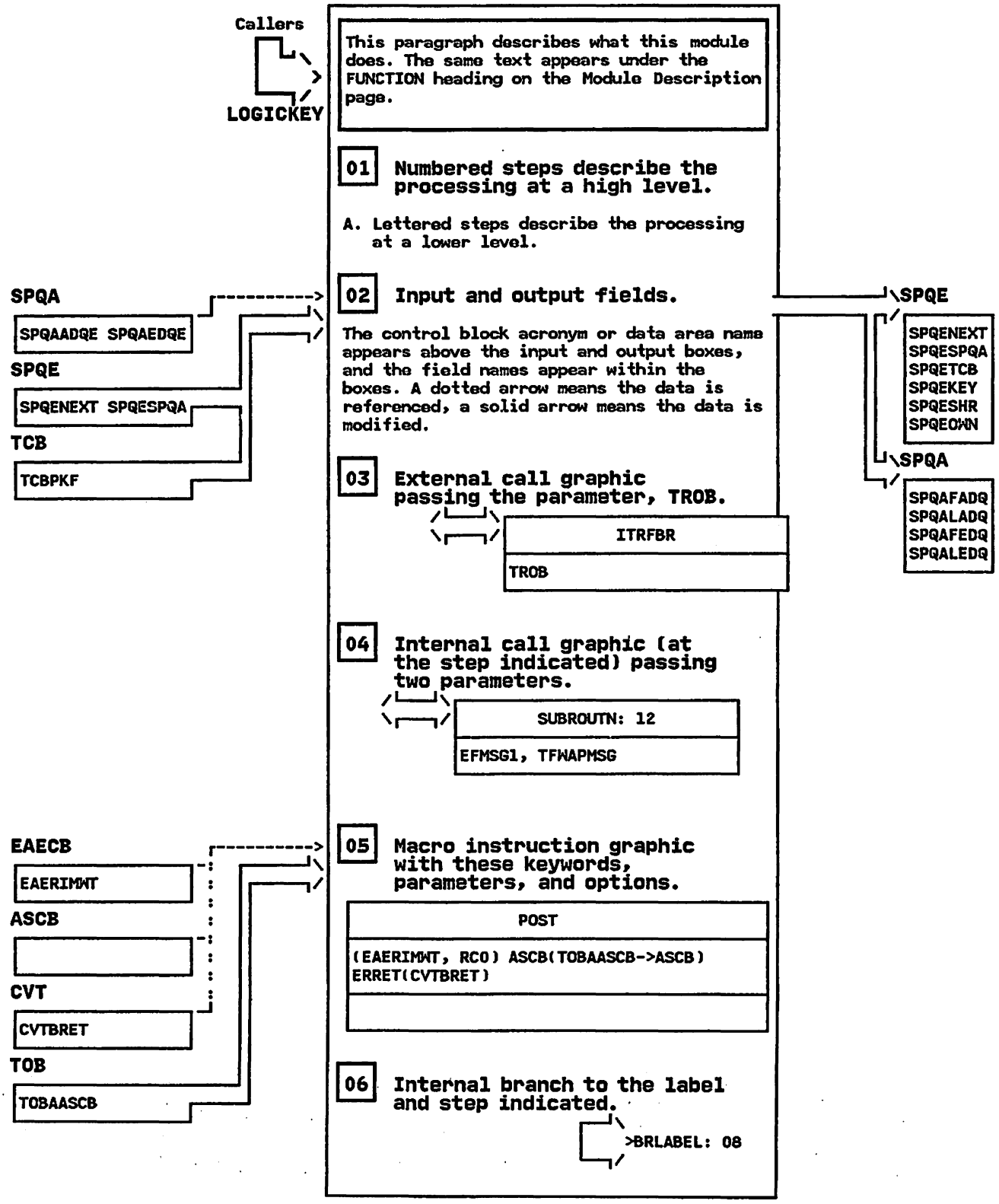


Figure 21. Key to Logic Diagrams (Part 1 of 2)

LOGICKEY - Key to the Logic Diagrams STEP 07

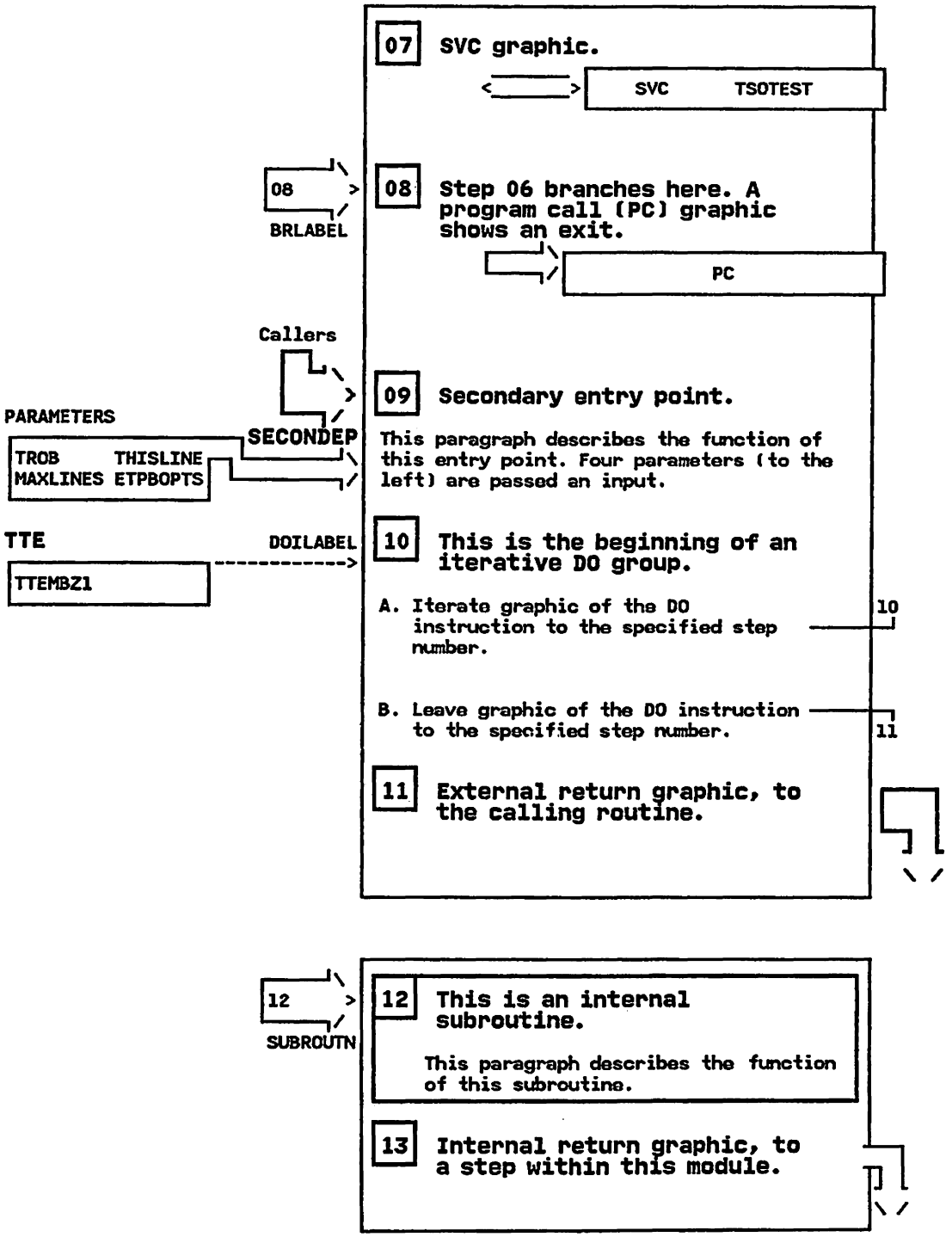


Figure 21. Key to Logic Diagrams (Part 2 of 2)

RTM1 Overview (Part 1 of 2)

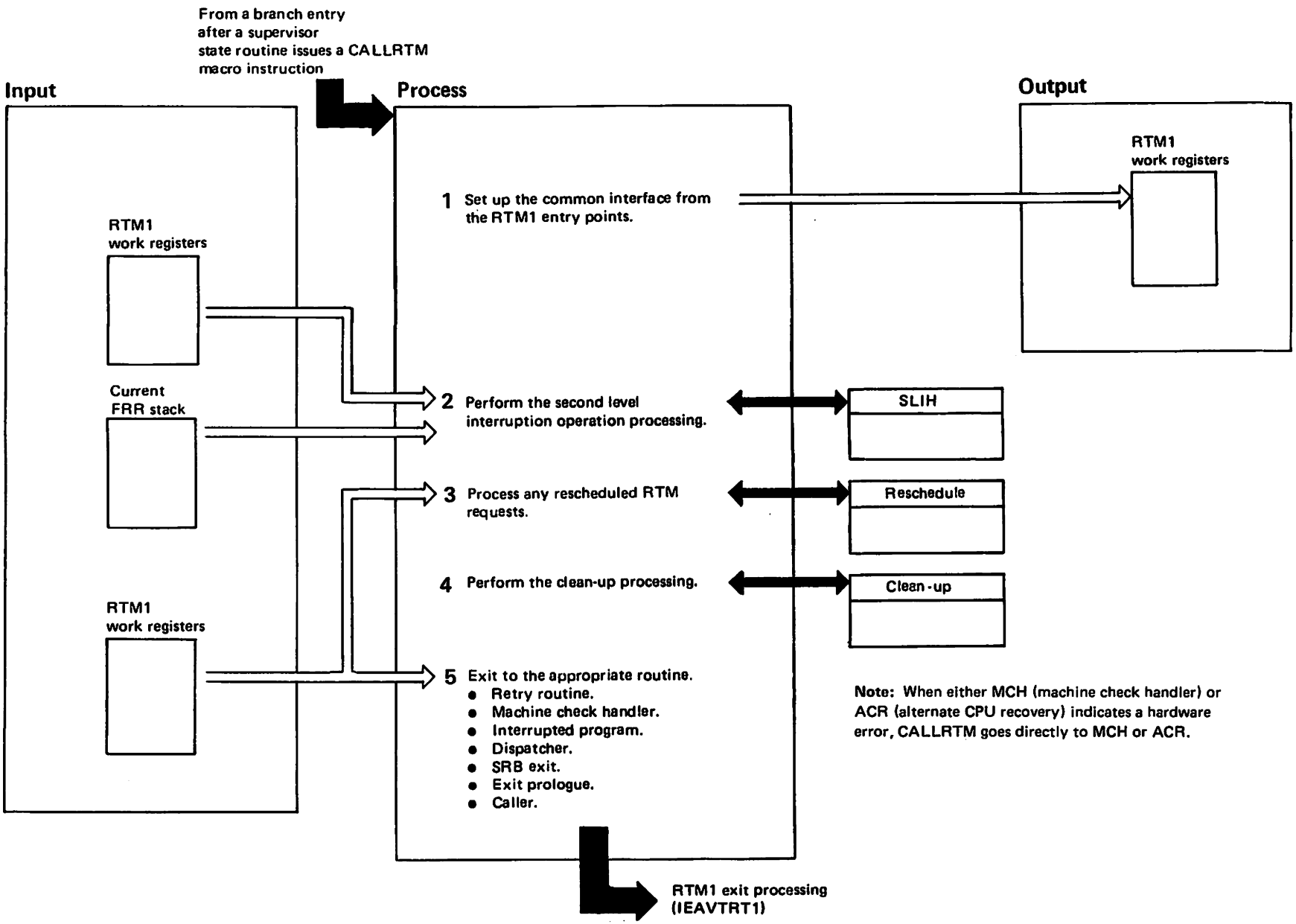


Figure 21 (Part 1 of 2). RTM1 Overview

RTM1 Overview (Part 2 of 2)

Extended Description

The RTM1 service of recovery termination management (RTM) provides a recovery interface with other supervisor routines. When a supervisor routine (principally the interruption handlers) detects an error situation, it passes control to RTM1, via the CALLRTM macro, to initiate recovery from the error. RTM1 records the error (both hardware and software) on SYS1.LOGREC.

RTM1 does not perform the recovery function itself; it routes control to functional recovery routines (FRRs) established by locked, disabled SRB routines or enabled, unlocked task (EUT) routines. These FRRs are placed on a last-in-first-out FRR stack by a SETFRR macro issued by the routine requesting protection. The macro expansion places the FRRs on one of the following stacks, depending on its functional path through the supervisor. (The super FRR is placed on *each* stack by NIP processing.)

- ACR stack
- RTM1 stack
- SVC-I/O-dispatcher stack
- Machine check stack
- Program check stack
- External interruption handler 1 stack
- External interruption handler 2 stack
- External interruption handler 3 stack
- Restart interruption handler stack

Additionally, a normal FRR stack contains the recovery status for other paths through the system.

RTM1 receives control for the following reasons:

- Program checks
- Restart operations
- SVC errors
- STERM errors
- Machine checks
- DAT (dynamic address translation) errors
- Abnormal termination (ABTERM) requests for a task with an ASID (address space identifier) specified
- Abnormal termination requests for a task in the current address space
- Address space termination requests
- Reentry for abnormal termination requests
- Reentry for machine checks
- Branch entries for abnormal termination requests
- RMGRCML resource manager

Extended Description	Module	Label
1 RTM1 creates a common interface for its sub-functions using various entry point data and establishes recursion control for service routine requests.	IEAVTRT1	
2 The program check IH (interruption handler), SVC IH, restart IH, and machine check handler (MCH) all can request that RTM1 perform second level interruption handler (SLIH mode) processing. When RTM1 processes an SLIH mode entry, via a CALL RTM (that is, TYPE = PROGCK, SVCERR, RESTART, DATERR, or MACHCK) it continues processing the interruption only after IEAVTRTV verifies that the PSA pointer (PSACSTK) to the current FRR stack contains a valid FRR stack address. If there is a valid address, then RTM1, while in SLIH mode, determines the state of the system at the time of the interruption so that recovery from the interruption can be attempted in either system mode or task mode. If PSACSTK is not the address of a valid FRR stack, IEAVTRTV invokes IGFPTERM to put the system in a X'084' wait state and to issue message IEA797W, requiring the system to be re-ipled.	IEAVTR10	
3 RTM1 performs reschedule processing for a service routine entry (that is, the CALLRTM request was for ABTERM, MEMTERM, or STERM). The reschedule function can also be performed as part of SLIH mode processing. This would occur if the action indicated by routine to FRRs required a reschedule service or if the processor had been in task mode (no FRRs established) when the error interruption occurred.	IEAVTRTM	
4 The cleanup function frees any resources no longer necessary before determining the appropriate type of exit.		
5 RTM1 creates the final exit linkage based on an indicator established in IEAVTRTM except for FRR retry and resume processing, which are performed by IEAVTR10.	IEAVTRT1 or IEAVTR10	

Figure 21 (Part 2 of 2). RTM1 Overview

RTM2 OVERVIEW

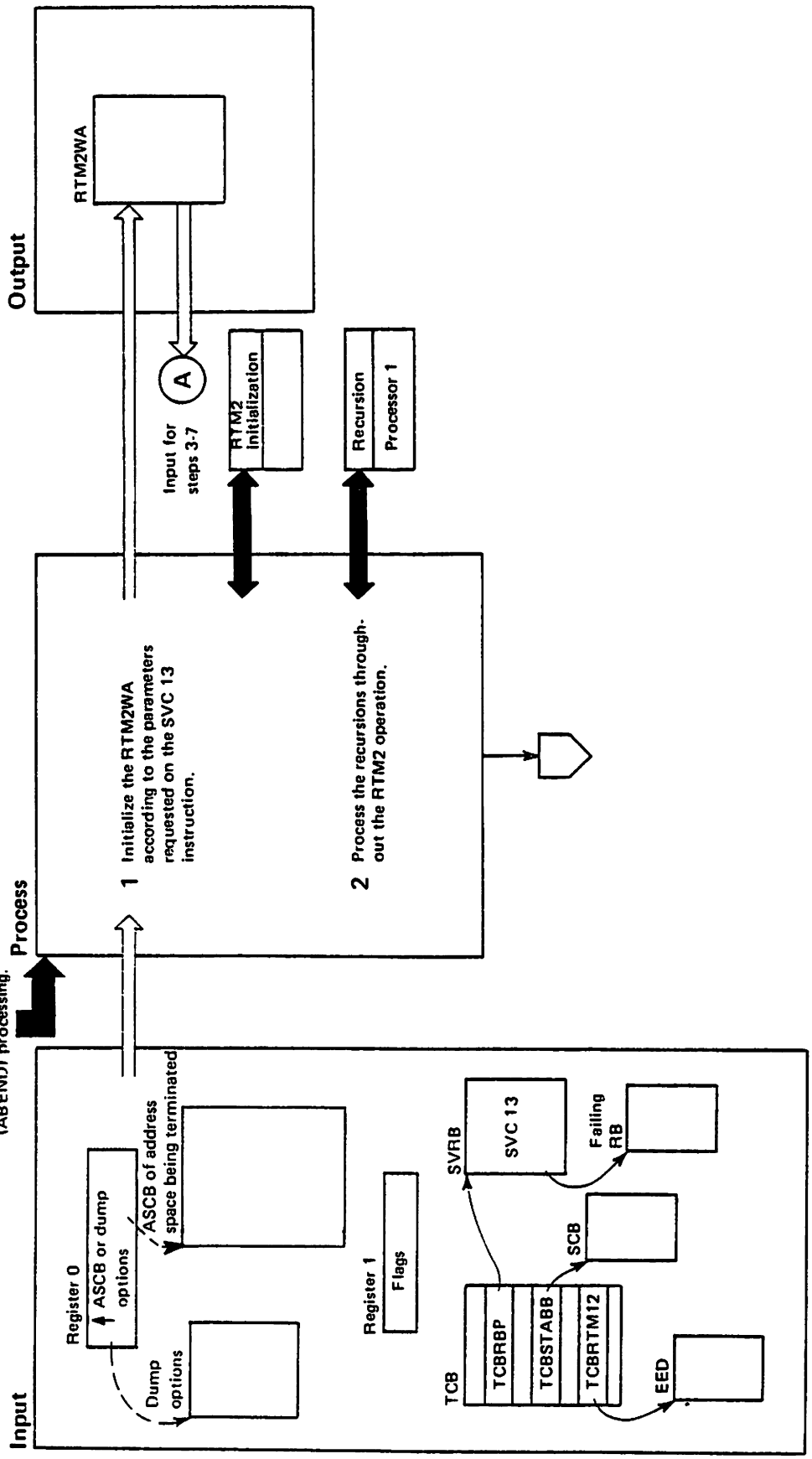


Figure 22 (Part 1 of 4). RTM2 Overview

RTM2 Overview (Part 2 of 4)

Extended Description	Module	Label
The RTM2 function responds to SVC 13 (ABEND) requests after receiving control from the SVC IH (interruption handler). Basically, RTM2: • Initializes a common work area called the RTM2WA. This work area contains the information needed by the various RTM2 routines to service the SVC 13 request; the work area serves as the input for the rest of RTM2 processing. • Provides for error handling in RTM2 by tracking any possible recursions that occur. Unlike other supervisor routines, RTM2 does not rely on FRRs (functional recovery routines) to handle errors. Instead, RTM2 uses recursion tracking to perform recovery by tracking the various RTM2 routines as they execute. • Performs any of the basic RTM2 services: task recovery, storage displays, synchronizing failing tasks, purging task resources, and purging address space resources. • Exits to the correct RTM2 exit routine depending on the following conditions indicated in the RTM2WA: permanent or last task exit, retry, normal EOT (end-of-task) abnormal termination of a task, address space termination, subtask waiting to terminate, convert-to-step request, or recursion exit condition. Control then goes to the dispatcher (IEAVEDS0) or EXIT prolog (IEAVEEXP).		
1 RTM2 initializes an RTM2WA with the information needed to perform the requested service. RTM2 routines use the information placed in the RTM2WA as input. The IEAVTRT2—RTM2 initialization M.O. diagram shows how RTM2 obtains and initializes the RTM2WA.	IEAVTRT2	RT2INWA
2 Recursion processing occurs throughout RTM2 processing. Basically, RTM2 indicates each logical section of code as it executes in the RTM2SCTC field of the RTM2WA. This field shows the sequential processing of segments, and marks how far RTM2 processed any request. The IEAVTRT2—recursion processor 1 M.O. diagram shows this function. After a recursion occurs, RTM2 either retries the segment if the segment can recover from the error, or skips the segment for any further processing requiring that segment. The M.O. diagram IEAVTRTE — Recursion Processor 2 shows this function.		

Figure 22 (Part 2 of 4). RTM2 Overview

RTM2 Overview (Part 3 of 4)

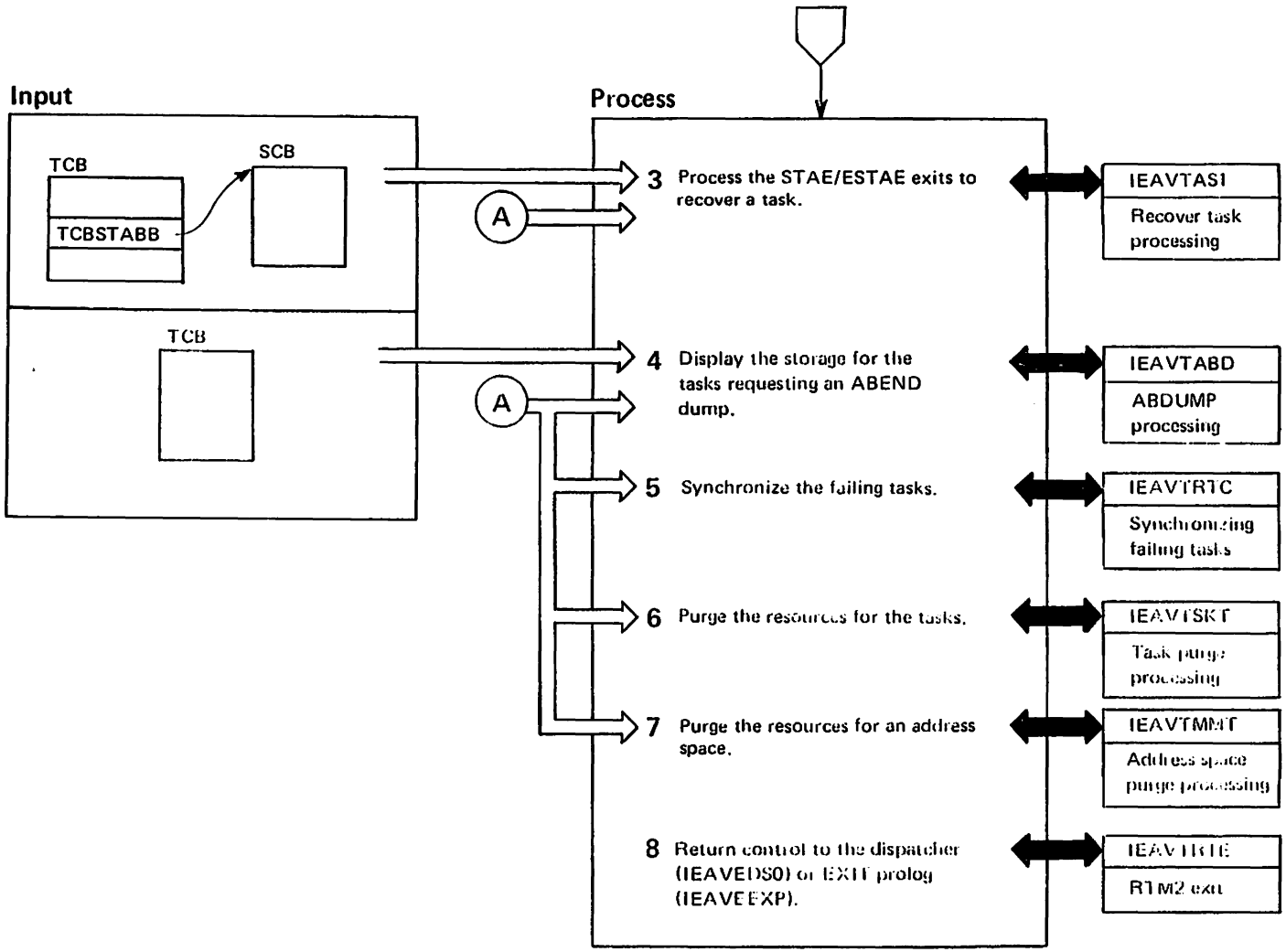


Figure 22 (Part 3 of 4). RTM2 Overview

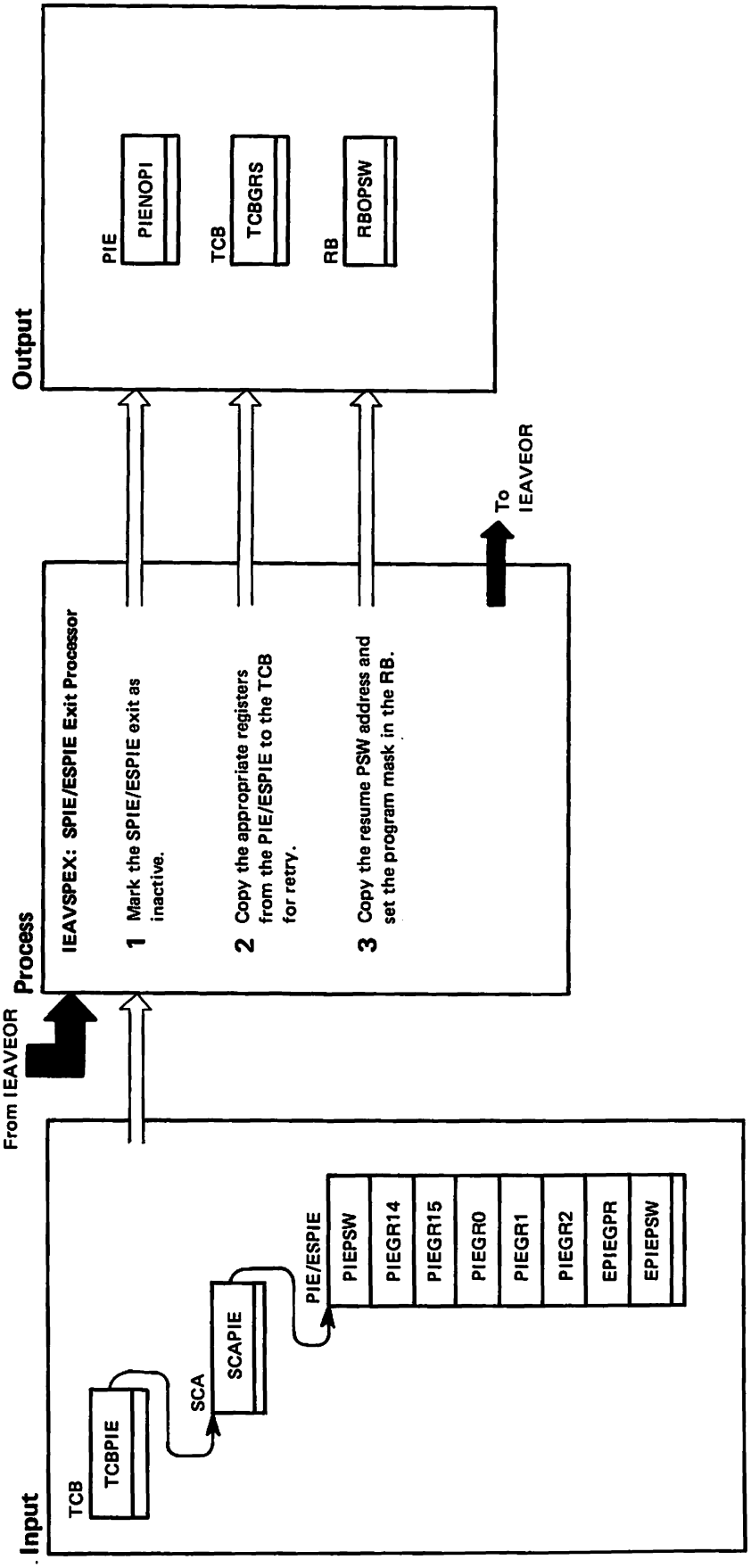
RTM2 Overview (Part 4 of 4)

Extended Description	Module	Label
3 RTM2 will process the STAE/ESTAE exits. The M.O. diagram IEAVTAS1 – Recover Task Processing shows the STAE/ESTAE recovery function, the M.O. diagram IEAVSTA0 – STAE/ESTAE processing shows the creation of the STAE/ESTAE exit and the SCB (STAE control block).	IEAVTRTC IEAVTAS1 IEAVTAS2 IEAVTAS3	
4 RTM2 displays storage when the caller specifies dump. The M.O. diagram IEAVTABD – ABDUMP Processing in the section "Dumping Services" shows the processing involved to dump selected areas of main storage.	IEAVTABD	
5 Failing tasks will complete their termination even if they are subtasks of a task that fails during their termination processing. RTM2 synchronizes failing tasks to independently terminate all the tasks in a TCB family that fail. The M.O. diagram IEAVTRTC – Synchronizing Failing Task shows this processing.	IEAVTRTC IEAVTRTE	
6 RTM2 routes control to the resource manager routines to perform necessary clean up for task termination. The M.O. diagram IEAVTSKT – Task Purge Processing shows this processing.	IEAVTRTE IEAVTSKT	
7 RTM2 purges address space resources for address space termination requests. The M.O. diagram IEAVTMMT – Address Space Termination Processing shows this processing.	IEAVTRTE IEAVTMMT	
8 Exit processing for RTM2 consists of returning control to the dispatcher (IEAVEDS0) or EXIT prolog (IEAVEEXP). The settings in the RTM2FLX field of the RTM2WA indicate the exit conditions that RTM2 processes. The M.O. diagram IEAVTRTE – RTM2 Exit Processing shows this processing.	IEAVTRTE IEAVTRT2	

Figure 22 (Part 4 of 4). RTM2 Overview

This page left blank intentionally.

IEAVESPI - SPIE/ESPIE PROCESSING



IEAVESPI – SPIE/ESPIE Processing (Part 1 of 6)

IEAVESPI – SPIE/ESPIE Processing (Part 2 of 6)

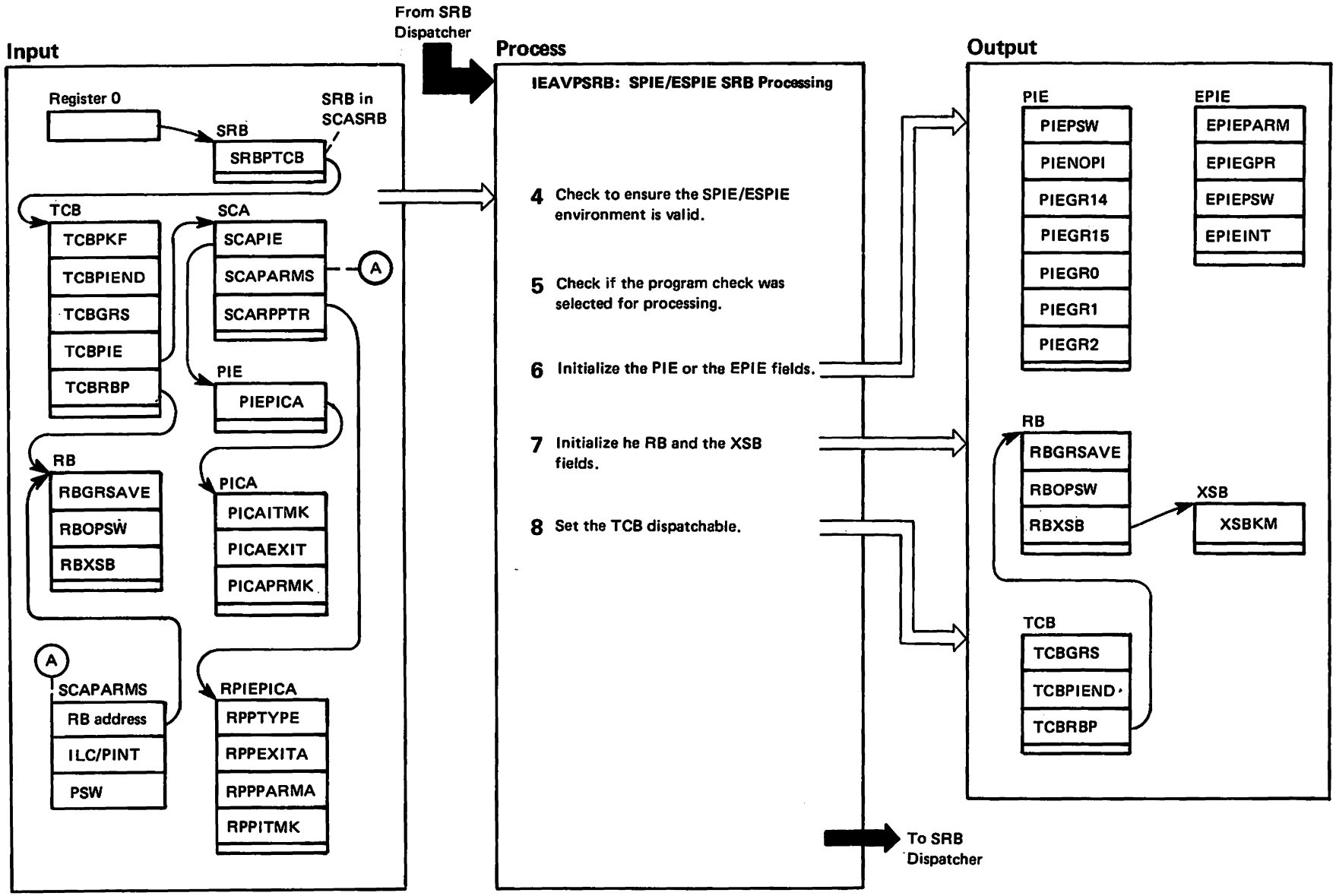
Extended Description	Module	Label
SVC exit processing, IEAVEOR, calls IEAVSPEX, an entry point in IEAVESPI, whenever a program issues the SPIE/ESPIE SVC 3 contained in IEAVTESG. IEAVSPEX determines if a SPIE or ESPIE exit issued the SVC 3. If a SPIE or ESPIE exit did not issue the SVC 3, IEAVSPEX sets a return code of 0. Otherwise, IEAVSPEX sets the TCB registers for retry, sets the RB resume PSW address and mask, sets a return code of 4, and returns control to IEAVEOR.	IEAVESPI	IEAVSPEX

1 IEAVSPEX resets the program interrupt element flag bit (PIENOPI) to indicate that neither a SPIE nor an ESPIE exit is currently in control and that it is valid to schedule the exit if an error occurs.

2 If a SPIE exit issued the SPIE/ESPIE SVC 3, IEAVSPEX copies registers 0-2 and 14-15 from the program interrupt element control block (PIE) to the TCB. If an ESPIE exit issued the SPIE/ESPIE SVC 3, IEAVSPEX copies all the registers from the extended program interrupt element control block (EPIE) to the TCB.

3 IEAVSPEX then copies the address portion of the PIE or EPIE PSW to the RB old PSW. This becomes the retry address. IEAVSPEX sets the program mask in the RB old PSW from the interrupts specified on the SPIE or ESPIE to complete processing of a SPIE or ESPIE issued from within the returning exit.

IEAVESPI – SPIE/ESPIE Processing (Part 3 of 6)



IEAVESPI – SPIE/ESPIE Processing (Part 4 of 6)

Extended Description

The program check interruption handler (IEAVSPPF) schedules IEAVPSRB, an entry point in IEAVESPI, when a program interruption occurs and a SPIE or ESPIE is eligible to receive control. IEAVPSRB is an SRB that initializes the PIE or EPIE with the PSW and register contents at the time of the program check. IEAVPSRB then modifies the TCB and RB so that the SPIE or ESPIE exit receives control when the TCB is redispatched.

4 IEAVPSRB checks to ensure that the environment is valid to schedule the SPIE or ESPIE exit. If the TCB is dispatchable, IEAVPSRB returns without causing the SPIE or ESPIE exit to receive control. If there are no SPIE or ESPIE exits to process or if a SPIE or ESPIE exit is currently in control (a SPIE or ESPIE exit suffered the program interruption), IEAVPSRB terminates the task by using the CALLRTM interface; the SPIE or ESPIE exit does not receive control.

5 IEAVPSRB checks the PICA (for a SPIE) or the RPIEPICA (for an ESPIE) to determine if the program check was one selected for processing by the issuer of SPIE or ESPIE. If the program check interrupt is not to be processed by the SPIE or ESPIE, IEAVPSRB terminates the task by using the CALLRTM interface; the SPIE or ESPIE exit does not receive control.

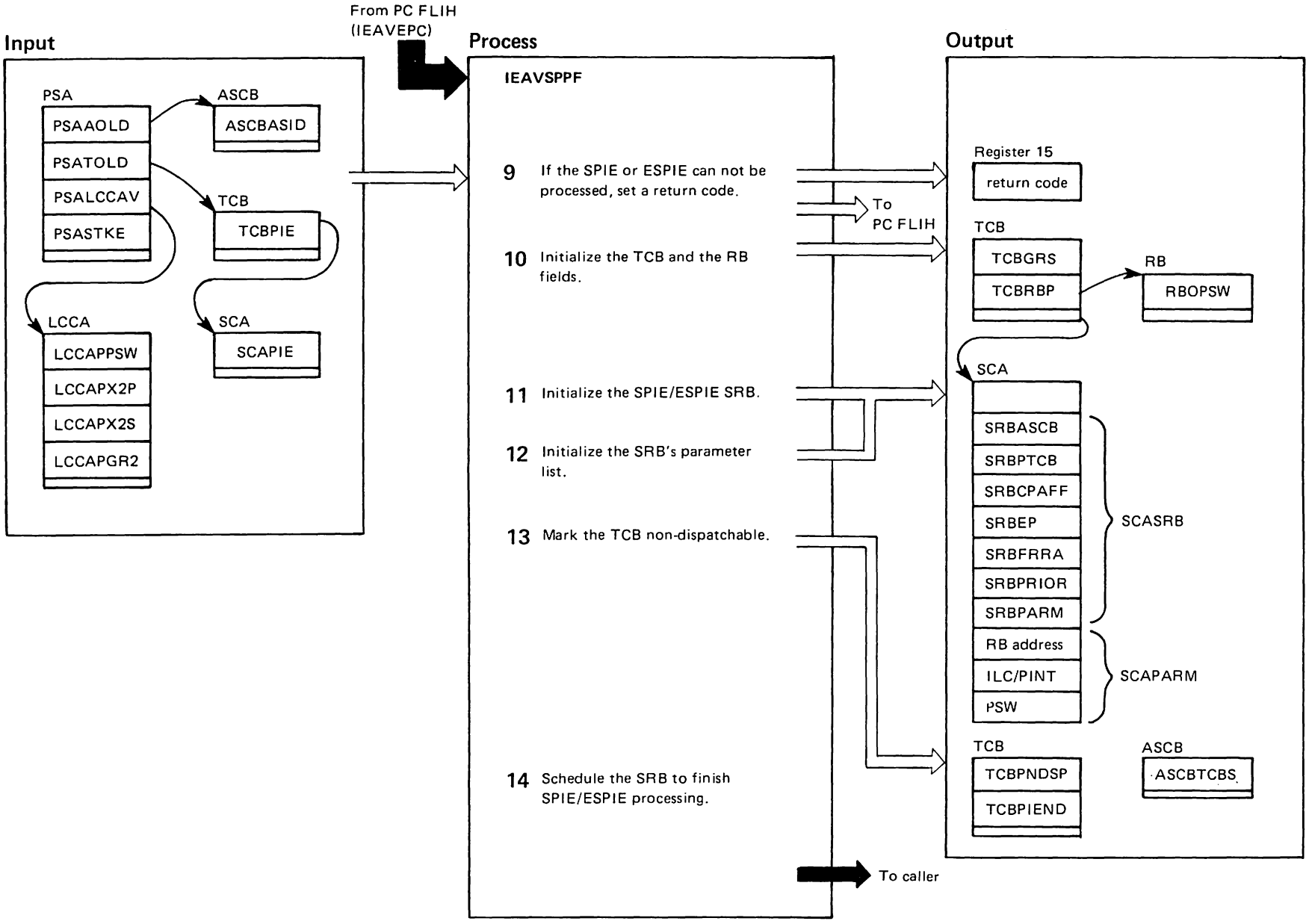
6 IEAVPSRB initializes either the PIE or ESPIE with the PSW and the register contents at the time of error.

7 IEAVPSRB initializes the RB resume PSW so that the exit receives the registers saved in the TCB or RB so that the exit receives control with the appropriate register interface. IEAVPSRB uses the TCB key to set the PKM in the extended status block (XSB).

8 IEAVPSRB invokes the STATUS service (to mark the TCB dispatchable).

Module	Label
IEAVESPI	IEAVPSRB

IEAVESPI – SPIE/ESPIE Processing (Part 5 of 6)



IEAVESPI – SPIE/ESPIE Processing (Part 6 of 6)

Extended Description

Module	Label
IEAVESPI	IEAVSPPF

The PC FLIH (IEAVEPC) calls entry point IEAVSPPF for all program interruptions except PER, monitor event, special operation, and translation exception. If the system environment at the time of the interrupt does not preclude a SPIE or ESPIE from being given control, IEAVSPPF schedules an SRB to complete the processing necessary to give the exit control. When IEAVSPPF returns to the PC FLIH, register 15 contains a code that indicates whether to continue SPIE or ESPIE processing or to abnormally terminate the task.

9 IEAVSPPF tests the execution environment to determine if the SPIE or ESPIE SRB should be scheduled. If the SRB should not be scheduled, IEAVSPPF sets a return code of 8 in register 15 and returns control to PC FLIH to abnormally terminate the task.

10 IEAVESPPF copies the registers at the time of the interrupt to the TCB and copies the PSW at the time of the interrupt to the RB.

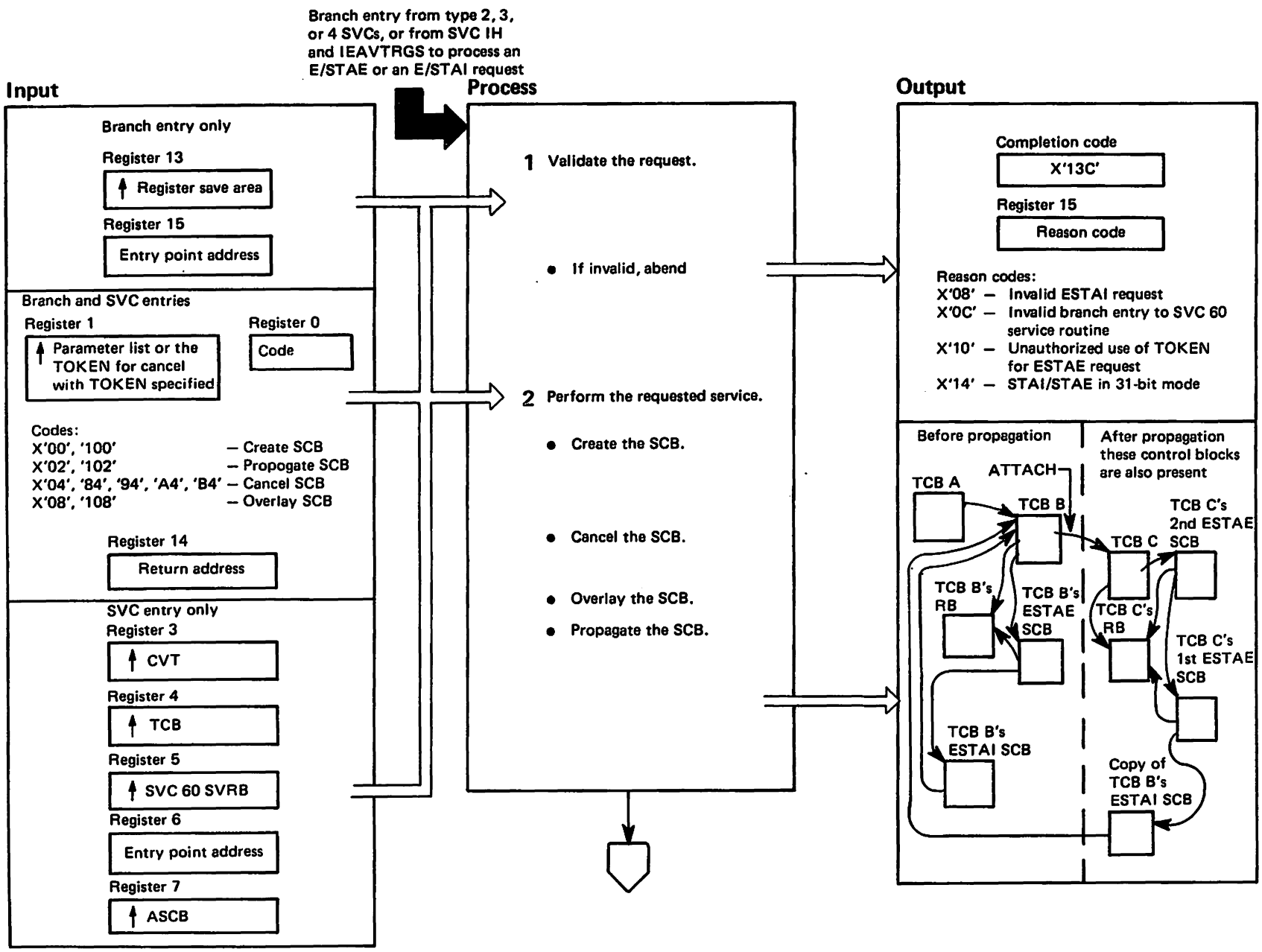
11 IEAVSPPF initializes the SPIE/ESPIE SRB contained in the SCA. The SPIE/ESPIE SRB processor routine contained in IEAVESPI (IEAVESRB) receives control when the SRB runs.

12 IEAVSPPF initializes the SRB parameter list contained in the SCA.

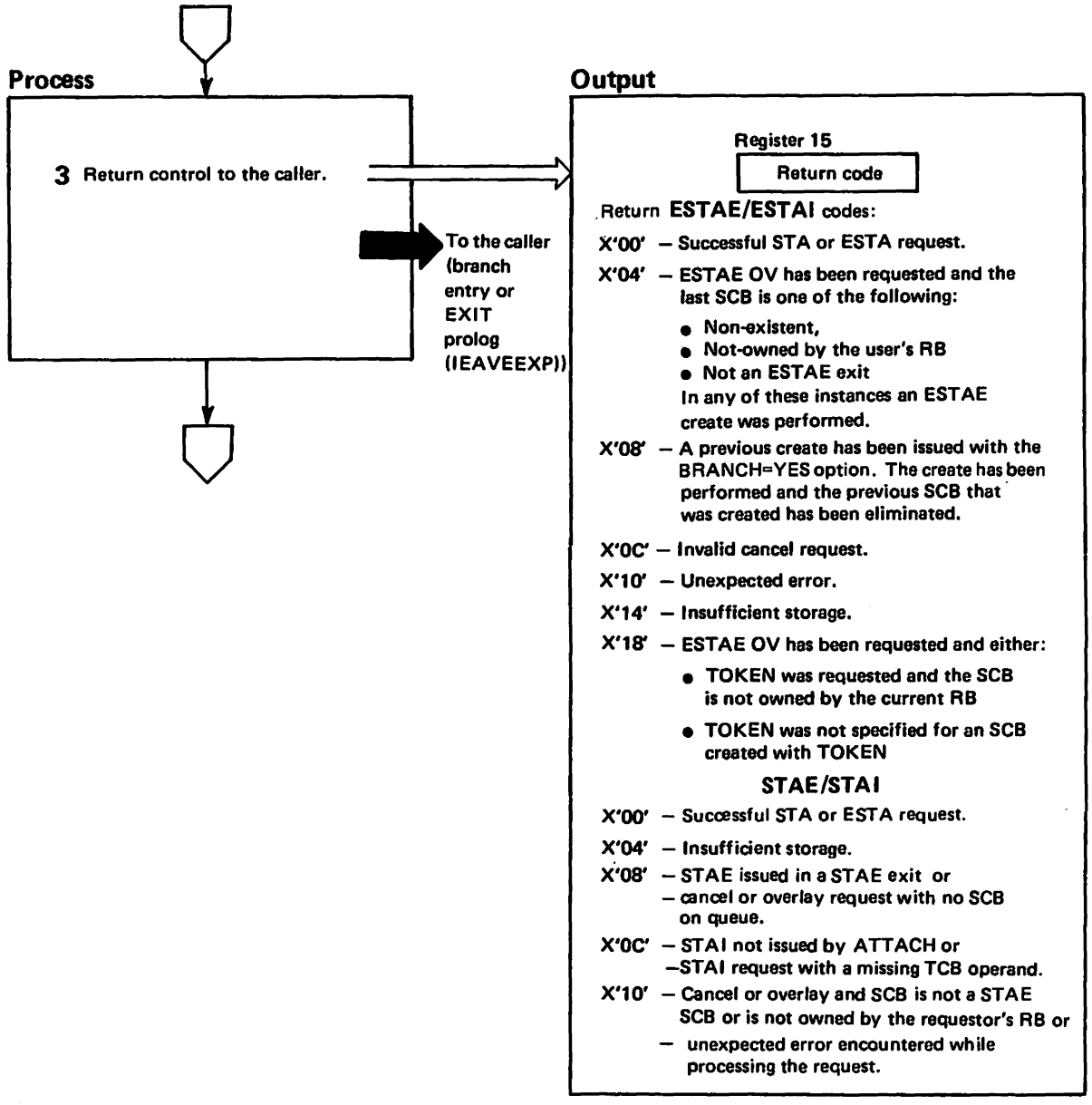
13 IEAVSPPF makes the TCB non-dispatchable by setting the TCBPIEND and TCBPNDSP bits on in the TCB and decreases the number of ready TCBs in the ASCB (ASCBTCBS).

14 IEAVSPPF schedules the SPIE/ESPIE SRB to complete processing for the exit.

IEAVSTA0 – STAE/ESTAE Service Routine (Part 1 of 6)



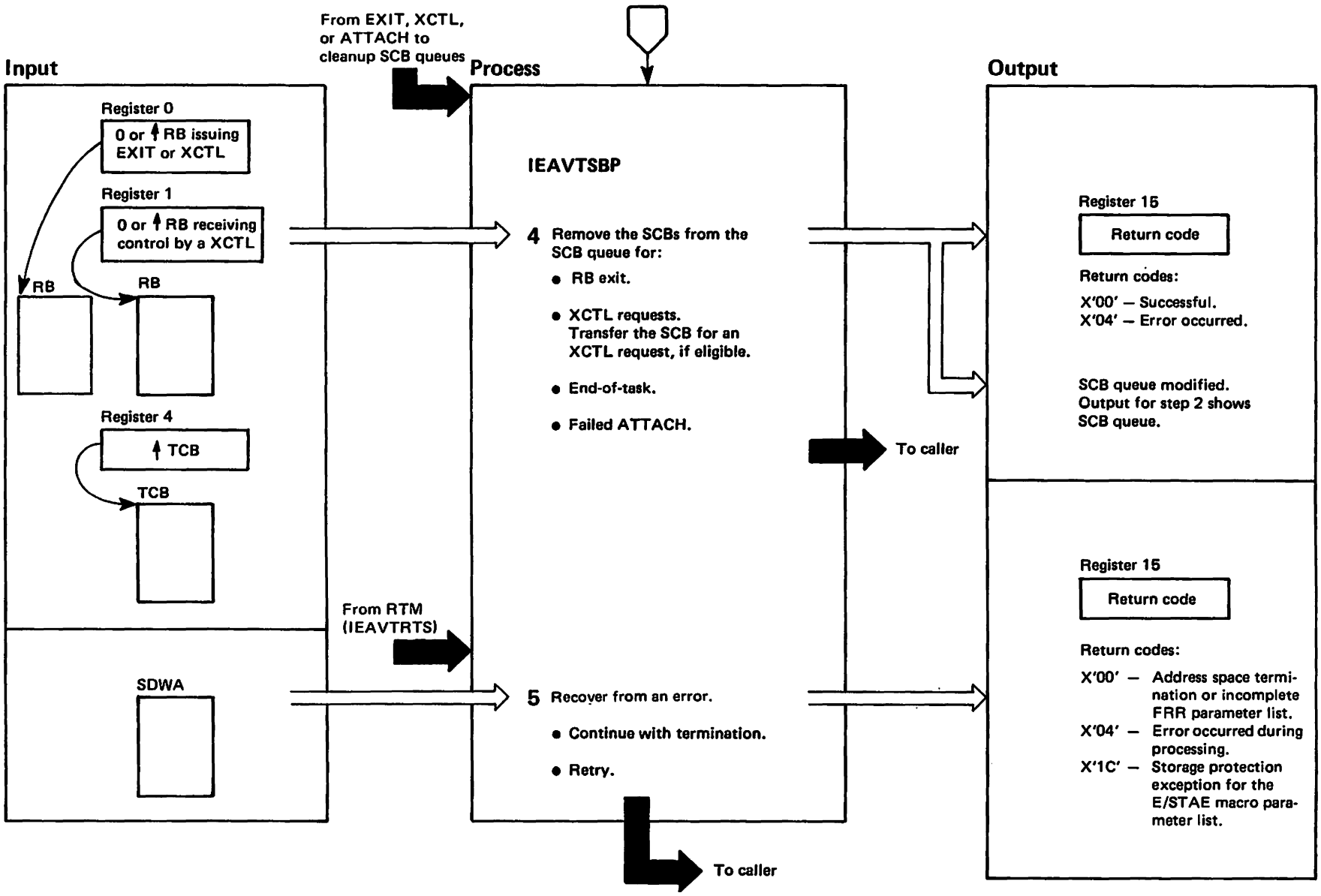
Extended Description	Module	Label
The STAE/ESTAE service routine creates and initializes an SCB (STAE/ESTAE control block) to represent an abnormal interruption exit routine. The STAE/ESTAE service routine can create, cancel, overlay, or propagate an SCB, according to the action codes passed as input. An ESTAE SCB can have an unique identifier (TOKEN) associated with it. If an ESTAE SCB is created with a TOKEN, then this TOKEN is used to locate the SCB to cancel or overlay it. The STAE/ESTAE service routine receives control from the SVC interrupt handler or from type 2, 3, or 4 SVCs by branch entering module IEAVTRGS, which preserves the addressing mode and return address of its caller and then branches to IEAVSTA0 in 31-bit mode. Control returns to the caller.		
1 The STAE/ESTAE service routine validates both branch-entered and SVC issued requests. The STAE/ESTAE service routine abnormally terminates invalid callers, issuing a X'13C' abend. The value in register 15 indicates the reason for the termination.	IEAVSTA0	
2 The STAE/ESTAE service routine performs the requested service, as indicated in register 0.		
(a) For create requests, the STAE/ESTAE service routine obtains storage for an (E) STAI or (E) STAE SCB from the SCB cellpool (if available) or by issuing a GETMAIN request. The STAE/ESTAE service routine chains each newly created SCB to the SCB queue, pointed to by the appropriate TCB. The STAE/ESTAE service routine indicates that the caller owns an SCB by setting an indicator in the RBSCB field of the caller's RB. When an SCB associated with an ESTAE is created with TOKEN=token-address, after STAE/ESTAE processing completes, the user-supplied token-address field contains the token created for this request. When processing a STAI or ESTAI request, the STAE/ESTAE service routine automatically propagates the STAI or ESTAI SCBs from all former (E) STAI requests. (see (d))		
(b) For cancel requests, the STAE/ESTAE service routine dequeues the last SCB related to the caller's RB. For an ESTAE with TOKEN=token-address specified, the STAE/ESTAE service routine uses TOKEN to find and dequeue the SCB identified by TOKEN and all newer SCBs associated with the caller's RB. If the caller does not own any more SCBs, the RBSCB indicator in the caller's RB is set to zero.		
(c) For overlay requests, the STAE/ESTAE service routine initializes the existing SCB with the new values. If the original SCB was created with a TOKEN, then the STAE/ESTAE service routine uses TOKEN to find this specific SCB and replaces the old exit information with the new ESTAE exit routine information. In addition, if TOKEN was specified, the STAE/ESTAE service routine deletes all newer SCBs associated with the caller's RB.		
(d) For (E) STAI propagation, the STAE/ESTAE service routine obtains storage for the other SCB(s) from the SCB cellpool (if available) or by issuing a GETMAIN, copies the SCB information from the appropriate SCB(s) (addressed by the TCB pointed to in register 4), and chains both the new and the propagated SCB(s) to the newly attached TCB.		



IEAVSTA0 – STAE/ESTAE Service Routine (Part 4 of 6)

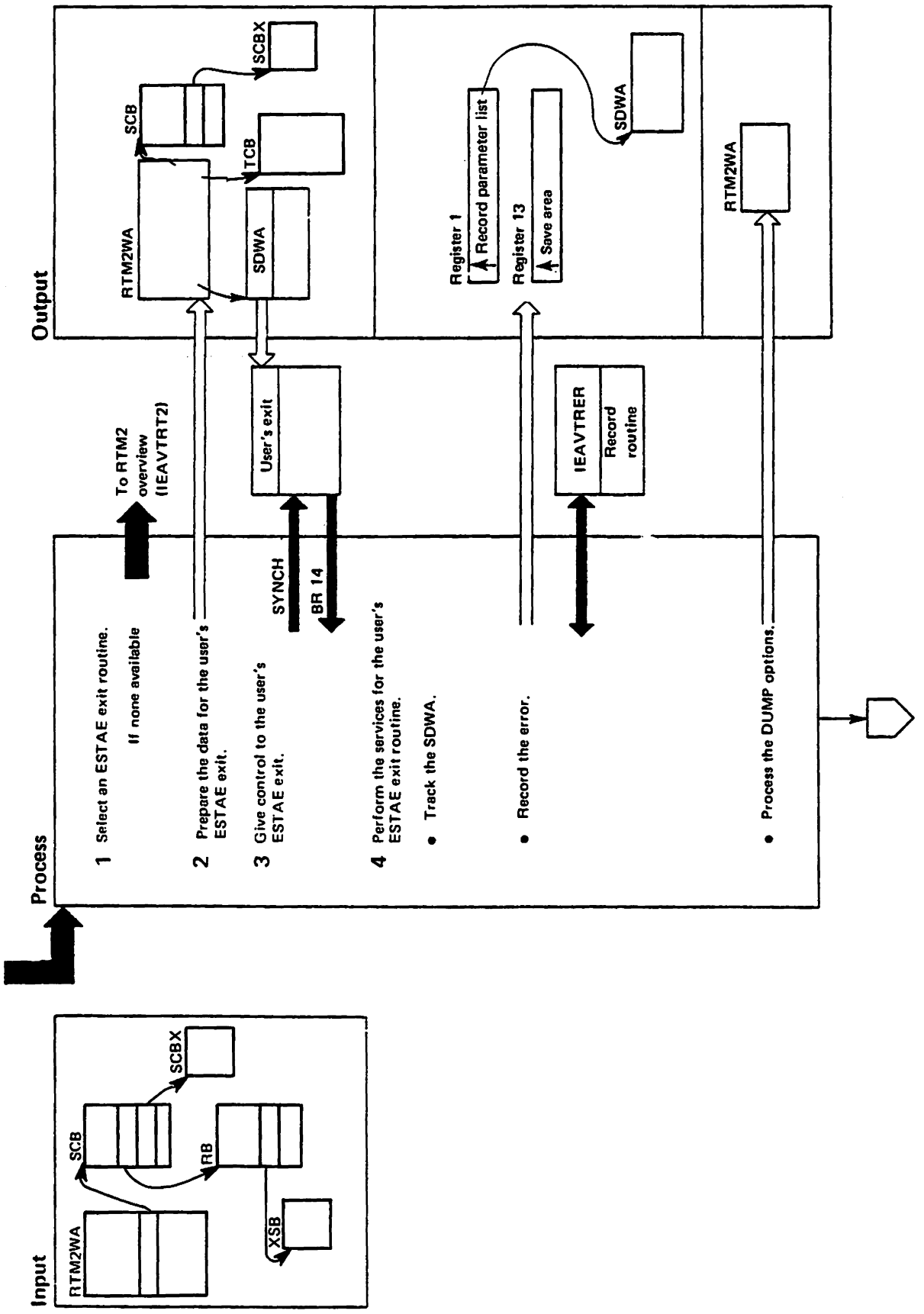
Extended Description	Module	Label
3 The STAE/ESTAE service routine returns control to the caller, with a return code in register 15 indicating the results of the request.		

IEAVSTA0 – STAE/ESTAE Service Routine (Part 5 of 6)



Extended Description	Module	Label
EXIT, XCTL and ATTACH use the SCB task recovery resource manager (IEAVTSBP) to transfer or dequeue SCBs. IEAVTSBP builds an FRR to recover from errors. Upon completion, control returns to the caller.		
4 IEAVTSBP either removes or transfers SCBs. If IEAVTSBP finds a 0 in register 4, it returns the caller a return code of 4 in register 15. IEAVTSBP sets the calling RB's SCB indicator to show that no SCB is owned. In addition, depending on the caller, IEAVTSBP does one of the following:	IEAVTSBP	
• For an RB issuing EXIT, removes the SCBs from the SCB queue.		
• For an RB issuing XCTL, transfers all SCBs created with the XCTL=YES option and removes all SCBs if the XCTL=YES option was not specified. IEAVTSBP sets the new RB's SCB indicator to show that at least one SCB is owned.		
• For an RB issuing end-of-task EXIT or if an ATTACH request failed, dequeues the SCBs from the SCB queue and sets the SCB address field in the TCB to 0 to indicate that no SCBs are on the active queue.		
5 If an error occurred in IEAVTSBP, its functional recovery routine (FRR) attempts to recover.	IEAVTSBP	TRRMFRR
• If the error occurred under address space switch conditions or if the FRR parameter list was incomplete, no retry is permitted and the FRR finishes termination.		
• If the caller requested dequeueing of all SCBs associated with this task, the FRR makes the SCB queue pointer in the TCB equal zero.		
• For RB EXIT and an XCTL request, the FRR first checks for storage key failures and storage data checks. If either is found, the FRR scans the SCB queue for an SCB within the address range of the storage error indicated in the SDWA. If the FRR finds an SCB within this range, the FRR makes the SCB queue pointer in the TCB equal zero. If the FRR finds no SCB within this address range or if there was no storage error, the FRR dequeues all the SCBs owned by this RB.		

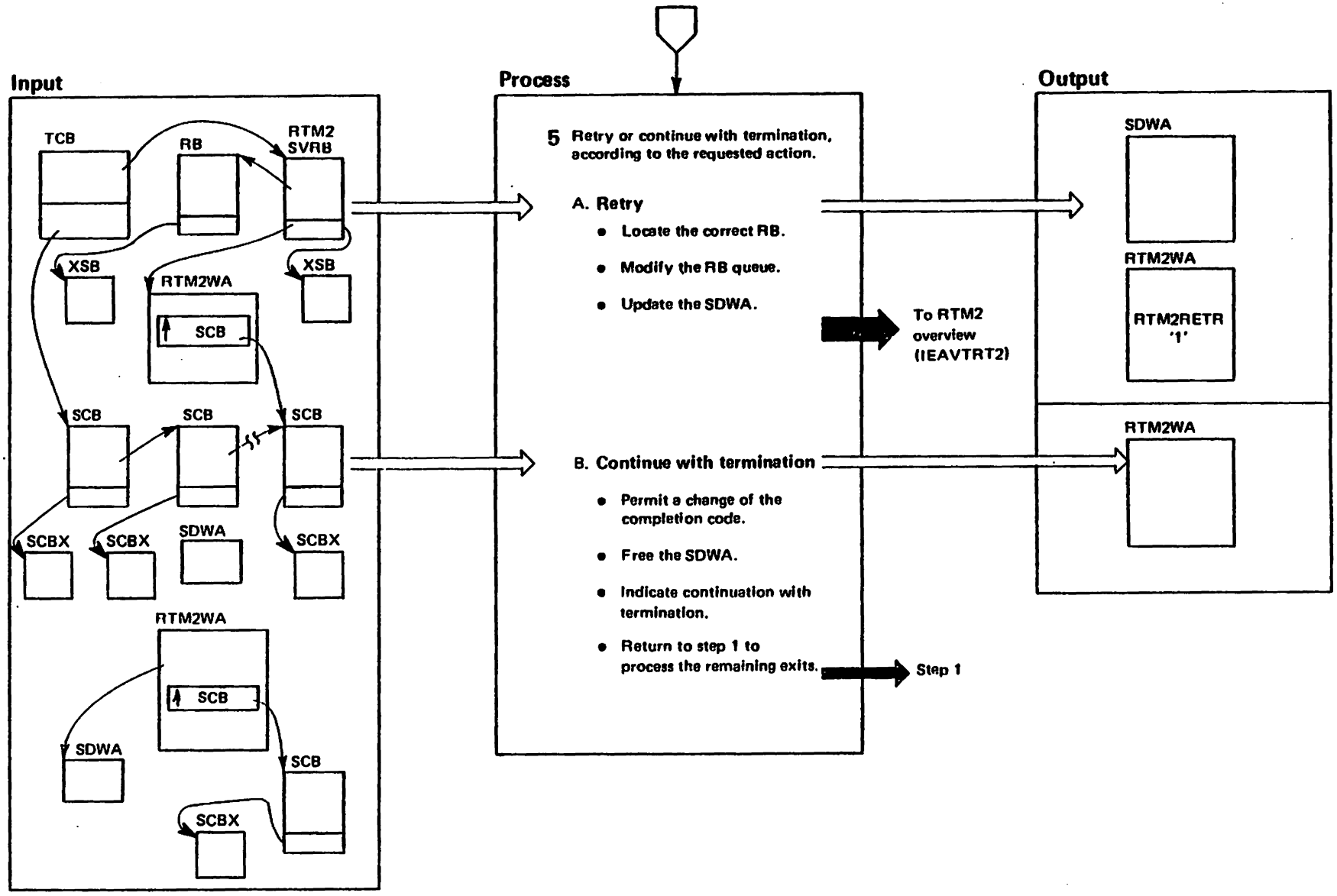
IEAVTAS1 - RECOVER TASK PROCESSING



Extended Description	Module	Label
RTM2 routes control to user-written exit routines before it terminates a task. These exit routines -- either STAE (specify task abnormal exit) or ESTAE (extended STAE) – receive control to attempt to recover an abnormally terminated task. (See the M.O. diagram IEAVSTA0 – STAE/ESTAE Service Routine for a description of how the user creates a STAE control block (SCB)). See *Supervisor Services and Macro Instructions* for a description of how a user creates an ESTAE routine.)		
RTM2 selects an ESTAE/STAE routine from the SCB queue, and branches to it. If the terminating task can recover after the ESTAE/STAE routine processes, RTM2 performs any processing necessary for a retry condition, and the task resumes processing. Otherwise, the task is terminated.		
RTM2 places diagnostic information in the SDWA during ESTAE/STAE processing.		
1 RTM2 searches the SCB queue to select the exit to be given control. The searching sequence follows:	IEAVTAS1	FINDSCB
• On initial entry, the most recently established exit will be selected.		
• During percolation, (a previously selected exit has not elected to retry) – the next exit on the queue is selected.		
• During percolation only one STAE (as opposed to ESTAE) is selected; all others are bypassed.		
• During TERM processing, only those exits with the TERM option (TERM=YES on the ESTAE macro instruction) are selected.		
• If the queue is exhausted with no exit requesting retry, control returns to RTM2 and the task is terminated.		
Note: If more than 32 consecutive exit routines fail (because of a program check or issuance of the ABEND macro) the next exit in the chain will be skipped. IEAVTAS1 changes the completion code to X'60D' with a reason code of 0.		

Extended Description	Module	Label
2 RTM2 initializes some fields in the internal RTM2WA (RTM2 work area) to ensure the accuracy of the SDWA during percolation.		WKUPDAT
RTM2 obtains and initializes an SDWA with information that will aid the user in diagnosing the error.		SDWAINIT
RTM2 performs the user otpions indicated on the ESTAE macro instruction. Abnormal exit processing may be blocked and active I/O may be halted or quiesced. I/O options are performed only for the first exit selected; all subsequent exits receive an indication of I/O status.		USEROPTS
3 RTM2 initializes parameter registers for the exit routine. Additionally, RTM2 sets the interface with the SYNCH macro (used to give control to the exit).		EXITINTR
4 On return from the user exit routine, RTM2 uses the macro to trace either the SDWA (if one exists) or the return information. RTM2 writes the SDWA on the SYS1.LOGREC data set if one of the following is true:	IEAVTAS2	GTFHOOK RCRDSDWA
• SLIP requested recording.		
• The user exit requested the SDWA be recorded, and it is available.		
• RTM2 is processing a restart error that occurred while in the enabled, unlocked task mode.		
RTM2 initializes the RTM2 work area with user dump options if any exist. RTM2 combines any dump parameters with existing options; it adds storage ranges to the end of the existing storage range list, wrapping around to the top again if necessary. (A maximum of thirty storage ranges can be accumulated.) If the user requested no dump, RTM2 sets the existing options to zero.		DUMPORTS

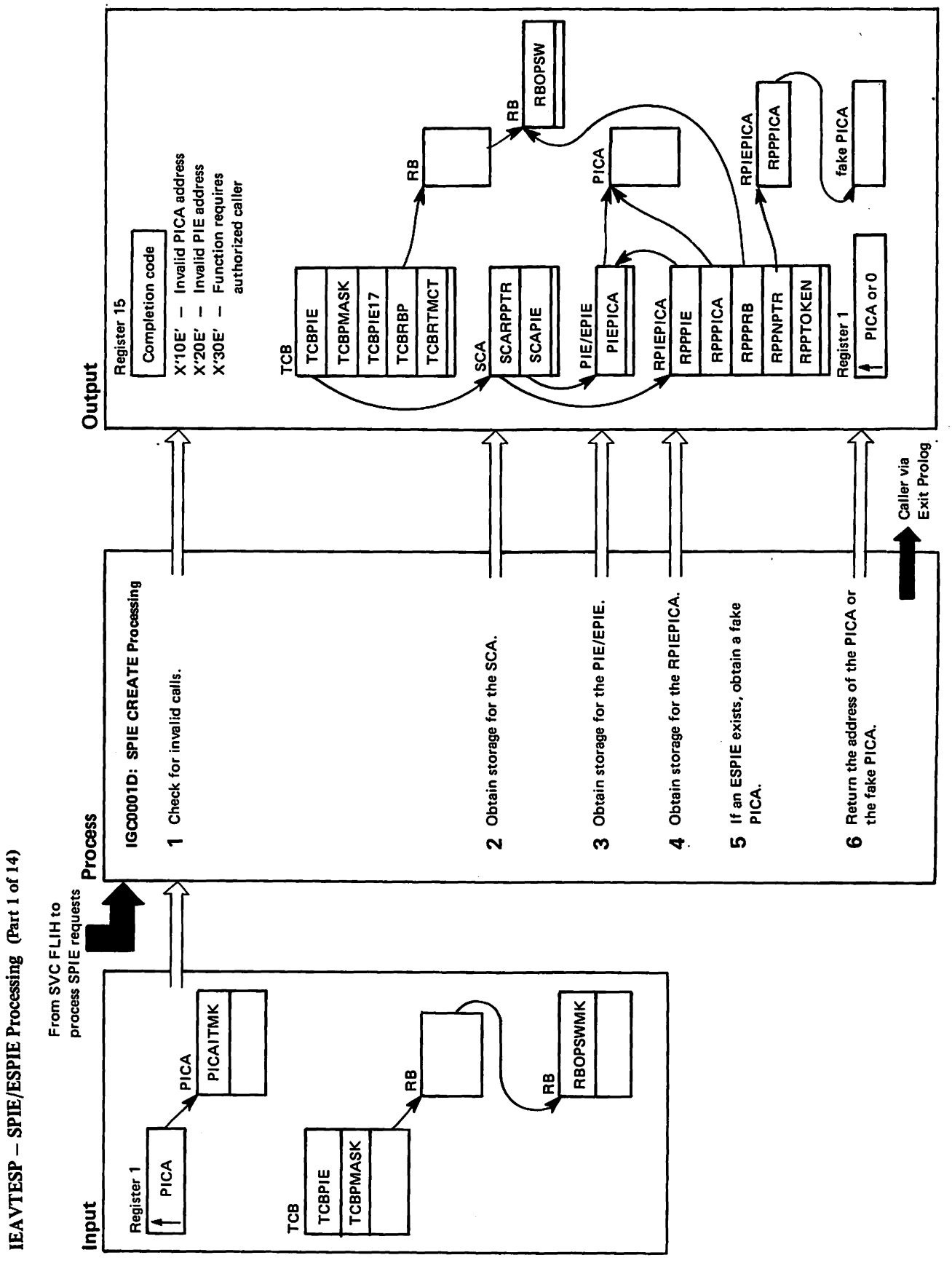
IEAVTAS1 – Recover Task Processing (Part 3 of 4)



IEAVTAS1 – Recover Task Processing (Part 4 of 4)

Extended Description	Module	Label
5A If a retry can be performed (this is not term exit processing), RTM2 selects a retry RB. For a STAE/ESTAE retry, the SCB contains the RB address. For an ESTAR retry, RTM2 uses the oldest RB. For a STAI/ESTAI, RTM2 performs the retry under the PRB for the last STAE/ESTAE or STAI/ESTAI exit routine if one exists. Otherwise, RTM2 purges the RB queue until only PRBs remain and the STAI/ESTAI retry routine will run under the newest PRB left on the queue.	IEAVTAS3	FINDRB
RTM2 prepares the RB queue for a retry. RTM2 purges resources and closes open, embedded data sets. RTM2 sets the primary and secondary address spaces of RBs to be purged (those between the retry RB and the ABEND SVRB) to the home address space, turns off their PSW S-bits, points their resume PSW to EXIT, and sets their wait count to zero. If registers update was requested on the retry, RTM2 inserts the retry register values to ensure that the correct registers are passed to the retrying RB. If register update was not requested, RTM2 initializes the parameter registers to be passed to the retry RB. To ensure that the retry RB will run in the home address space, RTM2 sets the primary and secondary address spaces to the home address space in the XSB, and turns off the RBOPSW S-bit. RTM2 places the keymask for a retry in the retry RB's XSB. The registers and PSW at the entry to ABEND can still be found in the RTM2WA. This work area resides in the LSQA and the TCBRTWA field of the TCB points to it.		RBPRGE
According to the user's request, RTM2 either updates the SDWA to be passed to the retry routine, or frees it. Task recovery returns control to RTM for further preparation for retry.		RTRYSDWA
5B RTM2 saves the information to be passed to the next exit during percolation (a changed completion code or a serviceability indicator) in the RTM2WA and frees the SDWA. In addition, RTM2 initializes percolation information in the RTM2WA.	IEAVTAS3	SCBPERC

IEAVTESP - SPIE/ESPIE PROCESSING



IEAVTESP – SPIE/ESPIE Processing (Part 2 of 14)

Extended Description | **Module** | **Label**

SPIE processing contains the following entry points:

- IGC0001D – SPIE create processing
- IGX00028 – ESPIE SET, RESET, and TEST processing
- IEAVSPIE – SPIE/ESPIE termination resource manager
- IEAVSPI – SPIE/ESPIE checkpoint/restart processing

SPIE processing handles user requests for program interruption exit routines. When the user codes a SPIE macro instruction, SPIE create processing initializes the fields of a PICA (program interruption control area) with a program mask, the address of a user's program interruption exit routine, and an interruption mask.

If a program check interruption occurs while a program is executing on behalf of the user's task, the user's program interruption exit routine must handle it using information from the PICA. Otherwise, the program whose error caused the program interruption is abnormally terminated. The user's exit routine also uses information from the PIE (program interruption element).

SPIE processing places in the TCB of the macro-issuing program an indirect pointer to the user's exit routine. If a program interruption occurs, the SPIE SRB processor (M.O. diagram IEAVESPI-SPIE/ESPIE SRB Processor) sets up the PIE, checks the TCB indirect pointer field, and passes control to the user's exit routine.

SPIE must refer to the PIE and PICA in the key of the caller, so that unauthorized references to the PIE or PICA will result in a program check. The SPIE FRR (functional recovery routine) converts the program check to either a X'10E' or X'20E' ABEND code.

Extended Description | **Module** | **Label**

1 If the caller is in supervisor state, the caller's key is other than that indicated in the TCBPKF field, or the caller is executing in 31-bit addressing mode, he cannot use SPIE. If PICAEXT does not equal 0, SPIE processing sets the TCBPIE17 bit to 1, if the user is authorized, to indicate page fault processing is requested.

2 If the TCBPIE field equals 0, this is the first time that the caller has issued a SPIE macro. SPIE create processing obtains and initializes the SCA and chains it to the TCB.

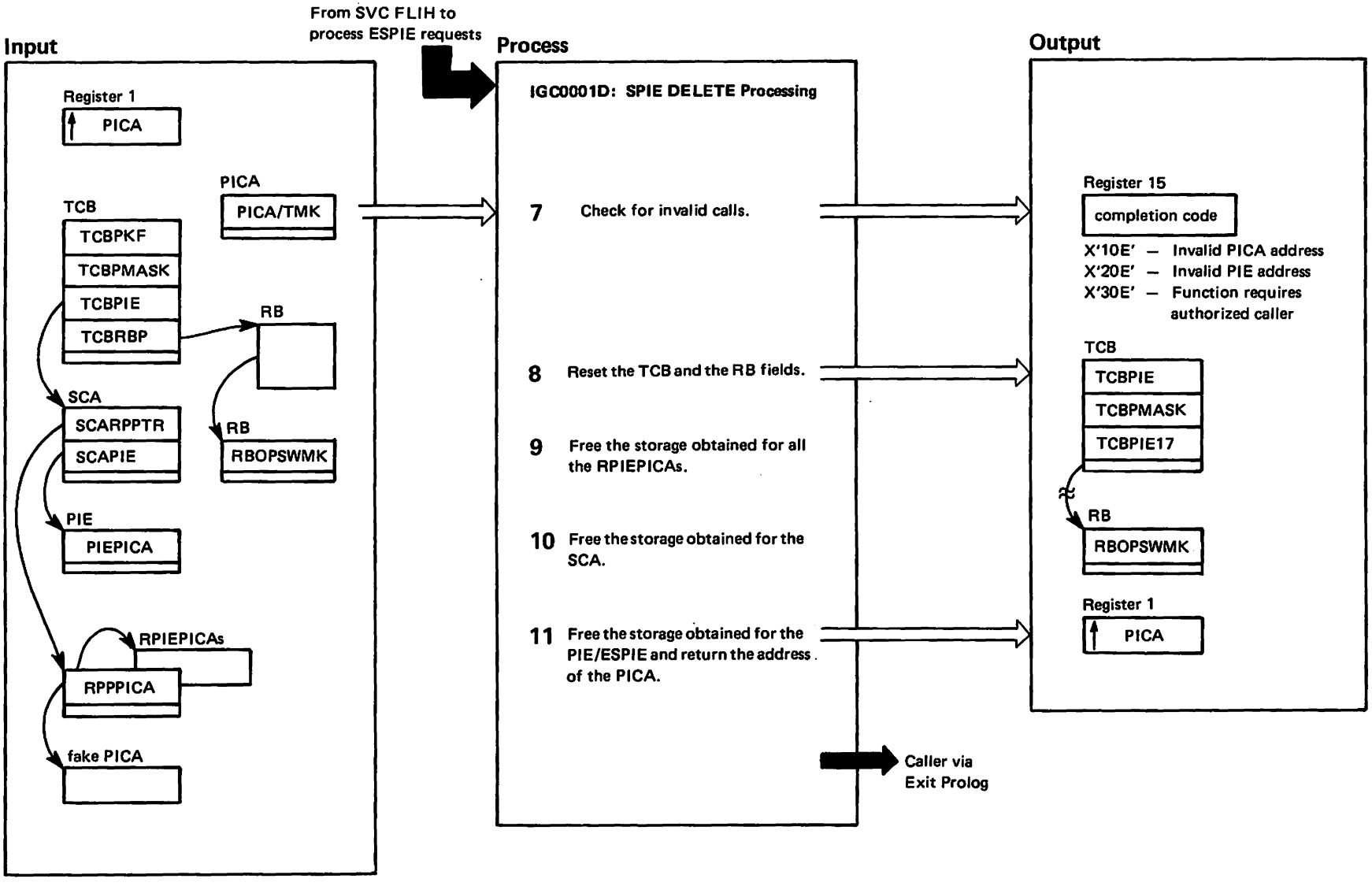
3 If a PIE does not exist (SCAPIE field equals 0), SPIE create processing obtains storage for a PIE and initializes it with a pointer to the user supplied PICA. SPIE create processing also obtains an extended program interruption element (EPIE) contiguous to the PIE so it will be available should the user specify an extended program interruption exit routine.

4 SPIE create processing uses the RPIEPICA for recovery processing. The SPIE create service initializes the RPIEPICA with pointers to the PIE, PICA, and RB issuing the SPIE, chains any previously active RPIEPICAs to the new RPIEPICA, and generates a token, representing the SPIE environment, from the TCBRTMCT field to represent this RPIEPICA.

5 If the previously active RPIEPICA represents an ESPIE exit, SPIE create processing obtains a fake PICA and initializes the fake PICA so that a SPIE restore request correctly restores the ESPIE exit.

6 On return to the issuer of the SPIE macro, register 1 contains the address of the previously active SPIE's PICA, the previously active ESPIE's fake PICA, or 0 if neither a SPIE nor an ESPIE was active.

IEAVTESP – SPIE/ESPIE Processing (Part 3 of 14)



IEAVTESP – SPIE/ESPIE Processing (Part 4 of 14)

Extended Description

The SPIE Delete processor (IGC0001D) receives control when a user issues a SPIE macro to delete a SPIE environment. When a user issues a SPIE macro with no operands, a zero PICA address is placed in register 1 during the macro expansion. A SPIE macro with no operands is a request to cancel all SPIEs or ESPIEs for the task.

SPIE processing always references the PIE and PICA in the key of the caller. An unauthorized reference to the PIE or PICA results in a program check. The SPIE functional recovery routine (FRR) converts the program check to either a X'10E' or X'20E' ABEND code.

7 If the caller is in supervisor state, is in a key other than that indicated in the TCBPKF field, or is executing in 31-bit addressing mode, the caller cannot use SPIE. SPIE processing issues an ABEND code.

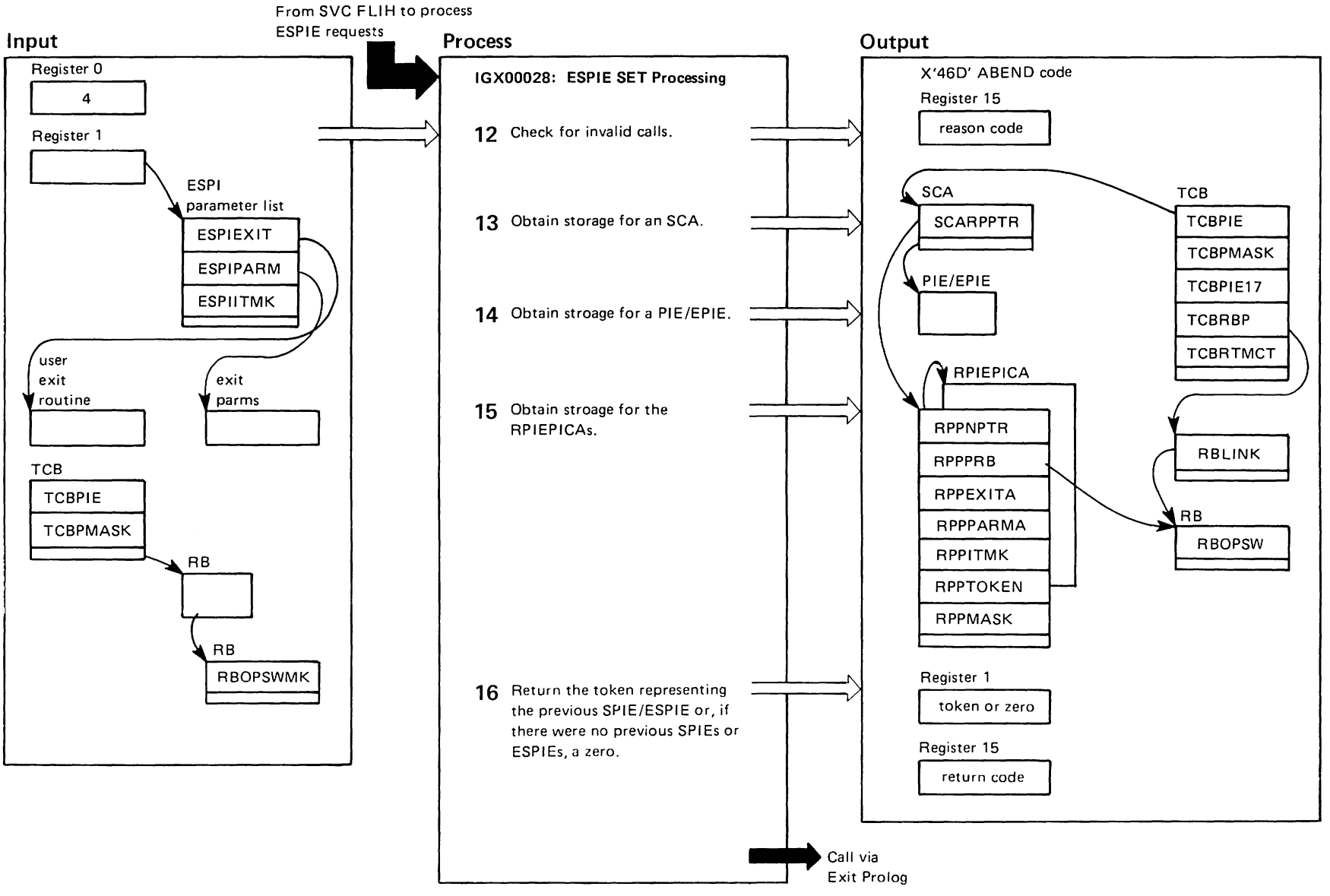
8 IGC0001D resets the TCB and RB fields.

9 IGC0001D frees the storage obtained for the RPIEPICAs and any fake PICAs anchored off the RPIEPICAs.

10 IGC0001D frees the storage obtained for the SCA.

11 IGC0001D frees the storage obtained for the PIE and EPIE and places the PICA address previously contained in the PIE into register 1 to pass back to the caller.

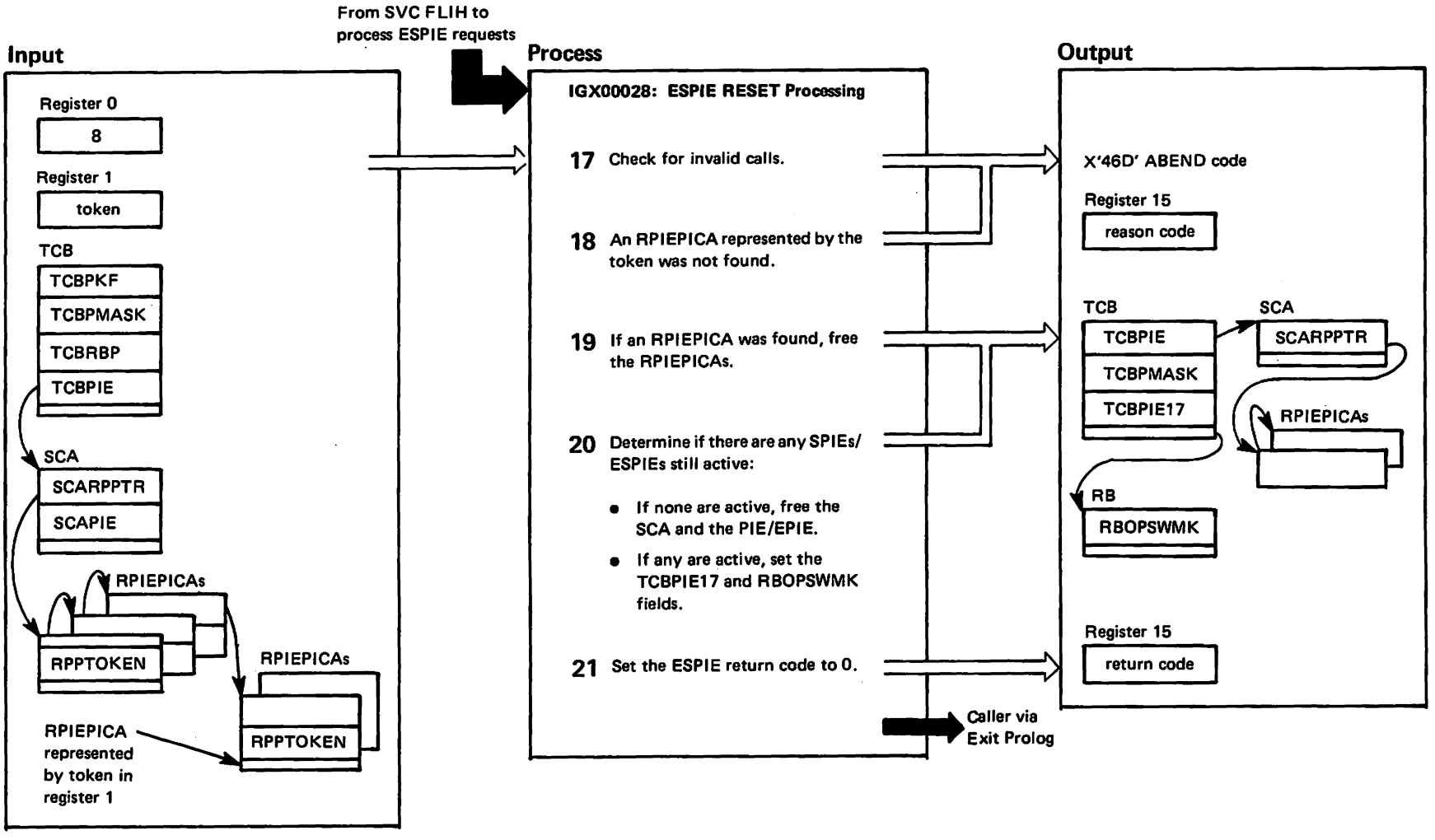
Module	Label
IEAVTESP	IGC0001D



Extended Description	Module	Label
ESPIE SET processing provides most of the SPIE functions to programs executing in 31-bit addressing mode as well as to 24-bit mode users. As does the SPIE service, the ESPIE service allows a program to establish an exit that is to receive control when a selected program interruption occurs. The ESPIE SET service records the program's selected program interruptions, exit address, and other control information in a control block (RPIEPICA) anchored off the TCB. If a program interruption occurs, the supervisor examines the recorded information to determine if the user exit routine should receive control to process the program interrupt.	IEAVTESP	IGX00028
12 The ESPIE SET function checks the validity of the information supplied in the ESPIE parameter list. The user's execution environment is also checked to ensure that the user is allowed to use the ESPIE services. If any errors are detected, ESPIE SET processing issues an X'46D' ABEND code and places a reason code into register 15 as follows:		CHECKENV
13 If the TCBPIE field equals 0, this is the first time the caller has issued an ESPIE or SPIE macro. ESPIE SET processing obtains storage for an SCA and initializes the SCA.		SETPROC
14 ESPIE SET processing obtains contiguous storage for the PIE and EPIE.		
15 The RPIEPICA contains recovery and control information. ESPIE SET processing obtains storage for the RPIEPICA and initializes it with information from the ESPIE parameter list, such as the user exit address and list of program interruptions to be processed. ESPIE SET processing generates a token to be used as input to the ESPIE RESET function, if required, to represent this RPIEPICA.		
16 On return to the issuer of the ESPIE macro, register 1 contains the token representing the previous SPIE or ESPIE or 0 if there are no previous SPIEs or ESPIEs.		

Hexadecimal code	*Reason*
4	An invalid function code was passed in register 0. The code was not that of SET, RESET, or TEST.
8	An invalid parameter list was passed. The area might not have been on a full-word boundary or might be in protected storage.
C	An invalid exit routine address might have been supplied or a field defined to be zero was found to be non-zero.
18	The caller of ESPIE was either in supervisor state or the execution key did not equal the TCB key.
20	The caller requested a function that required authorization, but the caller was not authorized.

IEAVTESP – SPIE/ESPIE Processing (Part 7 of 14)



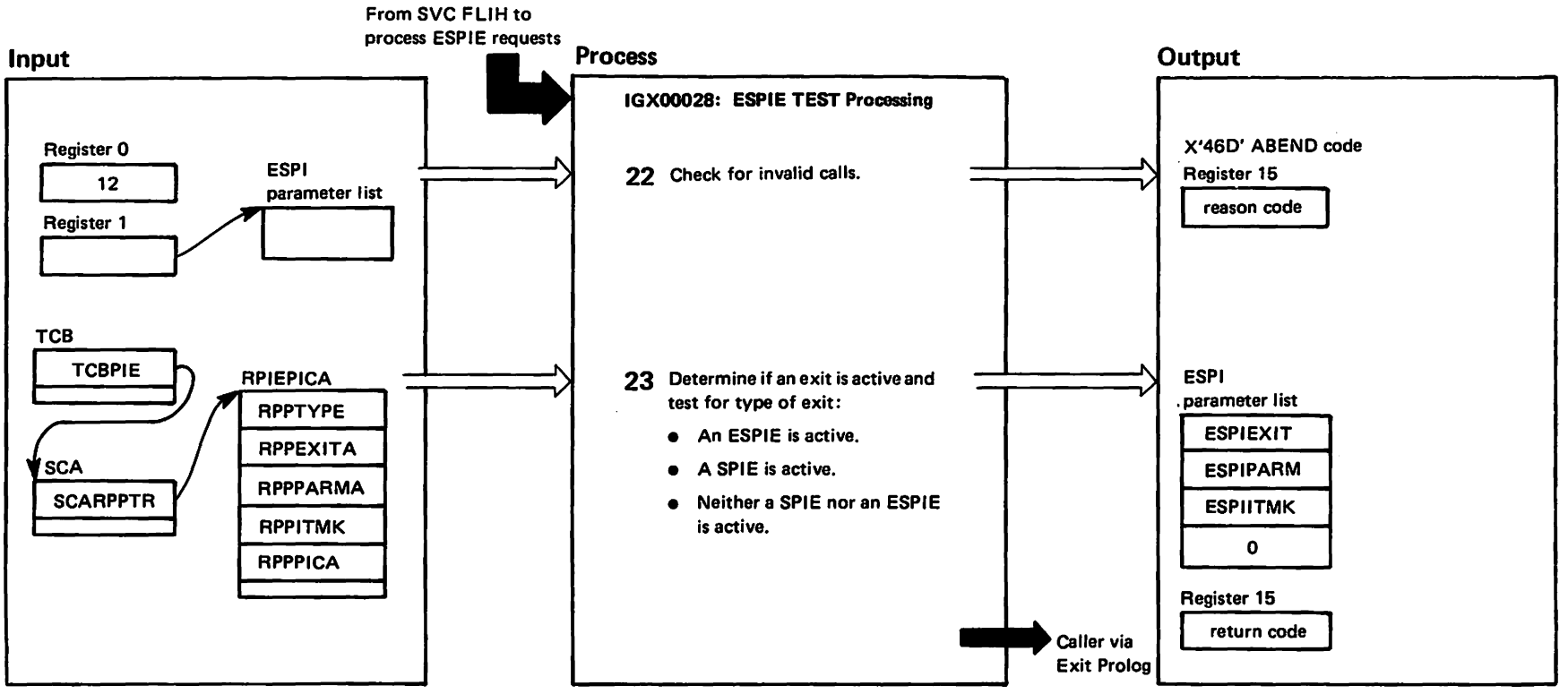
Extended Description	Module	Label
The ESPIE RESET function cancels all SPIE and ESPIE requests up to, but not including, the SPIE or ESPIE represented by the specified token. If a 0 token is specified, ESPIE RESET processing cancels all SPIEs and ESPIEs for the task.	IEAVTESP	IGX00028
17 ESPIE RESET processing checks the validity of the ESPIE input parameters and user's execution environment. If any errors are detected, ESPIE RESET processing issues a X'46D' ABEND code and places a reason code into register 15 as follows:		CHECKENV

Hexadecimal code	*Reason*
4	An invalid function code was passed in register 0. The code was not that of SET, RESET, or TEST.
18	The caller of ESPIE was either in supervisor state or the execution key did not equal the TCB key.

18 ESPIE RESET processing scans the RPIEPICA chain to verify that the token specified represents a valid SPIE or ESPIE environment. If not, the ESPIE RESET function issues a X'46D' ABEND code and places a reason code of 14 into register 15. If any RPIEPICAs on the chain prior to the specified RPIEPICA indicate an RB other than the RB of the issuing program, the request is invalid. The ESPIE RESET function issues a X'46D' ABEND code and places a reason code of 10 into register 15.

Extended Description	Module	Label
19 ESPIE RESET processing deletes all RPIEPICAs up to but not including the one rperesented by the specified token. Deleted RPIEPICAs are chained to the SCAFRPPQ as long as at least one ESPIE or SSPIE remains on the SCARPPTR. If the input token is zero, the ESPIE RESET function deletes all SPIEs and ESPIEs. The ESPIE RESET function also frees the storage obtained for all fake PICAs anchored off the freed RPIEPICAs.		RSETPROC
20 If no SPIEs or ESPIEs exits, the ESPIE RESET function frees the storage obtained for the SCA and PIE/EPIE and resets the TCB and RB fields. If any SPIEs or ESPIEs are still active, the SPIE RESET function sets the TCBPIE17and RB's program mask according to the information saved in the RPIEPICA and PICA.		

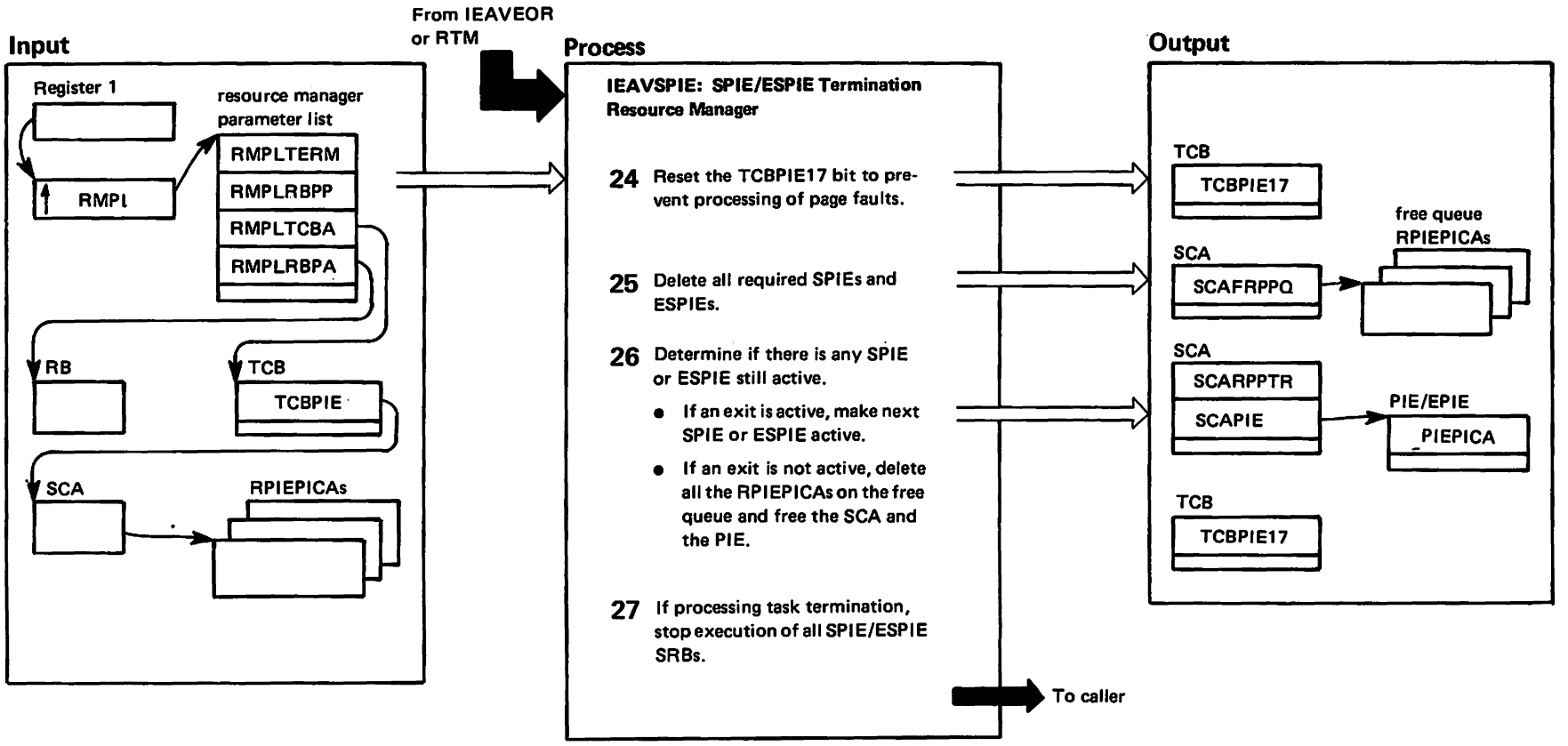
IEAVTESP – SPIE/ESPIE Processing (Part 9 of 14)



IEAVTESP – SPIE/ESPIE Processing (Part 10 of 14)

Extended Description	Module	Label
The ESPIE TEST function determines if an exit is active and the exit type. ESPIE TEST processing initializes the caller's parameter list with appropriate SPIE or ESPIE parameter information dependent on the current exit type and sets a return code that indicates whether a SPIE, ESPIE, or no exit is currently active.	IEAVTESP	IGX00028
22 ESPIE TEST processing checks the validity of the ESPIE input parameters and user's execution environment. If any errors are detected, the ESPIE TEST function issues a X'46D' ABEND code and places a reason code in register 15 as follows:		CHECKINV
23 ESPIE TEST processing checks the RPPTYPE field to determine whether the active exit is a SPIE or an ESPIE exit. If an ESPIE exit is active, ESPIE TEST processing copies the ESPIE parameter list that defined the exit and sets a return code of 0. If a SPIE exit is active, ESPIE TEST processing sets the ESPIPARM field of the ESPIE parameter list equal to the address of the current PICA and sets a return code of 4. If neither a SPIE or an ESPIE is active, ESPIE TEST processing sets a return code of 8.		TESTPROC

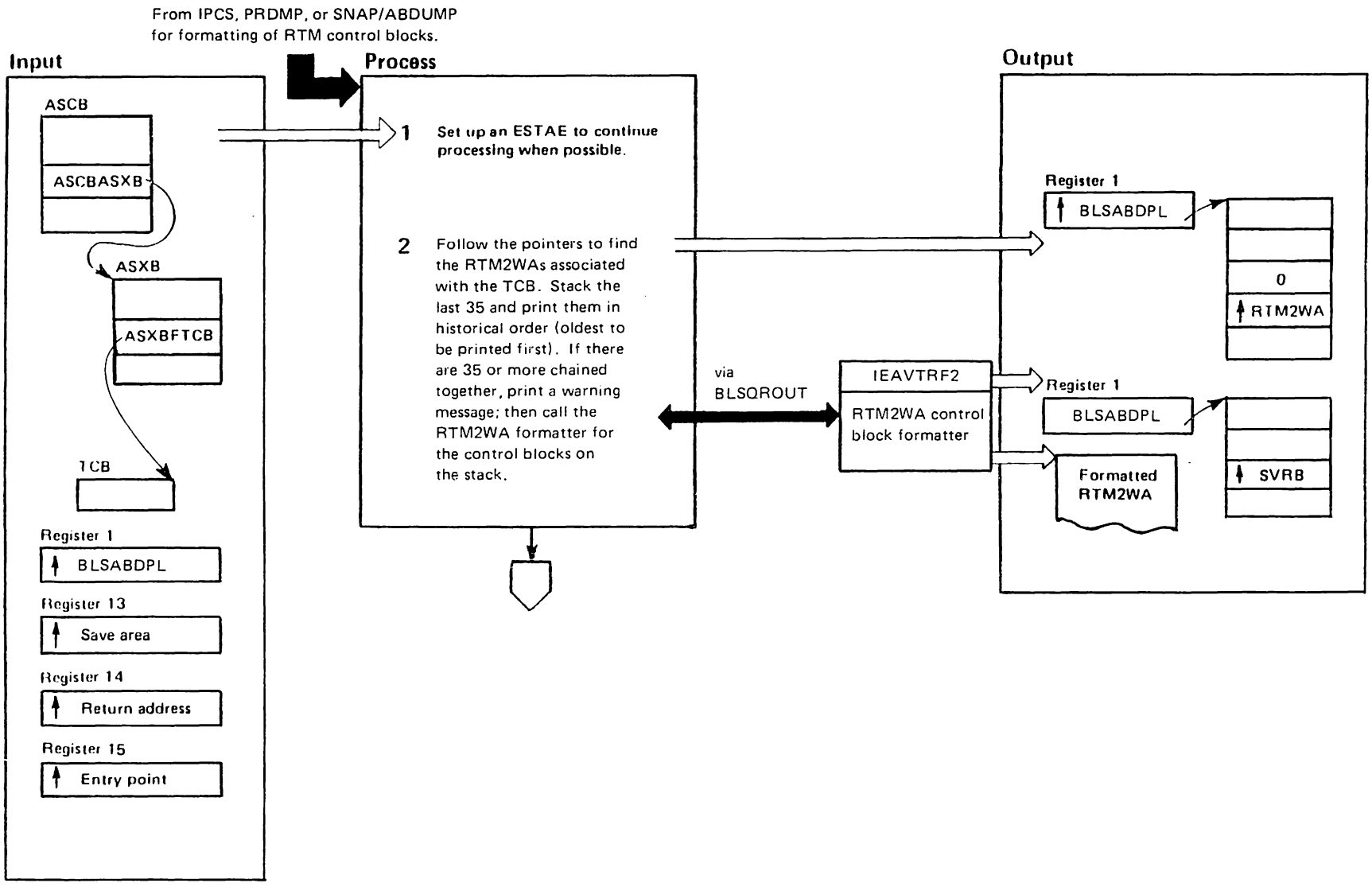
Hexadecimal code	*Reason*
4	An invalid function code was passed in register 0. The code was not that of SET, RESET, or TEST.
8	An invalid parameter list was passed. The area might not have been on a full-word boundary or might be in protected storage.
18	The caller of ESPIE was either in supervisor state or the execution key did not equal the TCB key.



IEAVTESP – SPIE/ESPIE Processing (Part 14 of 14)

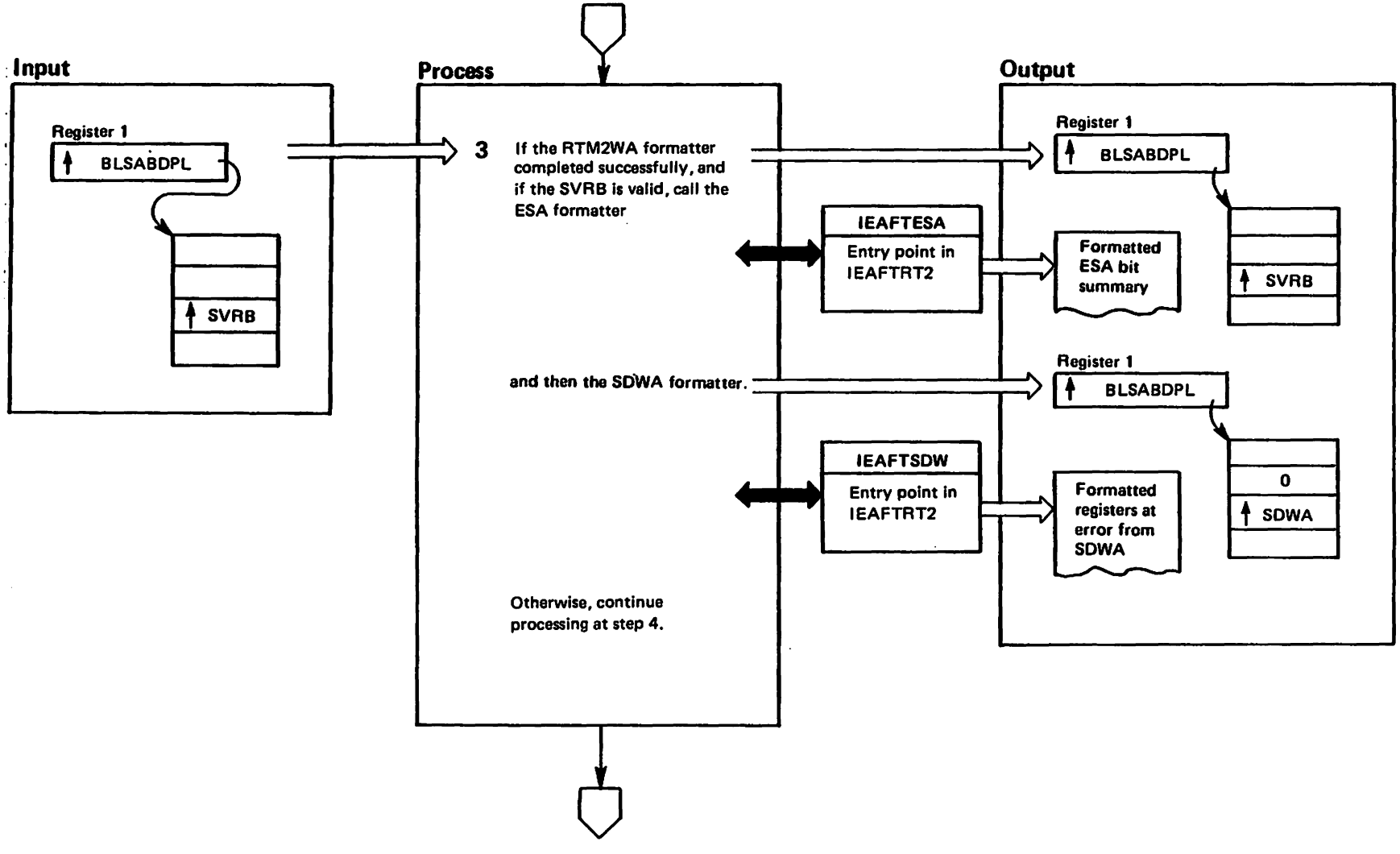
Extended Description	Module	Label
Checkpoint/restart calls IEAVSPI to save or restore SPIE and ESPIE environments or to determine the number of subsystem checkpoint record (SSCR) blocks required for checkpoint. The SSCR contains flags indicating whether the environment should be saved, restored, or whether a count of SSCRs should be calculated.	IEAVTESP	IEAVSPI
28 When performing checkpoint processing, IEAVSPI copies the information required to save the SPIE/ESPIE environment in the SSCR. If the page fault processing bit in the TCB (TCBPIE17) is on, IEAVSPI indicates this in the SSCR. IEAVSPI copies the program mask from the TCB, the SPIE control area (SCA), and all RPIEPICAs to the SSCR. IEAVSPI replaces the RB address in each copied RPIEPICA with a number that represents that RB's relative position from the TCB. That is, if the RPIEPICA points to the RB that is pointed to by the TCB, that RPIEPICA's RB address would be replaced with the value 1. If there is no more room in the SSCR for the RPIEPICAs, IEAVSPI uses the SSCR chain field to obtain the address of the next SSCR to be used.		VSPICHCP
29 When performing restart processing, IEAVSPI restores the SPIE/ESPIE environment according to information saved in the SSCR. IEAVSPI obtains storage for the SCA and initializes it from the SCA copy in the SSCR. IEAVSPI restores the TCBPIE17 bit and the program mask in the TCB. IEAVSPI obtains enough storage to contain all the RPIEPICAs, copies the RPIEPICAs from the SSCRs to this storage, and chains the RPIEPICA to the SCA. IEAVSPI replaces the relative RB number in each RPIEPICA with the actual RB address.		VSPIRSRT
30 IEAVSPI scans the active RPIEPICA chain anchored off the SCA to determine the number of SPIE and ESPIE environments that must be checkpointed. Using the number of SPIE and ESPIE environments, IEAVSPI calculates the number of required SSCR blocks needed. If no SPIE or ESPIE environments exist, IEAVSPI returns a zero count to checkpoint/restart and the checkpoint restart functions of IEAVSPI are not called. If at least one SPIE or ESPIE exists, IEAVSPI calculates the number of SSCRs.		VSPINUMC

IEAVTFMT – RTM Control Block Formatter (Part 1 of 12)

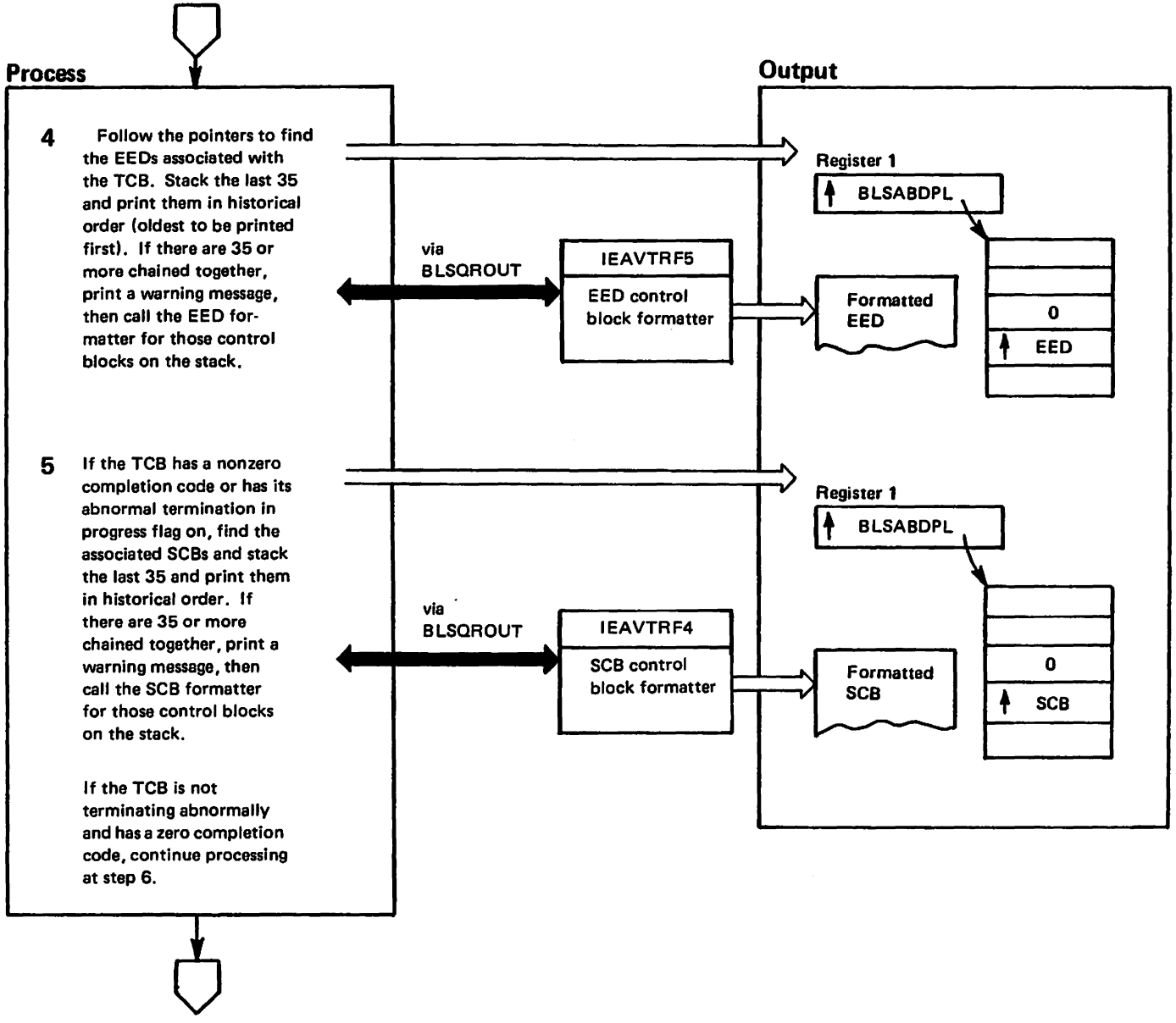


Extended Description	Module	Label
IEAVTFMT searches the control block chains to find the RTM information associated with the TCB passed in the parameter list. This information, if accessed through normal chain pointers and contained in the dump, is the RTM2WA's, EEDs, and SCBs pointed to by the TCB. It is also RTM1 information such as the FRR stack, for a current task, the IHSA for any interrupted or suspended task, the XSB or the STKE.	IEAVTFMT	
1 This ESTAE routine will simulate the PRDMP access service routine on errors caused by accessing dump data under SNAP. It will set a return code of 4 and continue processing. If the error was not a result of accessing dump data, a message will be printed with the abend code and control will return to the calling program.	IEAVTFMT	ESTAERTN
2 The TCB has a pointer to the RTM2WA. Save the address of the 35 most recent RTM2WAs in a stack. If there are 35 or more in the chain, print out a message text warning of the possibility of a loop in the RTM2WA chain.		

Extended Description	Module	Label
To format and print each RTM2WA in the stack, IEAVTFMT performs the following processing:	IEAVTFMT	RTM2RTN
1. IEAVTFMT calls BLSQROUT (Exit Services Router) to pass the requested service code (in this case the format service) and to pass the control block's acronym.		
2. BLSQROUT calls BSLQCFMT (Control Block Formatter) to check the passed control block acronym with the acronym entries in the control block acronym table (CBAT) and to load the requested control block formatter module (IEAVTRF2).		
3. BLSQCFMT calls IEAVTRF2 (RTM2WA Control Block Formatter) to pass to BLSQROUT the address of the RTM2WA formatting model, the address of the dump data to be formatted, and the requested service code.		
4. BLSQROUT then calls BLSQIFMT (Control Block Formatter Model) which loads the control block data to be formatted and formats the control block using RTM2WA's formatting model CSECT (IEAVTRP2).		
5. Finally, IEAVTRF2 performs a bit analysis summary.		

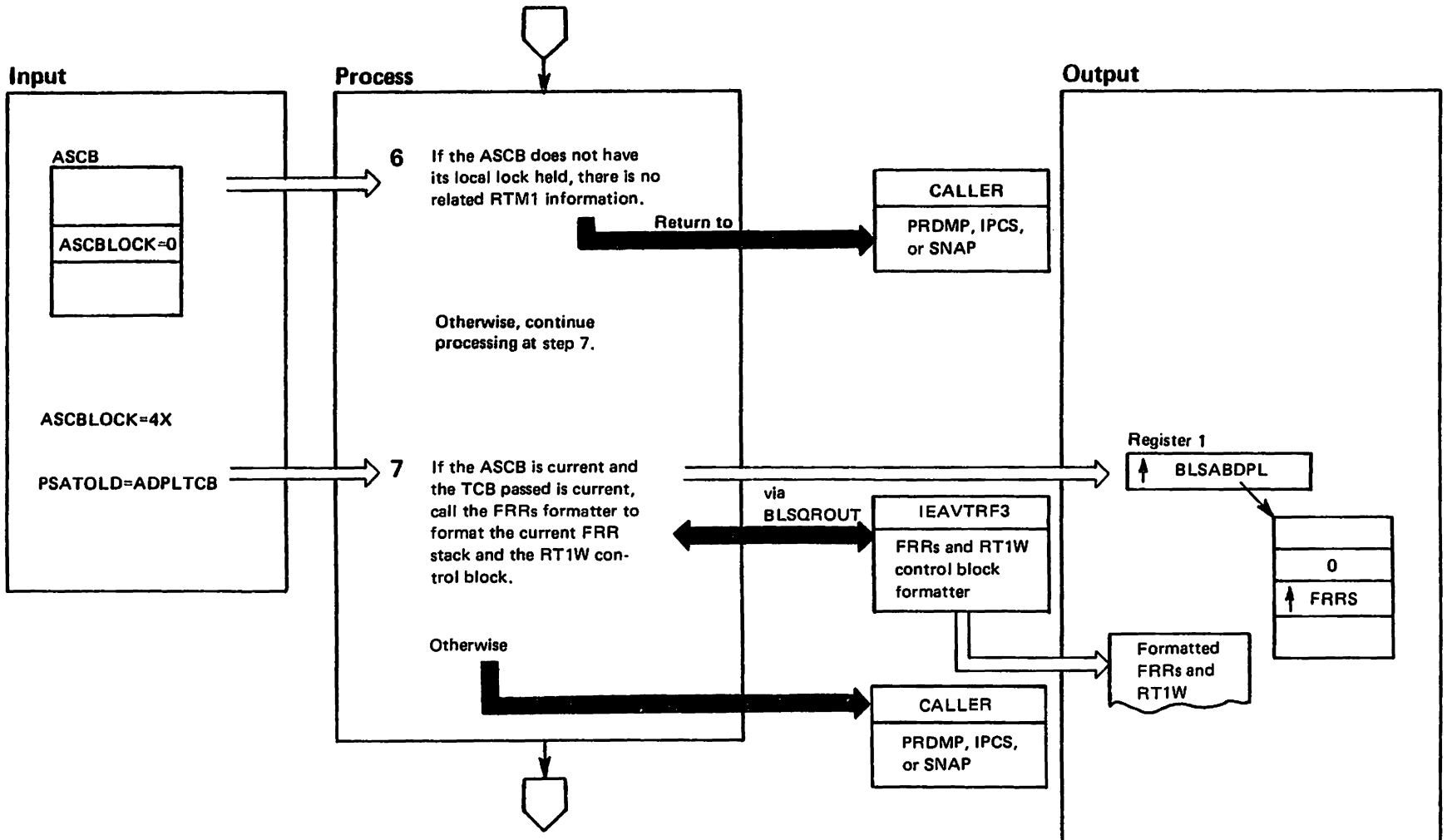


Extended Description	Module	Label
3 If the RTM2WA formatted without any problems (return code 0), IEAVTFMT will access the address of the related SVRB. The SVRB's extended save area contains information indicating reasons for entry to RTM1 or RTM2. IEAFTESA (an entry point in IEAFTRT2), formats these bits.	IEAVTFMT	RTM2RTN
	IEAFTRT2	
IEAVTFMT then locates and passes the SDWA to the SDWA formatter, IEAFTSDW (an entry point in IEAFTRT2), to print out the registers saved there at the time of the error.	IEAFTRT2	

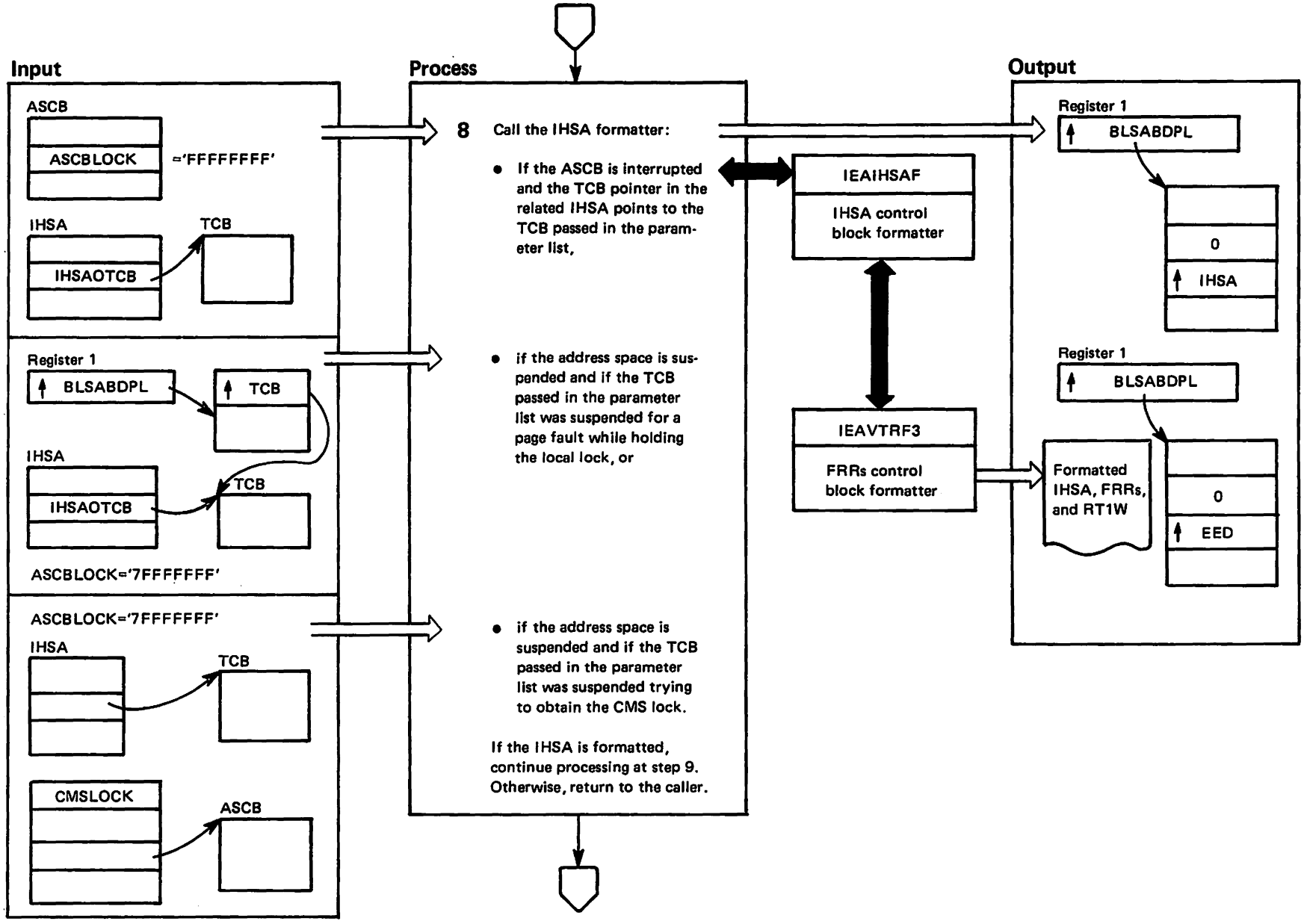


Extended Description	Module	Label
4 The TCB has a pointer to the EED. Save the address of the 35 oldest EEDs in a stack. If there are 35 or more in the chain, print out a message text warning of the possibility of a loop in the RTM2WA chain.	IEAVTFMT	EEDRTN
To format and print each EED in the stack IEAVTFMT performs the following processing: 1. IEAVTFMT calls BLSQROUT (Exit Services Router) to pass the requested service code (in this case the format service) and to pass the control block's acronym. 2. BLSQROUT calls BLSQCFMT (Control Block Formatter) to check the passed control block acronym with the acronym entries in the control block acronym table (CBAT) and to load the requested control block formatter module (IEAVTRF5). 3. BLSQCFMT calls IEAVTRF5 (EED Control Block Formatter) to pass to BLSQROUT the address of the EED formatting model, the address of the dump data to be formatted, and the requested service code. 4. BLSQROUT then calls BLSQIFMT (Control Block Formatter Model) which loads the control block data to be formatted and formats the control block using EED's formatting model CSECT (IEAVTRP5). 5. Finally, IEAVTRF5 performs a bit analysis summary.	IEAVTRF5	

Extended Description	Module	Label
5 Check whether the TCB has a nonzero return code or the TCB has the abnormal termination in progress flag on. Only under these circumstances does IEAVTFMT format the SCBs. The TCB has a pointer to the SCB. Save the address of the 35 most recent SCBs in a stack. If there are 35 or more in the chain, print out a message text warning of the possibility of a loop in the SCB chain.	IEAVTFMT	SCBRTN
To format and print each SCB in the stack IEAVTFMT performs the following processing: 1. IEAVTFMT calls BLSQROUT (Exit Services Router) to pass the requested service code (in this case the format service) and to pass the control block's acronym. 2. BLSQROUT calls BLSQCFMT (Control Block Formatter) to check the passed control block acronym with the acronym entries in the control block acronym table (CBAT) and to load the requested control block formatter module (IEAVTFF4). 3. BLSQCFMT calls IEAVTRF4 (SCB/SCBX Control Block Formatter) to pass to BLSQROUT the address of the SCB/SCBX formatting model, the address of the dump data to be formatted, and the requested service code. 4. BLSQROUT then calls BLSQIFMT (Control Block Formatter Model) which loads the control block data to be formatted and formats the control block using SCB/SCBX's formatting model CSECT (IEAVTRP4). 5. Finally, IEAVTRF4 performs a bit analysis summary.	IEAVTRF4	

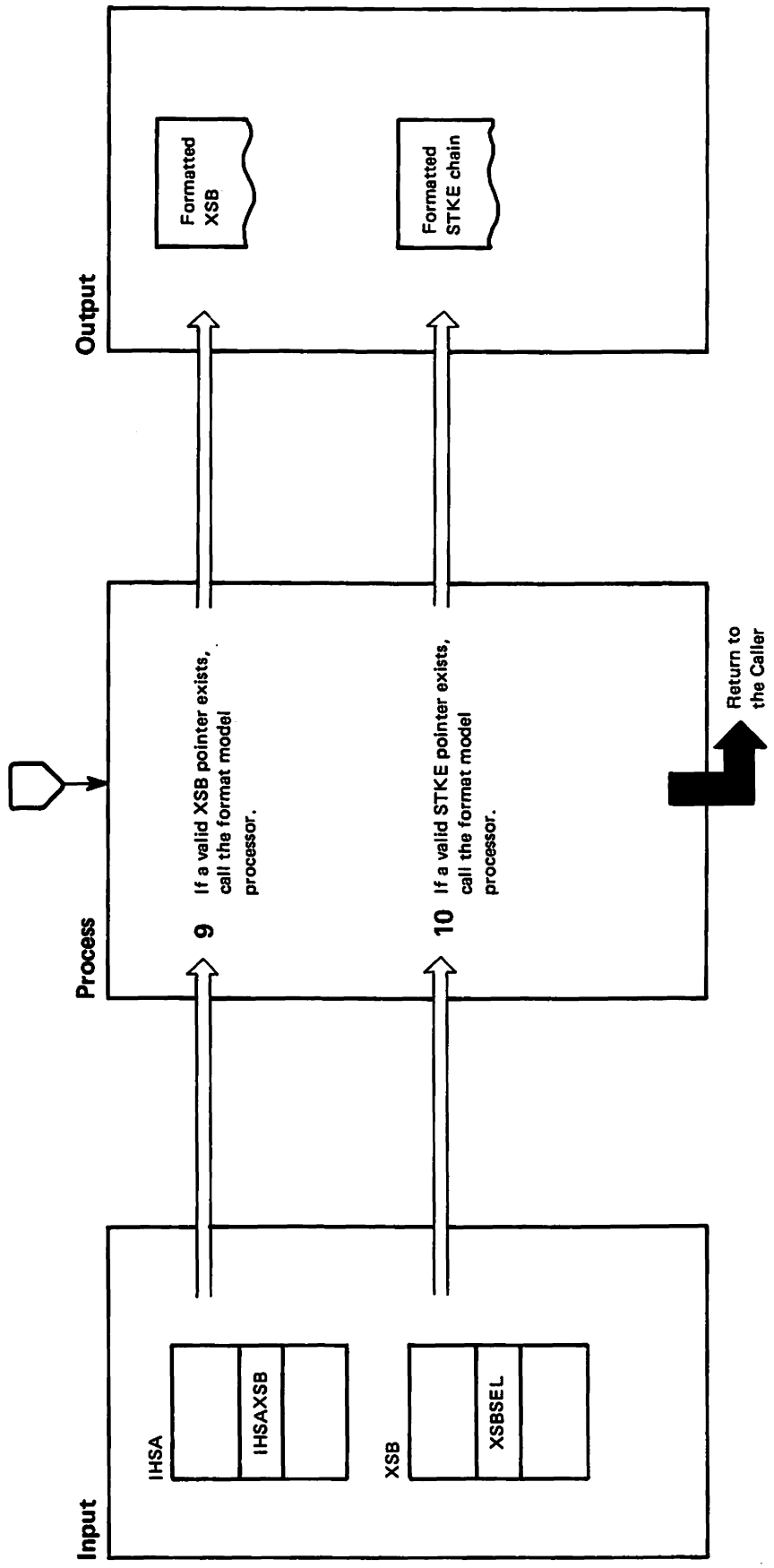


Extended Description	Module	Label
6 If the ASCB does not have its local lock held (ASCBLOCK=0), there is no RTM1 related information to be formatted. Return to the caller.	IEAVTFMT	RT1MAIN
7 If the TCB passed in the parameter list is the current task on any processor and it holds the local lock, find the correct PSA and place the FRR's pointer into the parameter list. (To determine the correct PSA, the ASCBLOCK contains the logical processor address of the processor in which this task was running. The PSA also contains the logical processor address for the processor with which it is associated. Thus it can be determined which PSA in an MP system contains the current RTM1 information.)		
To format and print the current FRR stack IEAVTFMT performs the following processing:	IEAVTRF3	
1. IEAVTFMT calls BLSQROUT (Exit Services Router) to pass the requested service code (in this case the format service) and to pass the control block's acronym.		
2. BLSQROUT calls BLSQCFMT (Control Block Formatter) to check the passed control block acronym with the acronym entries in the control block acronym table (CBAT) and to load the requested control block formatter module (IEAVTRF3).		
3. BLSQCFMT calls IEAVTRF3 (FRR Control Block Formatter) to pass to BLSQROUT the address of the FRR formatting model, the address of the dump data to be formatted, and the requested service code.		
4. BLSQROUT then calls BLSQIFMT (Control Block Formatter Model) which loads the control block data to be formatted and formats the control block using FRR's formatting model CSECT (IEAVTRP3).		
To format and print the RT1W in the FRR stack IEAVTRF3 performs the following processing:		
1. IEAVTRF3 (FRR Control Block Formatter) provides the address of the RT1W formatting model, the address of the dump data to be formatted, and the requested service code to pass to BLSQROUT.		
2. BLSQROUT then calls BLSQIFMT (Control Block Formatter Model) which loads the control block data to be formatted and formats the control block using RT1W's formatting model CSECT (IEAVTRP1).		
3. Finally, IEAVTRF3 performs a bit analysis summary.		
If RT1W is valid and if the pointer to the EED is nonzero, each EED in the stack is formatted. (See Step 4's extended description for an explanation of the EED processing.)		



Extended Description	Module	Label
8 IEAVTFMT performs the following checks to determine whether the IHSA contains RTM1 information that is pertinent to the current TCB. If any one of these checks is valid, the IHSA is formatted. Otherwise, the IHSA does not contain valid information and, therefore, control returns to the caller.		
IEAVTFMT's first check determines if the address space was interrupted (ASCBLOCK=FFFFFFFF) while holding the local lock and if the interrupt handler save area (IHSA) points to the TCB passed in the parameter list. If so, the IHSA contains RTM1 information pertinent to the TCB being formatted.	IEAVTFMT	INTERRUP
IEAVTFMT's second check determines if the address space is suspended and if the TCB was in control. One way to determine this is to check the status of the TCB. If the TCB was suspended for a page fault while holding the local lock, the IHSA contains RTM1 information pertinent to the TCB being formatted.	IEAVTFMT	SUSPEND
IEAVTFMT's last check determines if the address space was suspended while trying to obtain the CMS lock. IEAVTFMT searches the CMS suspend queue for the ASCB address. If it is on the queue, the TCB was in control and the IHSA contains related RTM1 information.	IEAVTFMT	CMSEARCH
Once IEAVTFMT has determined that the IHSA has valid information, IEAVTFMT formats and prints the IHSA through the following processing:	IEAIHSAF	

Extended Description	Module	Label
1. IEAVTFMT calls BLSQROUT (Exit Services Router) to pass the requested service code (in this case the format service) and to pass the control block's acronym.		
2. BLSQROUT calls BLSQCFMT (Control Block Formatter) to check the passed control block acronym with the acronym entries in the control block acronym table (CBAT) and to load the requested control block formatter module (IEAIHSAF).		
3. BLSQCFMT calls IEAIHSAF (IHSA Control Block Formatter) to pass to BLSQROUT the address of the IHSA formatting model, the address of the dump data to be formatted, and the requested service code.		
4. BLSQROUT then calls BLSQIFMT (Control Block Formatter Model) which loads the control block data to be formatted and formats the control block using IHSA's formatting model CSECT (IEAIHSAP).		
IEAIHSAF formats and prints any active FRRs in the stack and RT1W control blocks through module IEAVTRF3. (See Step 7's extended description for an explanation of the FRR and RT1W formatting process.)		
If RT1W is valid and if the pointer to the EED is nonzero, the EEDs are formatted. (See Step 4's extended description for an explanation of the EED processing.)		



IEAVTFMT - RTM Control Block Formatter (Part 11 of 12)

Diagram RTM-7. IEAVTFMT – RTM Control Block Formatter (Part 12 of 12)

Extended Description **Module** **Label**

9 If the XSB pointer in the IHSA is not zero, IEAVTFMT formats and prints the XSB in the following manner:

a. IEAVTFMT calls BLSQROUT (Exit Services Router) to pass the requested service code (in this case the format model service) and to pass the control block's acronym.

b. BLSQROUT then calls BLSQIFMT (Control Block Formatter Model) which loads the control block data to be formatted. BLSQIFMT formats the control block using the specified formatting model, IEAXSBP.

10 If the STKE pointer in the XSB is not zero, IEAVTFMT formats and prints the STKEs in the following manner:

a. IEAVTFMT calls BLSQROUT (Exit Services Router) to pass the requested service code (in this case the format model service) and to pass the control block's acronym.

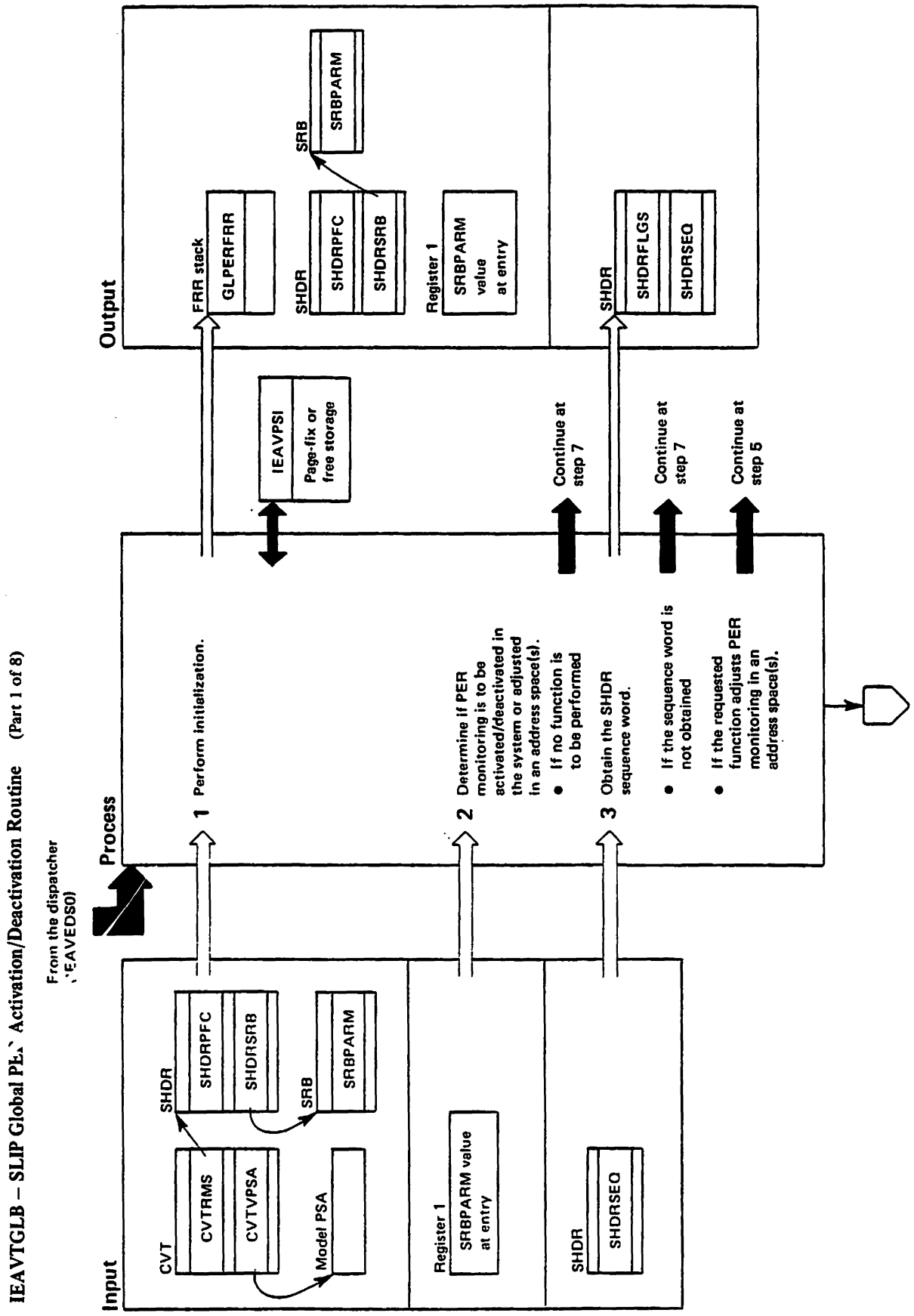
b. BLSQROUT then calls BLSQIFMT (Control Block Formatter Model) which loads the control block data to be formatted. BLSQIFMT formats the control block using the specified formatting model, IEASTKEP.

Extended Description

The following chart summarizes IEAVTFMT's formatting process of RTM's control blocks.

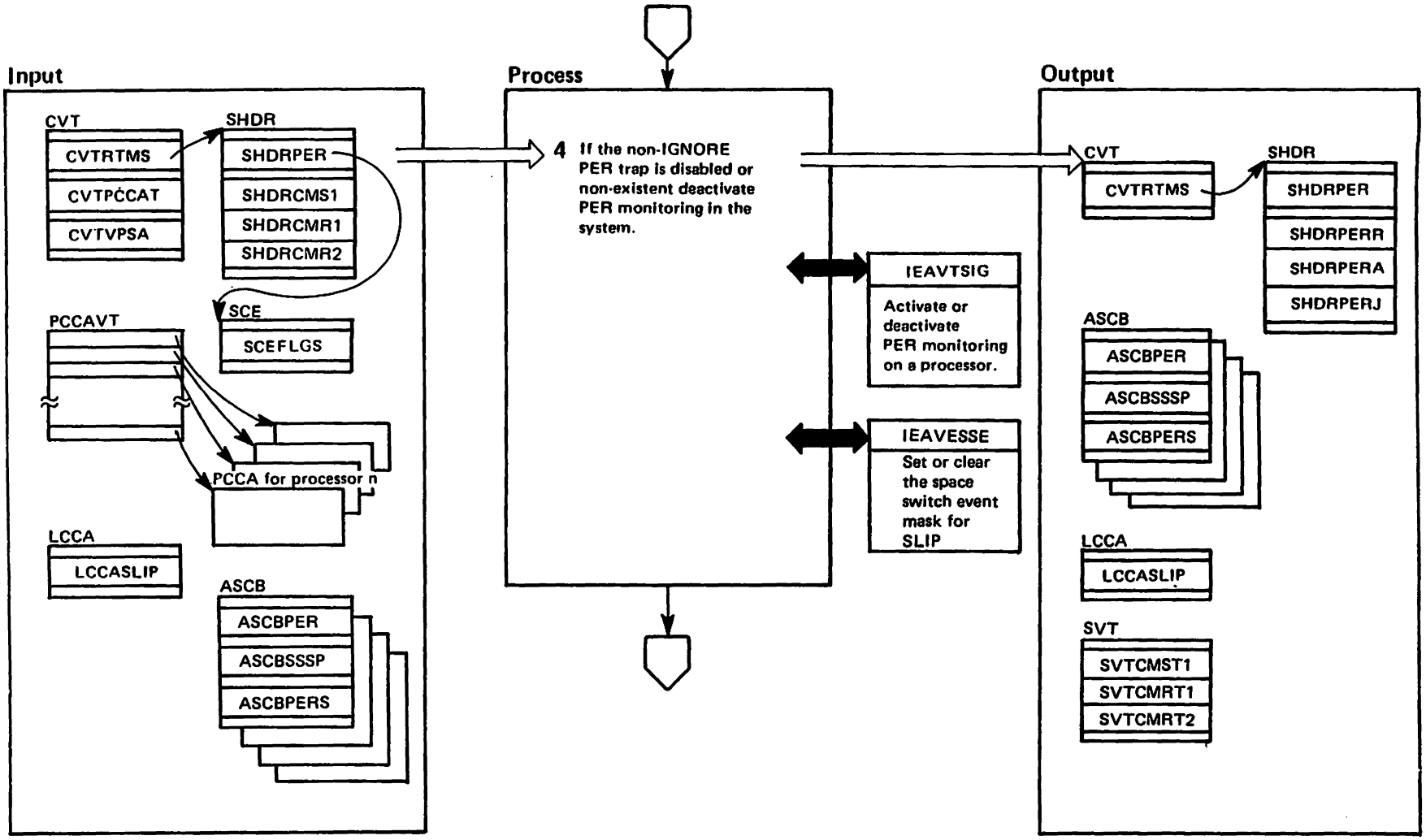
Control Block	*Formatter Module*	*Model*	*Bit Analysis Summary*	*Extended Description Step Number*
RTM2WA	IEAVTRF2	IEAVTRP2	Yes	2
EED	IEAVTRF5	IEAVTRP5	Yes	4
SCB	IEAVTRF4	IEAVTRP4	Yes	5
FRR	IEAVTRF3	IEAVTRP3	No	7
RT1W	IEAVTRF3	IEAVTRP1	Yes	7
IHSA	IEAIHSAF	IEAIHSAP	No	8
XSB	N/A	IEAXSBP	No	9
STKE	N/A	IEASTKEP	No	10

IEAVTGLB - SLIP GLOBAL PER ACTIVATION/DEACTIVATION ROUTINE

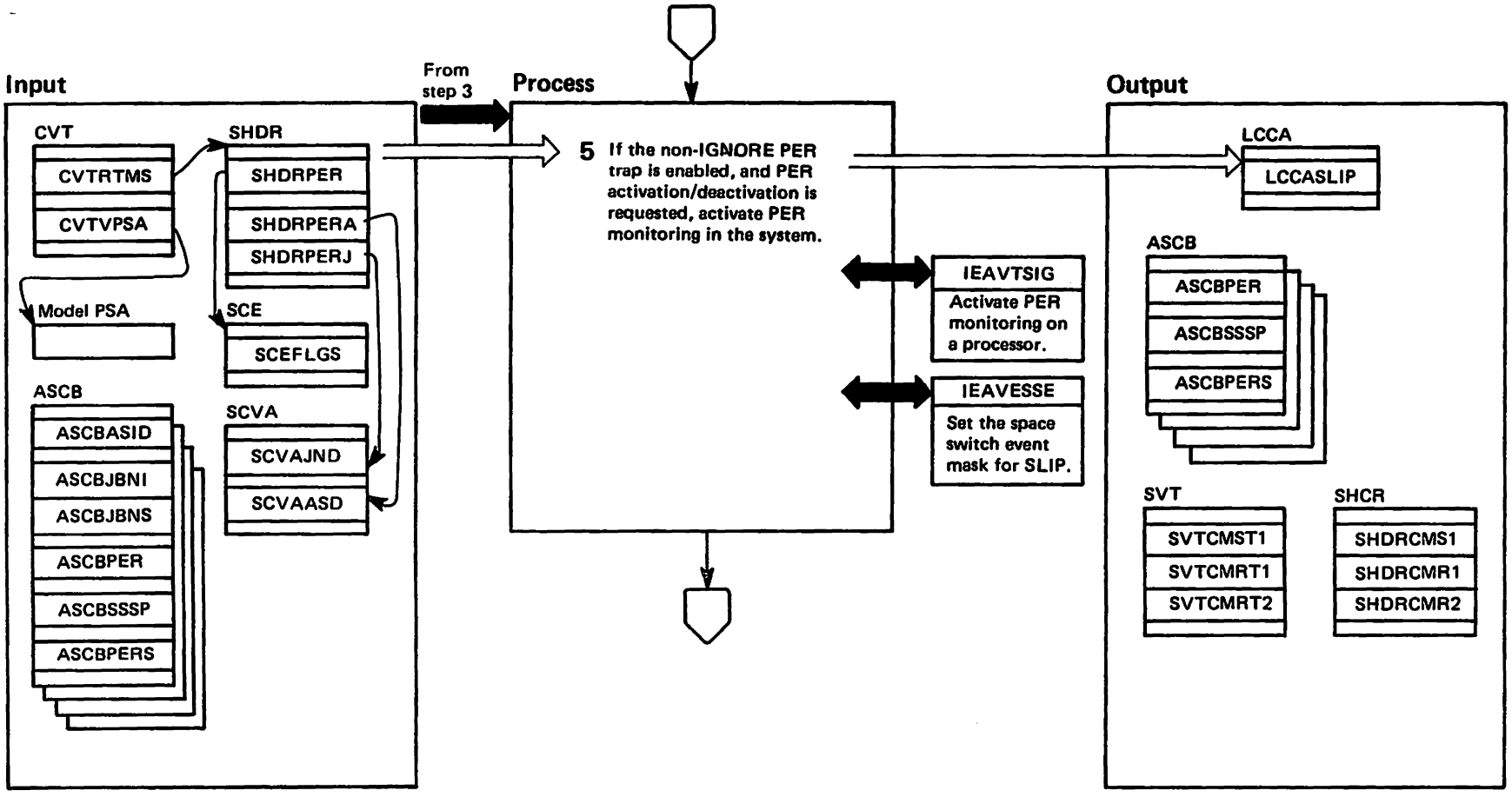


Extended Description	Module	Label
IEAVTGLB receives control from: • The dispatcher as the result of a global SRB being scheduled. • The SLIP command processor (IEECB905) when a SLIP user issues a command to enable or disable a non-IGNORE PER trap, or when IEECB905 detects that the SHDRSRBR flag is set, indicating that IEAVTGLB is to be rescheduled • The SLIP action processor (IEAVTSLS) when a non-IGNORE PER trap is to be disabled • The SLIP PER select interface routine (IEAVTJBN) when it is unable to acquire an SRB to schedule the SLIP local PER activation/deactivation routine (IEAVTLCL) • IEAVTLCL when it is unable to acquire the SHDR sequence word • IEAVTLCL when PVTMOD PER processing is activated and the PER control registers need to be set.		
IEAVTGLB either activates or deactivates PER monitoring in the system, or, if PER monitoring is already activated and is to remain active, adjusts PER control in the address spaces requiring a change in PER status.		
1 IEAVTGLB performs the following initialization functions. • Issues a SETFRR macro to add GLPERFRR to the FRR stack. • If the SLIP use counter (SHDRPFC) has not already been updated for this entry, adds one to the counter. This prevents the IEAVTSLP load module, which contains this CSECT, from being page-freed. • Obtains the LOCAL lock. • Page-fixes the model PSA if the model PSA exists and has not been page-fixed. • Obtains the CMS, SALLOC, and dispatcher locks. • Makes the SRB available by setting to one the SHDRSRBA flag in the SHDRSRB pointer. • Saves the contents of the SRBPARM field in register 1 and puts zeros in the SRBPARM field.	IEAVTGLB IEAVPSI	

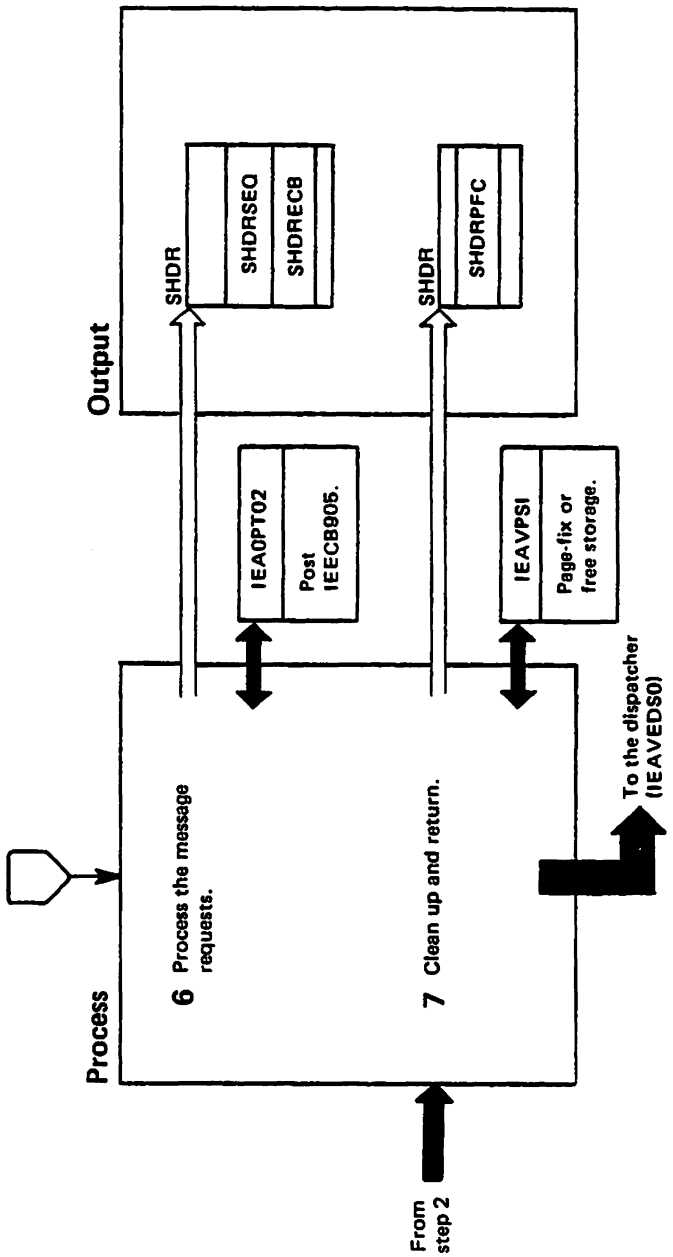
Extended Description	Module	Label
2 If the SRBPARM input value is negative, IEAVTGLB activates or deactivates PER monitoring on all processors in the system. If the value is positive, IEAVTGLB adjusts PER monitoring in all address spaces requiring a change in PER status. In either case, processing continues at the next step. If the value is zero, no function is performed, and processing continues at step 7.		
3 IEAVTGLB uses a CS (compare and swap) instruction to obtain the SHDR sequence word (SHDRSEQ), which serializes this routine with IEECB905 to prevent the SCE chain from being altered. Before attempting to obtain the sequence word, IEAVTGLB turns on the SHDRSRBR flag in the SHDRFLGS field. If the sequence word is not obtained, the SHDRSRBR flag is left at one, indicating to the routine that owns the sequence word that it is to reschedule IEAVTGLB when it releases the sequence word. Processing continues at step 7, where IEAVTGLB cleans up and returns to the dispatcher. If the sequence word is obtained, IEAVTGLB sets the SHDRSRBR flag to zero. When the requested function is to activate/deactivate PER monitoring in the system, processing continues at the next step. When PER monitoring is to be adjusted in an address space(s), processing continues at step 5.	IEAVTGLB	



Extended Description	Module	Label
4 If there is no non-IGNORE PER trap (SHDRPER=0) or the existing one is disabled (the SCEDSABL=1), IEAVTGLB deactivates PER monitoring in the system. To do so, IEAVTGLB:	IEAVTGLB	
a. Restores CMSET entry point addresses. b. Turns off PER monitoring in each active processor. c. Frees SLIP storage. d. Sets model PSA PSW PER bits to zero. e. Sets the PER trap pointers in the SHDR to zero. f. Turns off space switch and PER monitoring in all address spaces in which PER was activated.		
Each step is explained below in greater detail.		
a. *Restore CMSET entry point addresses.* IEAVTGLB restores the CMSET entry point addresses in the SVT that were saved in the SHDR when PER was activated.	IEAVTSIG	
b. *Turn off PER globally.* IEAVTGLB initializes a parameter list and passes control to IEAVTSIG, which sets PER control registers 9-11 and the external, I/O, and SVC new PSW PER bits to zero. In a multiprocessing environment, IEAVTGLB does this for each active processor in the PCCAVT. For active processors other than the one on which this module is executing, IEAVTGLB uses a RISGNL macro to pass control to IEAVTSIG. To turn off PER monitoring in this module's processor, IEAVTGLB calls IEAVTSIG directly.		
c. *Free SLIP storage.* After a processor has been signalled and before locating the next processor in the PCCAVT, IEAVTGLB attempts to free allocated SLIP storage. If the SLIP work area pointer is valid and the work area is not in use by IEAVTPER or IEAVTSLP (LCCASLIP > 0). IEAVTGLB frees the storage and sets the LCCASLIP value to zero. If the storage is being used (LCCASLIP<0), IEAVTGLB indicates that IEECB905 is to be posted to have this module rescheduled.	IEAVTGLB	
d. *Set the model PSA PSW PER bits to zero.* If the model PSA exists (CVTVPSA≠0) and is page-fixed, IEAVTGLB sets the EXT, SVC, and I/O new model PSA PSW PER bits to zero. If the model PSA is not page-fixed and an enabled non-IGNORE PER trap exists, IEAVTGLB indicates that IEECB905 is to be posted to issue message IEA424I. (When a processor is varied online, the model PSA is copied into the new processor's PSA.)		
e. *Set the PER trap pointers to zero.* IEAVTGLB sets to zero the following PER trap pointers: the SHDRPERJ, SHDRPER, SHDRPERA, and SHDRPERR fields.		
f. *Turn off space switch and PER monitoring in address spaces.* IEAVTGLB processes each ASCB pointed to in the ASVT as follows. To indicate that PER is deactivated in the address space, IEAVTGLB sets the PER bit (ASCBPER) to zero. If the ASCB's space switch event mask for SLIP is on (ASCBSSP=1), IEAVTGLB calls the space switch event mask manager (IEAVESSE) to clear the mask for SLIP. If the ASCB indicates that PER was activated in the corresponding address space (ASCBPERS=1), IEAVTGLB sets the ASCBPERS bit to zero. It then initializes and schedules a local SRB to enter IEAVTLCL. IEAVTLCL finds the ASCBPERS bit off and turns off PER monitoring.	IEAVTGLB	SCHEDSRB
A subroutine of IEAVTGLB (SCHEDSRB) issues a GETCELL to obtain storage for the SRB. If the cell is acquired, SCHEDSRB initializes an SRB parameter list and schedules IEAVTLCL to execute as a local SRB. (See the diagram and extended description for IEAVTLCL.)		
If the GETCELL fails, SCHEDSRB attempts to obtain a new extent, using a GETMAIN. If this fails and the PER trap is disabled, SCHEDSRB turns on the SHDRSRBR flag, indicating that this module is to be rescheduled. If the GETMAIN fails and the trap is enabled, SCHEDSRB indicates that IEECB905 is to be posted to issue message IEA742I. Processing continues at step 6.		
If the GETMAIN for a new extent is successful, SCHEDSRB issues a BLDCPOOL to build a cell pool. If this fails, IEAVTGLB abends with system code X'06E'. If the BLDCPOOL is successful, SCHEDSRB schedules IEAVTLCL.		



Extended Description	Module	Label
5 To activate PER monitoring in the system, IEAVTGLB:		
a. Replaces the CMSET entry point addresses in the SVT.		
b. Establishes PER monitoring in each active processor.		
c. Obtains SLIP storage for each active processor.		
d. Sets the model PSA PSW PER bits to one.		
e. Turns on space switch and PER monitoring in all of the required address spaces.		
Each step is explained below in greater detail.		
a. *Replaces the CMSET entry point addresses*. IEAVTGLB replaces the CMSET entry point addresses in the SVT with addresses of entry points into IEAVTSLC. The original CMSET entry addresses are saved in the SHDR. When PER monitoring is deactivated, IEAVTGLB restores the CMSET entry addresses.		
b. *Establishes PER monitoring globally*. IEAVTGLB initializes a parameter list with the PER mode values (SA or IF) found in the trap's SCEPFLG field, and the beginnning and ending addresses for PER monitoring (SCVAADD). (*Note:* SB PER traps are always set up initially in IF mode.) The module passes control to IEAVTSIG, which copies the parameter values into control registers 9-11, and sets the PER bit in the external, I/O, and SVC new PSWs to one. In a multiprocessing environment, IEAVTGLB does this for each active processor in the PCCAVT. For each active processor other than the one on which this module is executing, IEAVTGLB uses a RISGNL macro to pass control to IEAVTSIG. To activate PER monitoring in this module's processor, IEAVTGLB calls IEAVTSIG directly.		
c. *Obtains SLIP storage*. After a processor has been signalled, and before locating the next processor in the PCCAVT, IEAVTGLB issues a GETMAIN to obtain storage in the SQA for the SLIP work area (except when SLIP storage already exists, LCCASLIP≠0). IEAVTGLB puts the address of the work area in the LCCASLIP field.		
d. *Turns on the model PSA PSW PER bits*. If the model PSA exists and is page-fixed, IEAVTGLB sets the EXT, SVC, and I/O new model PSA PSW PER bits to one. If the model PSA is not page-fixed, IEAVTGLB indicates that IEECB905 is to be posted to issue message IEE4241.		
e. *Turns on space switch and PER monitoring in all required address space(s)*. If the PER trap was defined with an ASID list parameter (SHDRPERA≠0) and without MODE=HOME (SCEMHME=0), IEAVTGLB calls IEAVESSE to set the space switch event mask (ASCBSSSP) bit in each ASCB associated with each ASID in the list to one. IEAVTGLB also sets the PER bit	IEAVESSE	
(ASCBPER) to one when these conditions exist unless the trap was also defined with a JOBNAME parameter (SHDRPERJ≠0). If the trap was defined with a JOBNAME, IEAVTGLB only sets the PER bit (ASCBPER) to one when the jobname pointed to by either the ASCBJBNI or ASCBJBNS matches the jobname in the SCVA.		
To determine if PER monitoring is to be activated or deactivated in an address space, IEAVTGLB scans the ASVT, comparing the jobname fields and ASIDs of each ASCB entry with those specified on the enabled non-IGNORE PER trap. An address space is selected for PER monitoring if:	IEAVTGLB	
• The jobname pointed to by either the ASCBJBNI or ASCBJBNS field matches the jobname in the SCVAJND field of the enabled non-IGNORE PER trap (or there is no SCVA jobname entry, SHDPERJ=0), and,		
• MODE=HOME was specified on the trap and the ASCBASID field matches an ASID entry in the SCVAASD table of the enabled non-IGNORE PER trap (or there is no SCVA ASID entry, SHDRPERA=0), or,		
• MODE=HOME was not specified on the trap.		
If these conditions are not met, PER monitoring is to be off.		
IEAVTGLB compares the PER activation indicator of the address space (ASCBPERS) with the desired status determined above. If the two differ, IEAVTGLB adjusts the ASCBPERS flag to the desired status and schedules IEAVTLCL to execute as a local SRB. This processing is described in the previous step. (Recall that IEAVTLCL uses the ASCBPERS flag to determine whether to activate or deactivate PER monitoring). IEAVTLCL adjusts the old PSW PER bits in all RBs in the address space.	IEAVTGLB	SCHEDSRB
If private module PER monitoring is enabled but not active, IEAVTGLB schedules IEAVTLCL to execute as a local SRB. IEAVTLCL searches the local job pack area queue to find a matching private area module.		



IEAVTGLB – SLIP Global PER Activation/Deactivation Routine (Part 7 of 8)

IEAVTGLB – SLIP Global PER Activation/Deactivation Routine (Part 8 of 8)

Extended Description	Module	Label
6 IEAVTGLB releases the SHDR sequence word so IEECB905 can obtain it to process messages. If any messages have been requested in earlier processing, IEAVTGLB branch enters POST (IEA0PT02) to post IEECB905's ECB (SHDRECB). IEECB905 issues the messages. (See the diagram and extended description of IEECB905.)		
7 IEAVTGLB releases the CMS, SALLOC, and dispatcher locks, if held. If the model PSA is page-fixed, IEAVTGLB frees it. IEAVTGLB then releases the LOCAL lock, decreases the SLIP use counter (SHDRPFC) by one, and removes GLPERFRR from the FRR stack.		

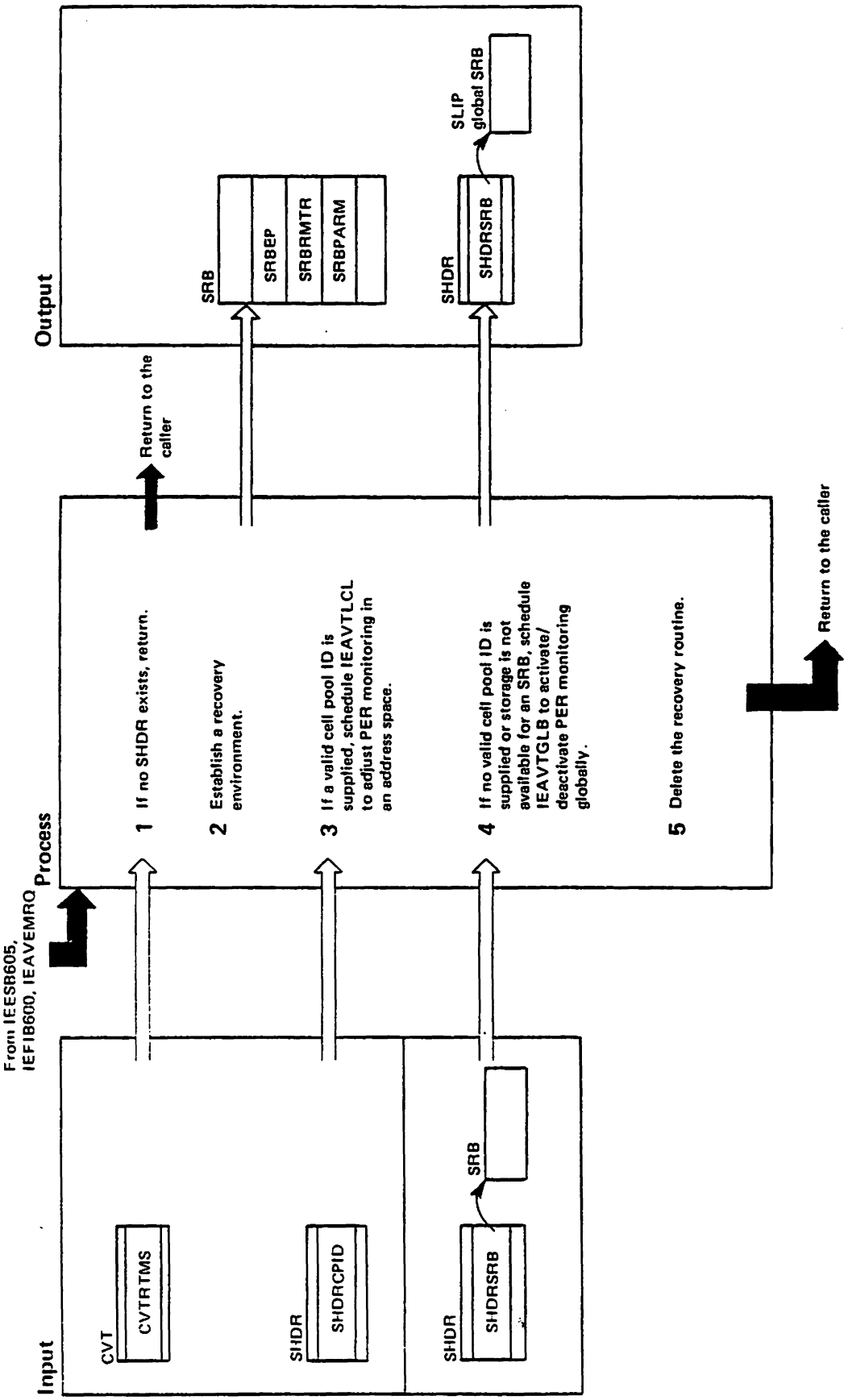
Recovery processing:

When a non-recursive error occurs while IEAVTGLB is executing, RTM gives GLPERFRR control, GLPERFRR:

- Indicates that the SRB is available (if necessary).
- Records the error in the SYS1.LOGREC data set and saves a retry address in the SDWARTYA field.
- If the SHDR sequence word is held, disables the enabled non-IGNORE PER trap, indicates that the SLIP command processor communications routine (IEECB905) is to be posted to issue message IEE743, and releases the SHDR sequence word. If the sequence word is not held, GLPERFRR indicates that message IEE415 is to be issued.
- Calls IEA0PT02 to post IEECB905's ECB.
- Releases the locks obtained by this FRR.
- Determines if a retry is allowed. If not (SDWACLUP=1), GLPERFRR requests percolation, page-frees the model PSA (if it was page-fixed), and decreases the SLIP use counter by one (if necessary).
- Sets the recursive error indicator.
- Returns to the dispatcher.

Extended Description	Module	Label
If a recursive error occurs, GLPERFRR: • Issues message IEA414I, using a RECORD macro, to notify the system operator of the recursive error. • Sets the enabled non-IGNORE PER trap pointer (SHDRPER) to zero. • Issues a SETRP macro to request that RTM free any locks currently held by IEAVTGLB. • Page-frees the model PSA (if it was page-fixed). • Releases the SHDR sequence word (if held). • Decreases the SLIP use counter by one (if necessary). • Percolates to RTM.		

IEAVTJBN - SLIP PER SELECT INTERFACE ROUTINE

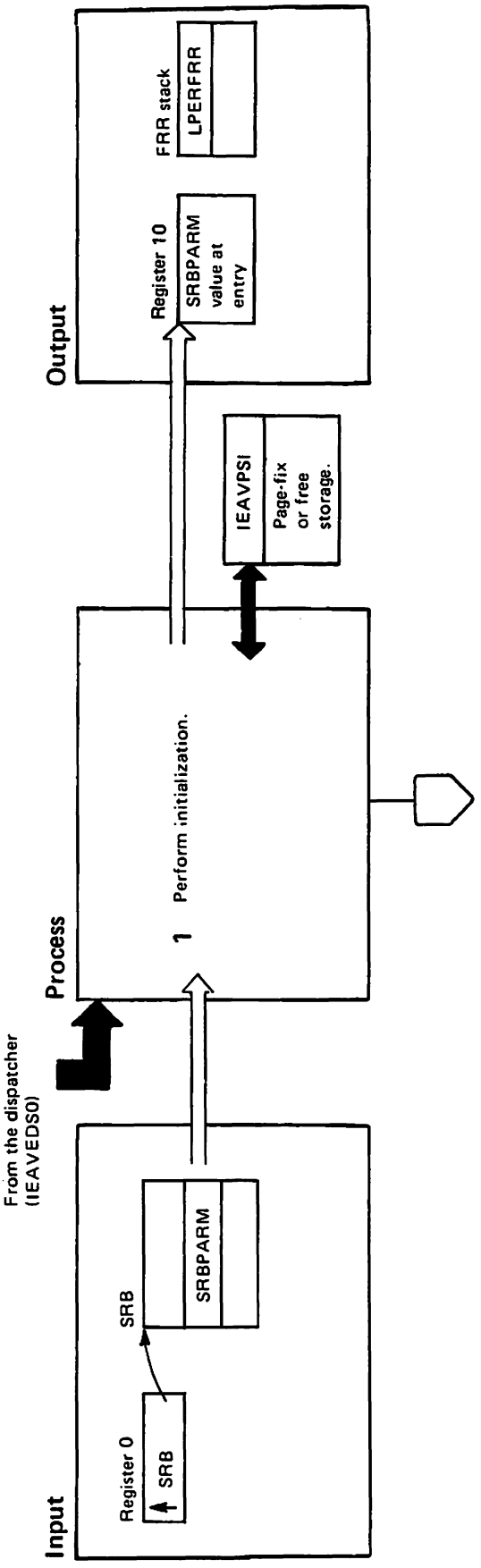


IEAVTJBN – SLIP PER Select Interface Routine (Part 1 of 2)

IEAVTJBN – SLIP PER Select Interface Routine (Part 2 of 2)

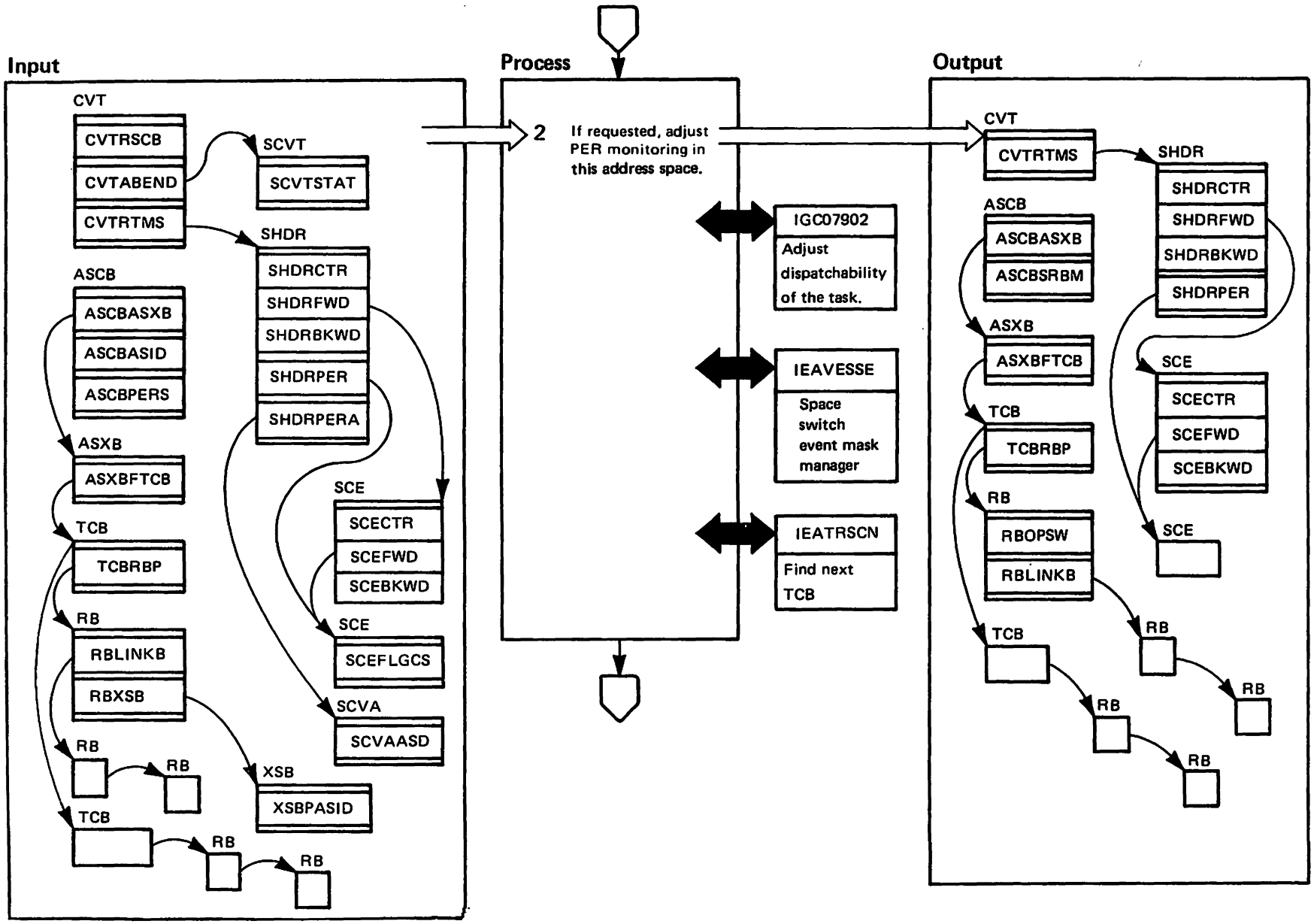
Extended Description	Module	Label
This module provides an interface between: • Address space create (IEAVEMRO) and the SLIP local PER activation/deactivation routine (IEAVTLCL) to determine if a newly created address space is to have PER monitoring active. • Job scheduler routines (IEESB605 and IEE1B600) and PER routines to determine if LOGON, START, or JOB SELECT commands require a change in the PER monitoring status of the address space.		
1 If an SHDR does not exist (CVTRTMS=0), IEAVTJBN returns to the caller.	IEAVTJBN	
2 IEAVTJBN obtains the LOCAL lock, then issues a SETFRR macro to establish entry point JBNFRR as its FRR.		
3 IEAVTJBN obtains the CMS lock. If the SLIP command processor (IEECB905) supplied a valid cell pool ID (SHDRCPID≠0), IEAVTJBN issues a GETCELL to obtain storage in the SQA for an SRB. The CMS lock is then released. If the GETCELL is successful, IEAVTJBN initializes an SRB parameter list with SRBPARM=1, indicating that full activation/deactivation of PER monitoring is requested, and with the SRBRMTR field pointing to an RMTR in IEAVTLCL. IEAVTJBN then schedules IEAVTLCL to execute as an SRB with local priority. IEAVTLCL turns PER monitoring on or off in the address space in which it is executing. Processing continues at step 5. If the GETCELL fails, or the cell pool ID is invalid, processing continues at the next step.		
4 IEAVTJBN uses a CS (compare and swap) instruction to acquire an SRB (pointed to by SHDRSRB), which the SLIP command processor initialized. If successful, IEAVTJBN schedules IEAVTGLB to execute as an SRB with global priority. If the CS instruction fails, IEAVTGLB is already scheduled and will make any PER monitoring changes necessary.	IEAVTJBN	
5 IEAVTJBN deletes the FRR, releases the LOCAL lock, and returns to the caller.		
Recovery processing:		
JBNFRR records the error in the SYS1.LOGREC data set. If the SHDR exists, JBNFRR sets the IEA422I message flag and posts IEECB905 to issue the message. JBNFRR then issues an SDUMP. If IEAVTJBN obtained the CMS lock, JBNFRR requests that RTM free it. If retry is not allowed (SDWACLUP=1), JBNFRR requests that RTM also free the LOCAL lock. If retry is allowed, JBNFRR indicates that RTM is to retry at RETRYADR in IEAVTJBN, where registers are restored and control is returned to the caller. JBNFRR returns control to RTM.	JBNFRR	

IEAVTLCL - SLIP LOCAL PER ACTIVATION/DEACTIVATION ROUTINE



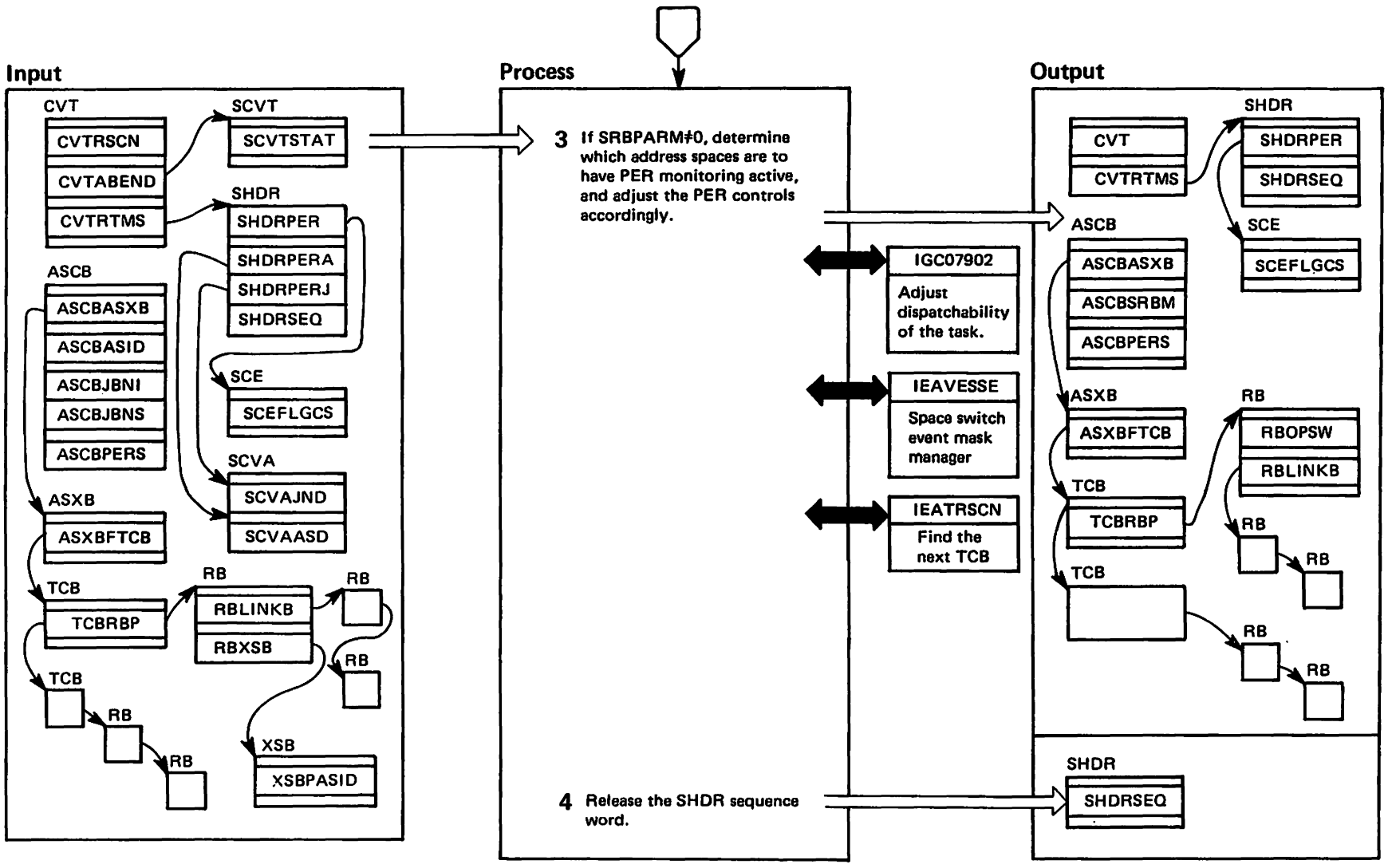
IEAVTLCL – SLIP Local PER Activation/Deactivation Routine (Part 1 of 10)

Extended Description	Module	Label
IEAVTLCL receives control as the result of an SRB routine scheduled with local priority by the SLIP global PER activation/deactivation routine (IEAVTGLB) or the SLIP PER select interface routine (IEAVTJBN). IEAVTLCL's function is to turn PER monitoring on or off in the address space in which it is executing and to search the local job pack area queue for a private area module that matches the current enabled PER trap.		
1 IEAVTLCL performs the following initialization functions. • Saves the contents of the SRBPARM field. • Provides recovery by adding LPERFRR to the FRR stack. • Obtains the LOCAL lock to allow a branch entry to page-fix, page-free, and STATUS in later processing. • Obtains the CMS lock to serialize GETCELL/FREECELL. • Frees the SRB storage. To do so, IEAVTLCL obtains the SLIP cell pool ID from the SHDRCPID field and issues a FREECELL. IEAVTLCL releases the CMS lock. If the freed cell was the last of an extent in which no other cells are currently allocated (FREECELL's return code=20), IEAVTLCL to page-fixes this segment of the module, obtains the SALLOC lock, then issues a FREEMAIN to free the extent. IEAVTLCL releases the SALLOC lock and page-frees its fixed pages. If the FREECELL fails and the cell belongs to an extent from the pool, IEAVTLCL issues an ABEND with code X'06E'.	IEAVTLCL	

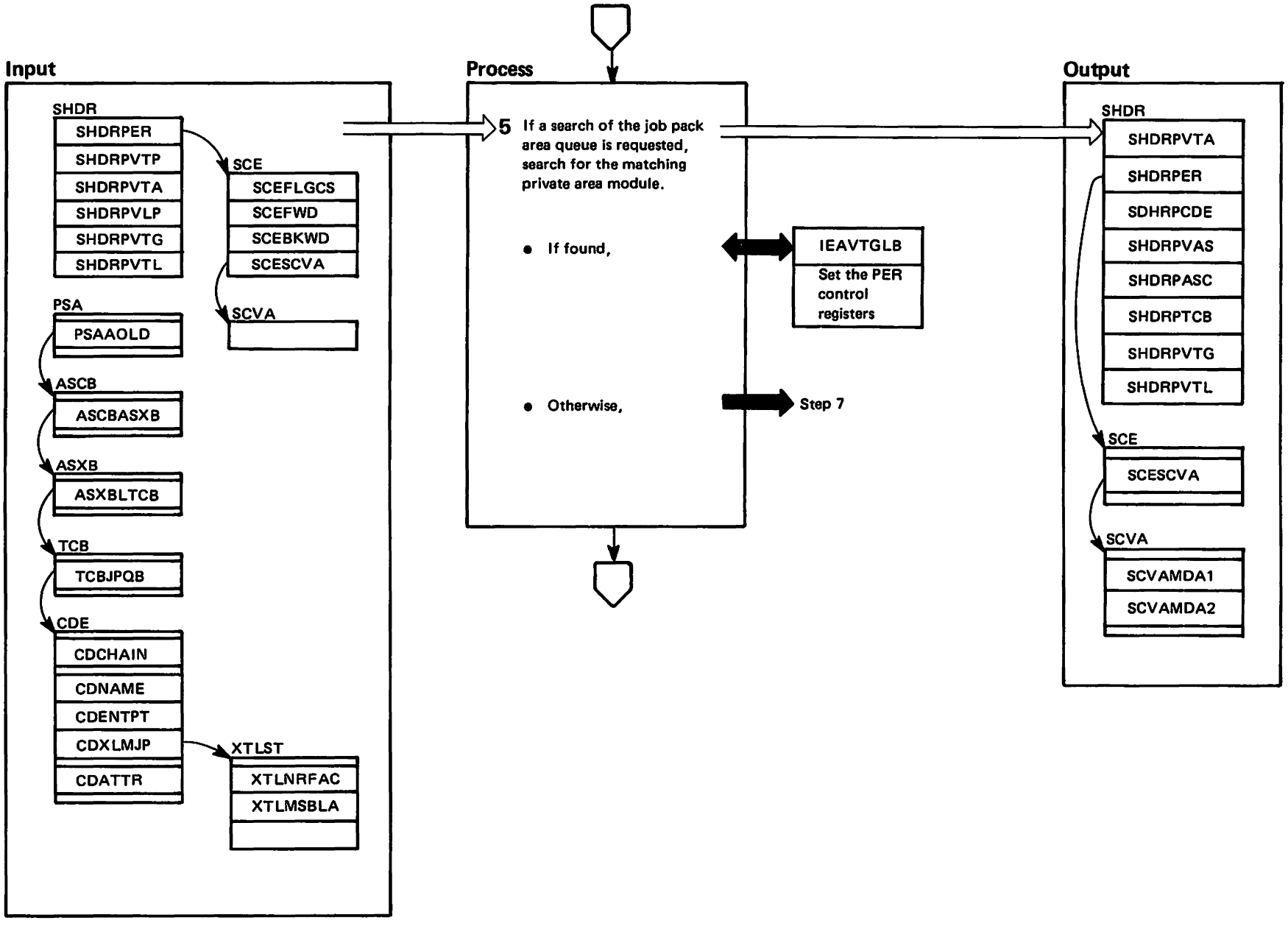


Extended Description	Module	Label
2 If the SRBPARM value at entry was zero, IEAVTGLB has already determined which address spaces should have PER monitoring active, and has adjusted the ASCBPERS bit accordingly. IEAVTLCL calls the internal subroutine TCBRBSCN to adjust PER monitoring in each request block in the address space.		
TCBRBSCN first branch enters the STATUS routine (IGC07902). STATUS sets all tasks in the address space non-dispatchable by turning on the SLIP secondary non-dispatchability flag in each TCB. (This prevents alteration of the TCB/RB chain while it is being scanned.) If the ASCBPERS bit indicates to turn PER off, TCBRBSCN locates the first TCB and, in each request block attached to it, sets the RBOPSW PER bit to zero. TCBRBSCN calls IEATRSCN to locate the next TCB on the TCB chain, and repeats the process until all TCBs have been located and the RBOPSW bits in their associated RBs have been adjusted.	IGC07902 IEAVTLCL IEATRSCN	TCBRBSCN
If the ASCBPERS bit indicates to turn PER on, TCBRBSCN calls the SCECTR routine to serialize the SCE chain up to the enabled non-IGNORE PER trap. TCBRBSCN locates the first TCB and, for each request block attached to it, does the following: • If MODE=HOME was requested on the enabled non-IGNORE PER trap, TCBRBSCN compares the primary address space (XSBPASID) of that request block with the home (ASCBASID). If they match, TCBRBSCN sets the RBOPSW bit to one. • If MODE=HOME was not requested on the enabled non-IGNORE PER trap and the ASID parameter was specified, TCBRBSCN compares the primary address space (XSBPASID) of that request block with each ASID in the ASID entry of the SCVA and if one matches, TCBRBSCN sets the RBOPSW PER bit to one. • If MODE=HOME was not requested and the ASID parameter was not specified, TCBRBSCN sets the RBOPSW PER bit to one in every request block.		

Extended Description	Module	Label
TCBRBSCN calls IEATRSCN to locate the next TCB on the TCB chain, and repeats this process until all TCBs have been located and their associated RBs have been checked.	IEATRSCN	
If MODE=HOME was requested or the ASID parameter was not specified on the enabled non-IGNORE PER trap, TCBRBSCN sets the ASCBPER bit in the ASCBSRBM field to one and calls IEAVESSE, the space switch event mask manager, to set the space switch event mask on for SLIP.	IEAVTLCL IEAVESSE	
TCBRBSCN calls STATUS to reset all tasks in the address space dispatchable.		

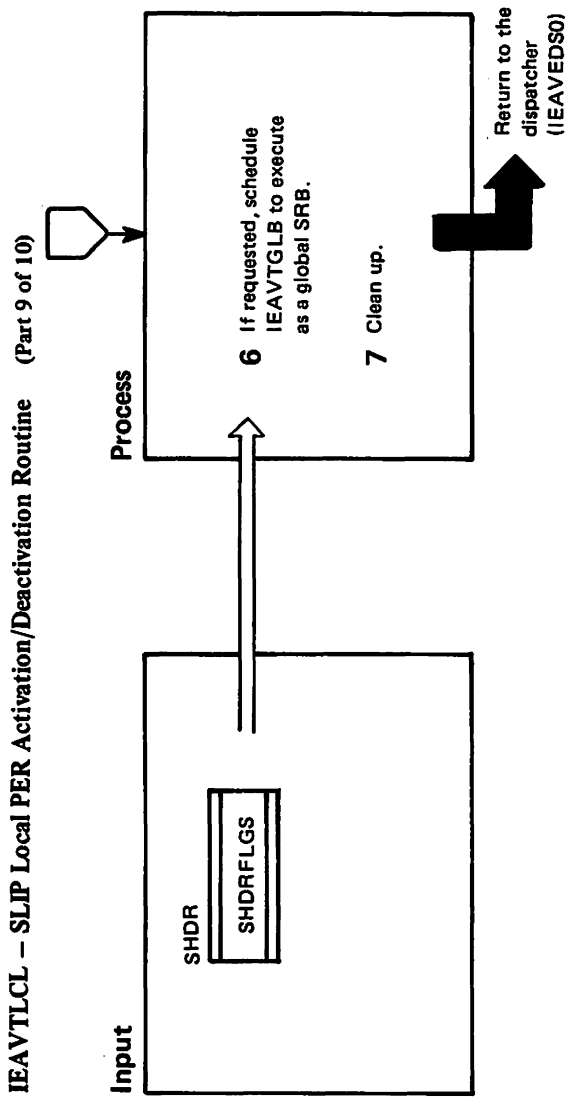


Extended Description	Module	Label
3 If SRBPARM≠0 at entry (the caller is IEAVTJBN), IEAVTLCL obtains the SHDR sequence word (SHDRSEQ). If it is unavailable, IEAVTLCL continues processing at step 6, where IEAVTGLB is scheduled. If the PER trap does not exist (SHDRPER=0) or it is disabled (SCEDSABL=1), IEAVTLCL continues processing at step 4.	IEAVTLCL	
If the PER trap indicates MODE=HOME was not requested (SCEMHME=0) and there is an ASID entry in the SCVA (SHDRPERA≠0) which matches the ASID of this address space (ASCBASID), IEAVTLCL:		
• Calls IEAVESSE to turn on the space switch event mask for SLIP in this address space.	IEAVESSE	
• If the jobname pointed to by either ASCBJBNI or ASCBJBNS matches the jobname in the SCVA of the PER trap (or there is no jobname entry, SCHRPERJ=0), turns on the PER indicator (ASCBPER).		
• Determines whether PER monitoring should be active in this address space. This is done by comparing the ASCB of this address space with the enabled non-IGNORE PER trap. PER monitoring is to be active in this address space if:		
– The jobname pointed to by either the ASCBJBNI or ASCBJBNS field matches the jobname in the trap (or there is no jobname in the trap, SHDRPERJ=0), and		
– Either MODE=HOME was requested (SCEMHME=1) and the ASCBASID value matches one of the ASIDs listed in the trap (or there are no ASIDs listed in the trap, SHDRPERA=0), or MODE=HOME was not requested (SCEMHME=0)		
If PER monitoring is not already active (ASCBPERS=0), IEAVTLCL sets the ASCBPERS bit to one and calls the internal subroutine TCBRBSCN to adjust PER monitoring in each request block as described in step 2.		
If any of the above conditions are not met, PER monitoring is to be off. IEAVTLCL sets the PER indicator (ASCBPER) to zero. If PER monitoring is active (ASCBPERS=1), IEAVTLCL also sets the ASCBPERS bit to zero and calls the internal subroutine TCBRBSCN to adjust PER monitoring in each request block as described in step 2.		
4 After adjusting PER monitoring, IEAVTLCL releases the SHDR sequence word.		



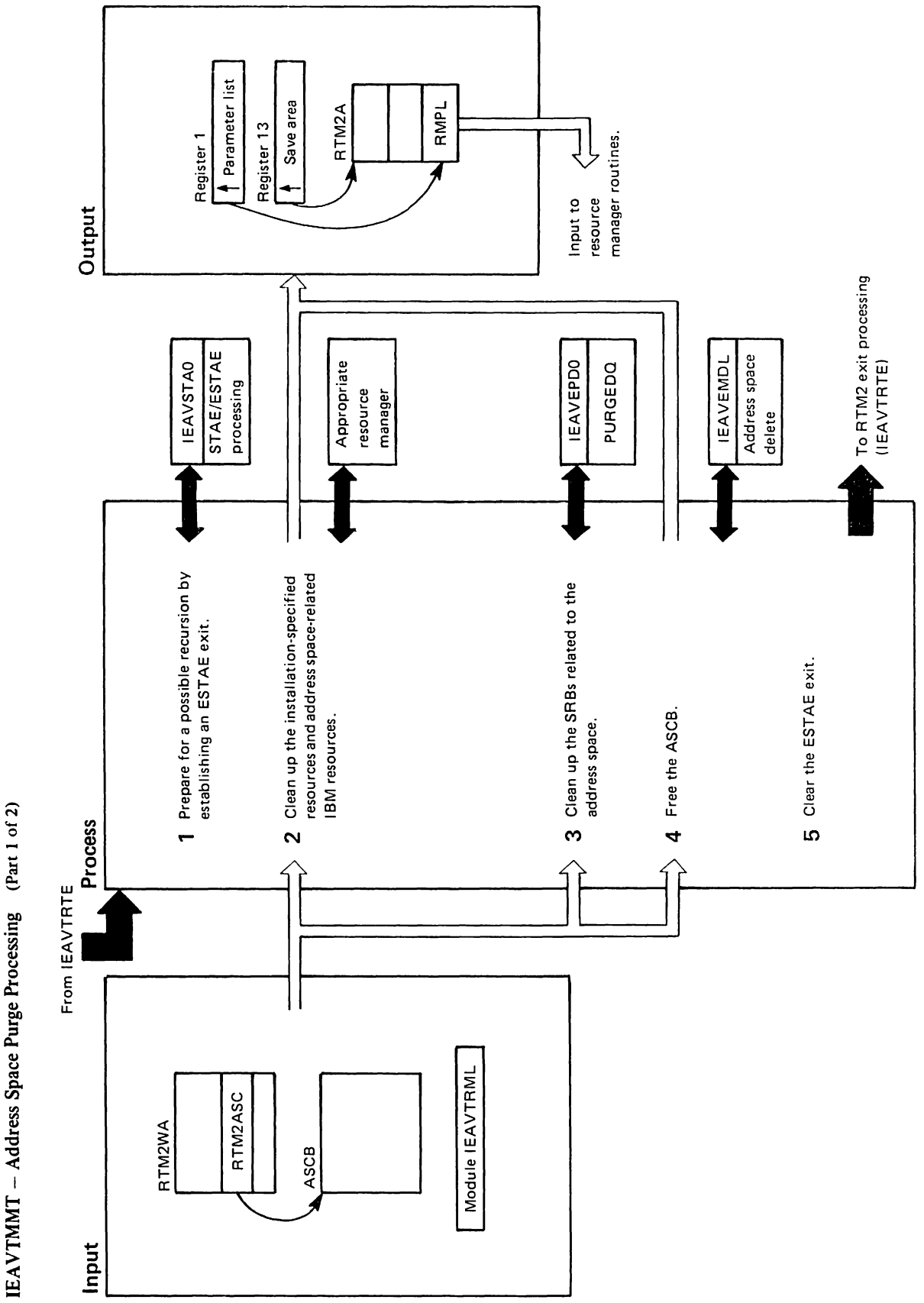
IEAVTLCL – SLIP Local PER Activation/Deactivation Routine (Part 8 of 10)

Extended Description	Module	Label
5 IEAVTGLB indicates to IEAVTLCL (by passing a 1 in bit position 30 (x'00000002') in the SRBPARM word) that a search of the job pack area queue (JPAQ) must be performed. IEAVTLCL's search attempts to find a private area module that matches the module specified in the enabled non-ignore PER trap.		
To prevent IEECB905 from deleting any of the traps being examined, SEARCHJP processing serializes the SCEs by incrementing the use counts in each SCE. SEARCHJP then examines each CDE in the JPAQ for each TCB in this address space. If the module name in the CDE matches that in the trap, SEARCHJP does further checking. SEARCHJP distinguishes between modules loaded locally and modules loaded globally (indicated by the CDGLOBAL bit).		SEARCHJP
Having found a match, SEARCHJP obtains the dispatcher lock. SEARCHJP processing determines if the module start offset, which is specified in the trap, lies within the module found. If so, SEARCHJP uses this module's CDE. Otherwise, SEARCHJP continues searching. For a matching module, SEARCHJP sets up the actual module addresses to be monitored (prior to this, only the module offsets were available) and saves information needed for IEAVTPVT's processing. IEAVTLCL then indicates that IEAVTGLB must be rescheduled to set the PER control registers, having determined the PER range to be monitored. After releasing the dispatcher lock, SEARCHJP releases the serialization of the SCEs by decrementing their use counts. If the module name in any CDE does not match that in the trap, continue processing at step 7.	IEAVELK	



Extended Description	Module	Label
6 If there is a request to have IEAVTGLB scheduled (the SHDRSRBR flag is set), IEAVTLCL tries to obtain the global SRB using a CS (compare and swap) instruction and the SHDRSRB field. If successful, IEAVTLCL schedules IEAVTGLB to run as a global SRB in the master address space. If the SRB is unavailable, processing continues at the next step.		
7 IEAVTLCL releases the local lock, deletes LPERFRR from the FRR stack, and returns to the dispatcher.		
Recovery Processing:		
If an error occurs while IEAVTLCL is executing, LPERFRR receives control to:		LPERFRR
• Record the error in the SYS1.LOGREC data set.		
• Set tasks in the address space dispatchable (if IEAVTLCL set them non-dispatchable) by obtaining the LOCAL lock, calling IEAVPSI to page fix this segment of the module, obtaining the SALLOC lock, and calling STATUS (IGC07902) to turn off the SLIP secondary nondispatchability flag in each TCB.		
• Issue an SDUMP.		
• Release the SALLOC lock (if obtained by the FRR), page free this segment of the module if it was fixed, and release the SHDR sequence word (if held).		
• Decrease the use counts in the SCEs and the SHDR if they have not already been decreased by IEAVTLCL.		
• Post the SLIP command processor communications routine (IEECB905). IEECB905 issues message IEA415I to the SLIP trap user, indicating an ABEND occurred while a PER request was being processed.		
• Release the LOCAL lock if it was obtained by the FRR.		
• If the LOCAL, CMS, or SALLOC lock is held, request that RTM release it.		
• Return to RTM with a return code of zero (request percolation).		
If IEAVTLCL has been scheduled but not yet dispatched when its address space terminates, it receives control at entry point LPERRMTR as the result of a PURGEDQ function issued by RTM2. IEAVTLCL obtains the LOCAL and CMS locks, issues a FREECELL to free SRB storage, then releases the LOCAL and CMS locks. If the freed cell is the last of an extent (the FREECELL return code=20), IEAVTLCL attempts to free the extent using a FREEMAIN macro. If an error occurred while FREECELL was executing (the return code≠0 or 20), and the cell belongs to an extent from the cell pool (return code≠8), IEAVTLCL issues an ABEND with system code X'06E', and returns to RTM. Otherwise; IEAVTLCL returns to the PURGEDQ function.		LPERRTMR

IEAVTMMT - ADDRESS SPACE PURGE PROCESSING



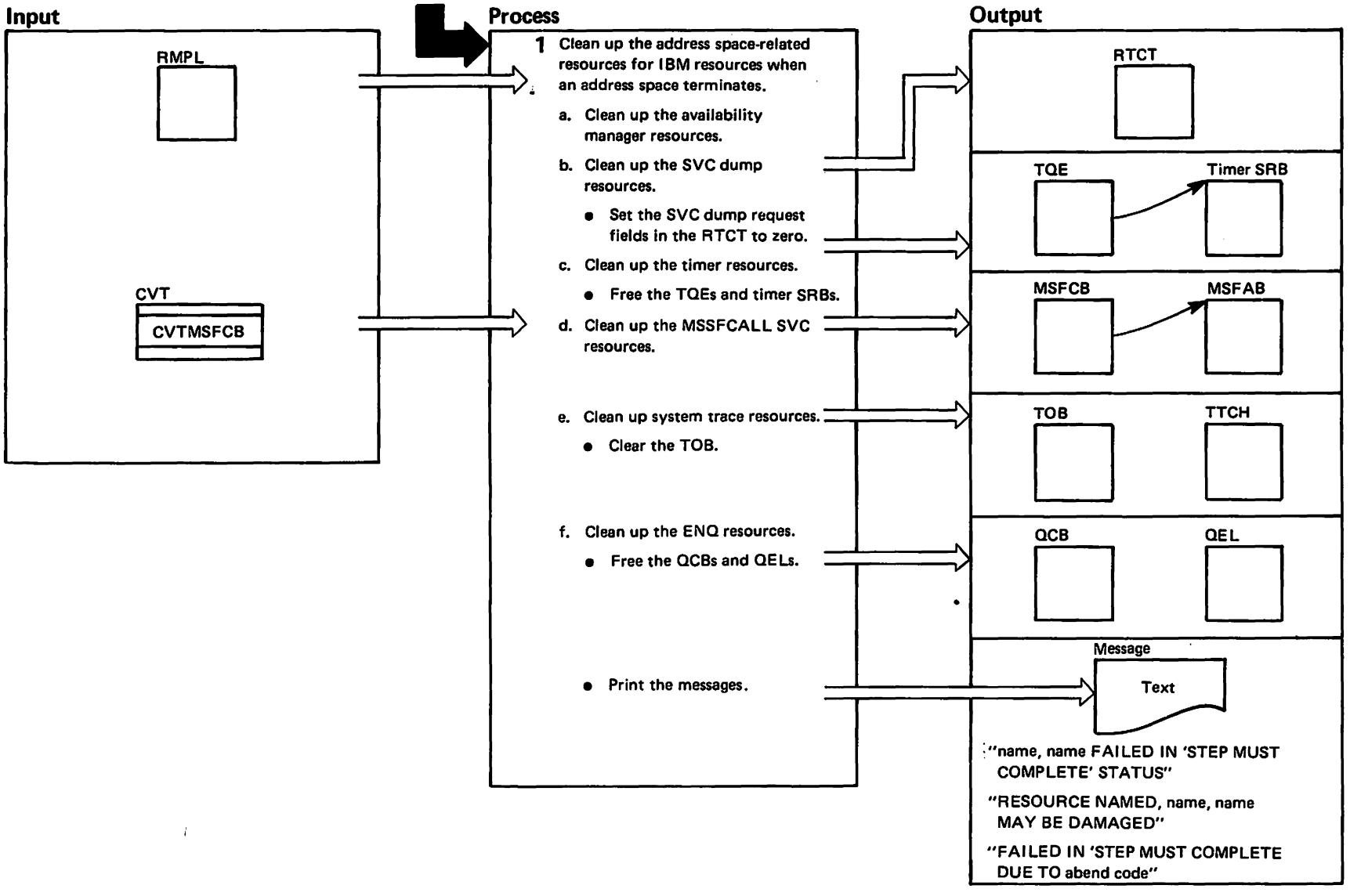
IEAVTMMT – Address Space Purge Processing (Part 1 of 2)

IEAVTMMT – Address Space Purge Processing (Part 2 of 2)

Extended Description	Module	Label
The address space purge function cleans up the address space resources when it terminates. Control initially goes to the RTM1 mainline code (see the M.O. diagram IEAVTRT2 RTM1 Overview) to service a CALLRTM=MEMTERM request. RTM1 then schedules the address space termination routines (see the M.O. diagram IEAVTMTC – Address Space Termination Processing) to terminate the address space. The final process in address space termination occurs when RTM2 receives a request from the address space termination routines to purge the resources from the address space.		
Address space purge processing uses the RTM2WA initialized by initialization processing (see the M.O. diagram IEAVTRT2 – RTM2 Initialization) for the basic input, along with the address of the ASCB being purged.		
The address space purge processing routine only honors requests from the master address space. Requesters from any other address space will be terminated.		
1 Address space purge processing establishes an ESTAE exit in case of failure.	IEAVTMMT IEAVSTA0	
2 Address space purge processing cleans up address space resources by first giving control to installation-defined subsystem cleanup routines (defined in module IEAVTRML) to clean up any subsystem resources. These subsystem cleanup routines will receive control sequentially until they have all executed. Control next passes to the IBM-defined resource managers, which clean up system control program routines. The resource managers receive control in the addressing mode indicated in the address field, in the following order:	IEFJRECM	
Availability manager	AVFMHTRM	
SVC dump	IEAVTSDR	
Timer	IEAVTRTI1	
MSSFCALL SVC	IEAVMFRM	
System trace	IEAVETRM	
ENQ/DEQ	IEAVENQ2	
Data management	IEG01C0A	
VTAM (virtual telecommunications access method)	ISTRAMA2	
TCAM (telecommunications access method)	IEDQOT01	
TIOC (terminal input/output coordinator)	IEDAY8	
VTIOC (VTAM terminal input/output coordinator TSO/VTAM)		
WTOR (write-to-operator with reply)	IEAVMED2	
Subsystem interface	IEFJRECM	
Initiator	IEFIRECM	
Scheduler allocation	IEFAB4E5	
Contents supervisor	IEAVLK02	
Virtual fetch	CSVVFMEM	
Linklist lookaside	CSVLLTRM	
PCAUTH	IEAVXPAM	
POST	IEAVEPST	
Virtual storage management	IEAVGFAS	
Lock management	IEAVELRM	
OLTEP (on-line test executive program)	IFDOLT0A	
IDMS	ICB2AIR	
RTM1	IEAVTMRM	
Type 1 message	IEAVTPMT	
SMF	IEASMFSP	
ASCB delete	IEAVEMDL	
The diagrams for the SPIE and RCT resource managers show the modules that perform the clean up, and the control blocks that are cleared.		
3 Control goes to the PURGEDQ routines (see the M.O. diagram IEAVEPD0 – PURGEDQ Processing) to remove any SRBs left in the address space. No more SRBs will be queued since IEAVTMMT sets the ASCB acronym to zero before passing control to PURGEDQ.		
4 Address space purge processing gives control to address space-delete to free any non-permanent address spaces (ASID $>$ 1 in the ASCB). Address space purge processing does not free the address space if:	IEAVEMDL	
• ASID = 0 - system wait task • ASID = 1 - master scheduler		
Address space purge processing clears the ESTAE routine and gives control to the caller (module IEAVTRTE).	IEAVTMMT	

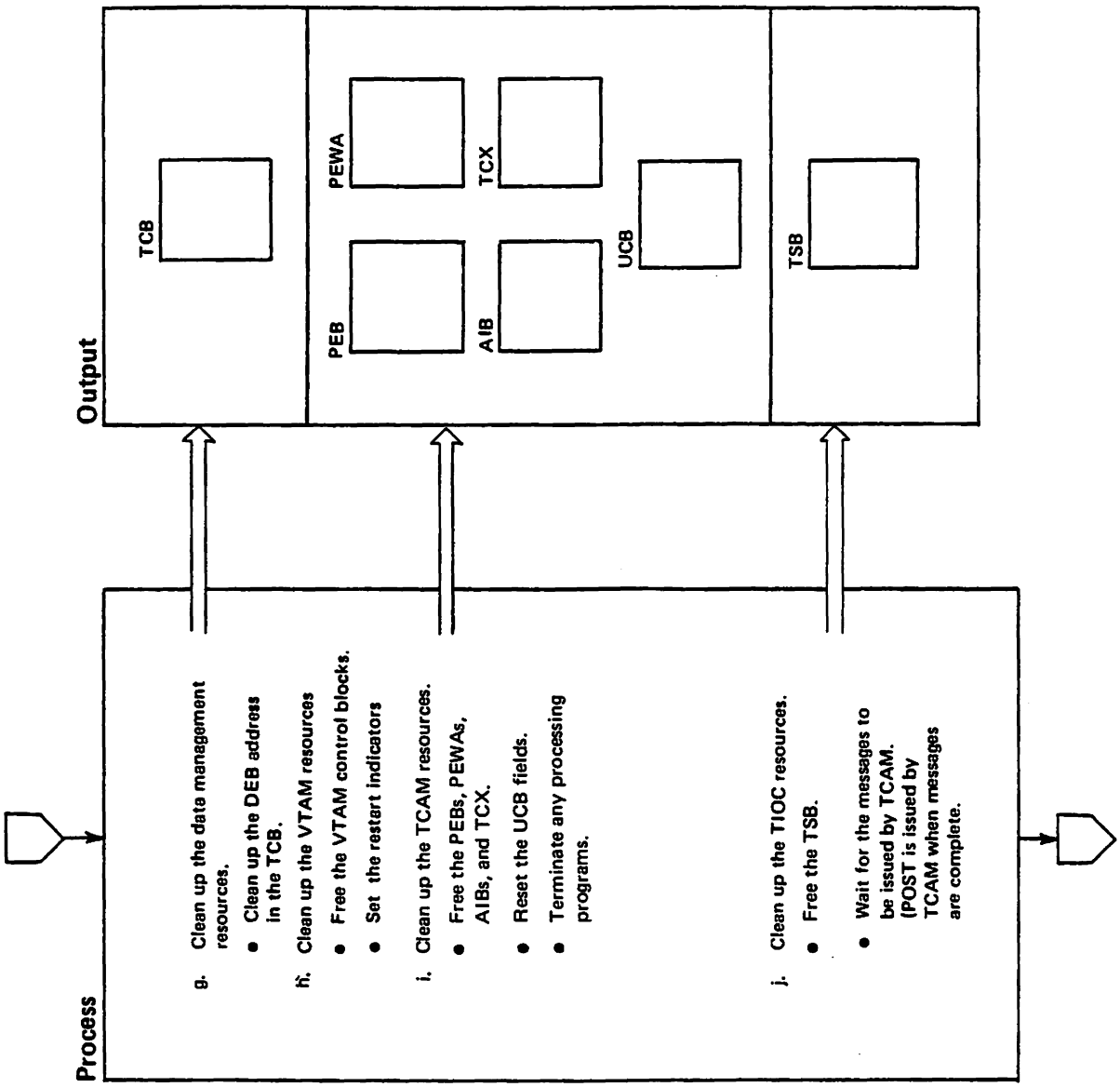
IEAVTMMT – Address Space Purge Resource Managers (Part 1 of 10)

From address space purge processing (IEAVTMMT) to clean up address space-related resources when an address space terminates



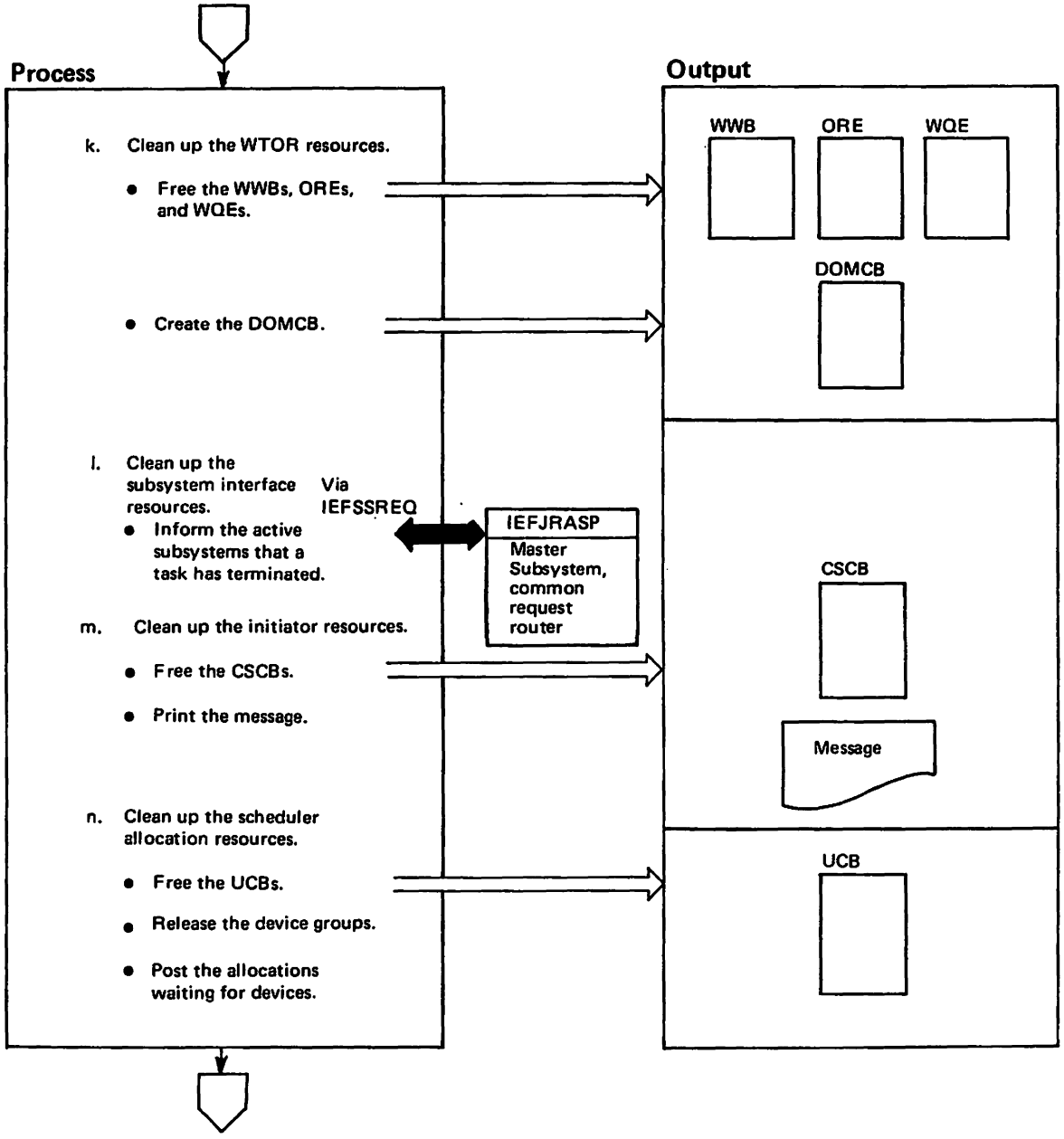
IEAVTMMT – Address Space Purge Resource Managers (Part 2 of 10)

Extended Description	Module	Label
The IBM-defined address space cleanup resource managers free any resources held by an address space during processing. The address space purge processing routine, module IEAVTMMT, routes control to these resource managers after establishing an interface. Control goes to each address space resource manager, in the appropriate addressing mode until all of them have performed their cleanup processing.		
1 The address space purge routine routes control to each of the IBM-defined resource managers. After one resource manager completes its processing, control returns to the address space purge routine, which routes control to the next resource manager. This continues until all the resource managers have performed clean up.	IEAVTMMT	
a. The AVM resource manager does one of the following: • Starts takeover. • Cleans up availability manager data areas. • If no action necessary, does nothing.	AVFMHTRM	
b. The SVC dump resource manager issues STATUS to set the system dispatchable if a dump was in progress in the failing address space.	IEAVTSDR	
c. The timer resource manager frees the TQEs (timer queue elements) and timer SRBs associated with the address space being terminated. (See the M.O. diagram IEAVRTI1 – Timer Supervision in the section "Timer" for a description of the timer purge routine.)	IEAVRTI1	
d. The MSSFCALL SVC resource manager dequeues the MSSFCALL control blocks.	IEAVMFRM	
e. If the terminating address space is not the trace address space, the system trace resource manager removes all trace table copy headers (TTCH) for the terminating address space from the TTCH queue and frees them. If the terminating address space is the trace address space, the system trace resource manager clears the trace option block (TOB) and notifies the operator that the trace address space has terminated. (See M.O. diagram IEAVETRM in the section "Trace" for a description of the system trace resource manager.)	IEAVETRM	
f. The ENQ resource manager frees associated ENQ resources used by the terminating address space by freeing QCBs (queue control blocks) and QELs (queue elements). The ENQ resource manager also writes messages explaining which address space failed while it controlled the resource. (See the section "Global Resource Serialization" for a detailed description of ENQ processing.)	IEAVENQ2	



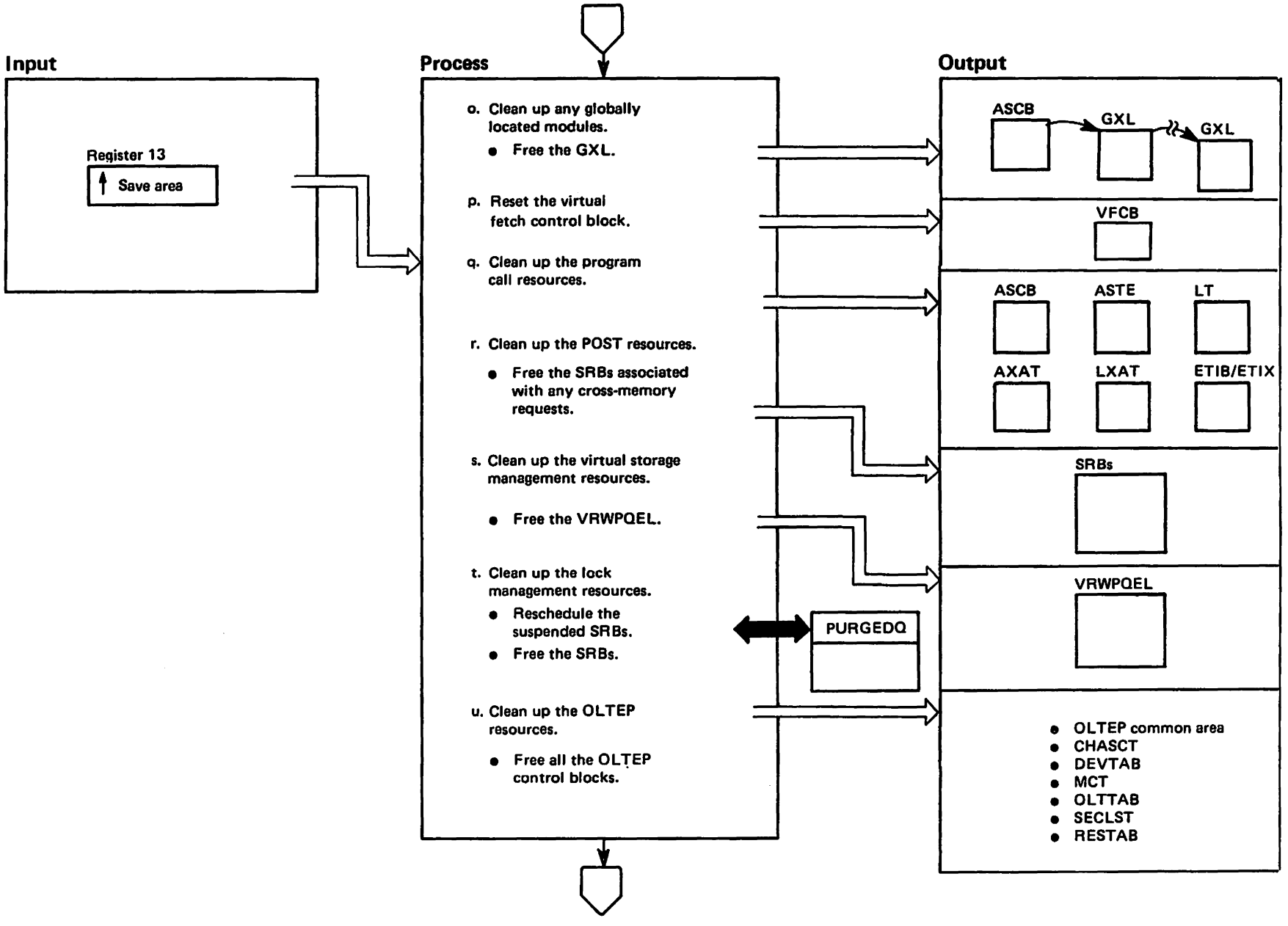
IEAVTMMT – Address Space Purge Resource Managers (Part 3 of 10)

Extended Description	Module	Label
g. The data management resource manager cleans up the TCBDEBAD field of the TCB. This field contains the DEB address from the DCB. (See the publication *Open/Close/EOV Logic* for more detailed information about the data management resource manager.)	IEG01C0A	
h. The VTAM resource manager cleans up resources associated with the VTAM user address space. These resources include storage, VTAM locks, and control blocks associated with the VTAM devices and applications which were active for this address space. The user's address space control blocks consist of: • Active CRAs (component recovery area) • DEBs (data extent block) • FMCBs (function management control block) • NCBs (node control block) • ICEs (inactive connection element) • ACEs (active connection element) • DCEs (DEB chain element) • PST (process scheduling table) • Application RDTEs (resource definition table) • Destination RDTEs • DVTs (destination vector table) • EPTs (entry point table) • MPSTs (memory process scheduling table) VTAM's address space control blocks consist of: • AVT (VTAM address vector table) • ATCVT (VTAM communications vector table) • ISTCONFT (configuration table) • CVT (communications vector table) If the terminated address space is VTAM's, appropriate indicators in the CVT are reset to zero to allow VTAM to be restarted. (These indicators are the CVTATCVT, the CVTRMPTT, and the CVTRMPMT.) (See the publication *VTAM Logic* for a description of VTAM processing.)	ISTRAMA2	
i. The TCAM (telecommunications access method) resource manager frees the resources associated with the terminating address space by freeing the PEBs (process extension blocks), PEWAs (process entry work areas), AIBs (application interface blocks), and TCX (TCAM CVT extension), and it resets UCB (unit control block) fields. (See the publication *TCAM Logic* for a description of the TCAM resource manager.)	IEDQOT01	
j. The TIOC (terminal input/output coordinator) resource manager cleans up the TSB (terminal status block) for the address space being terminated.	IEDAY8	



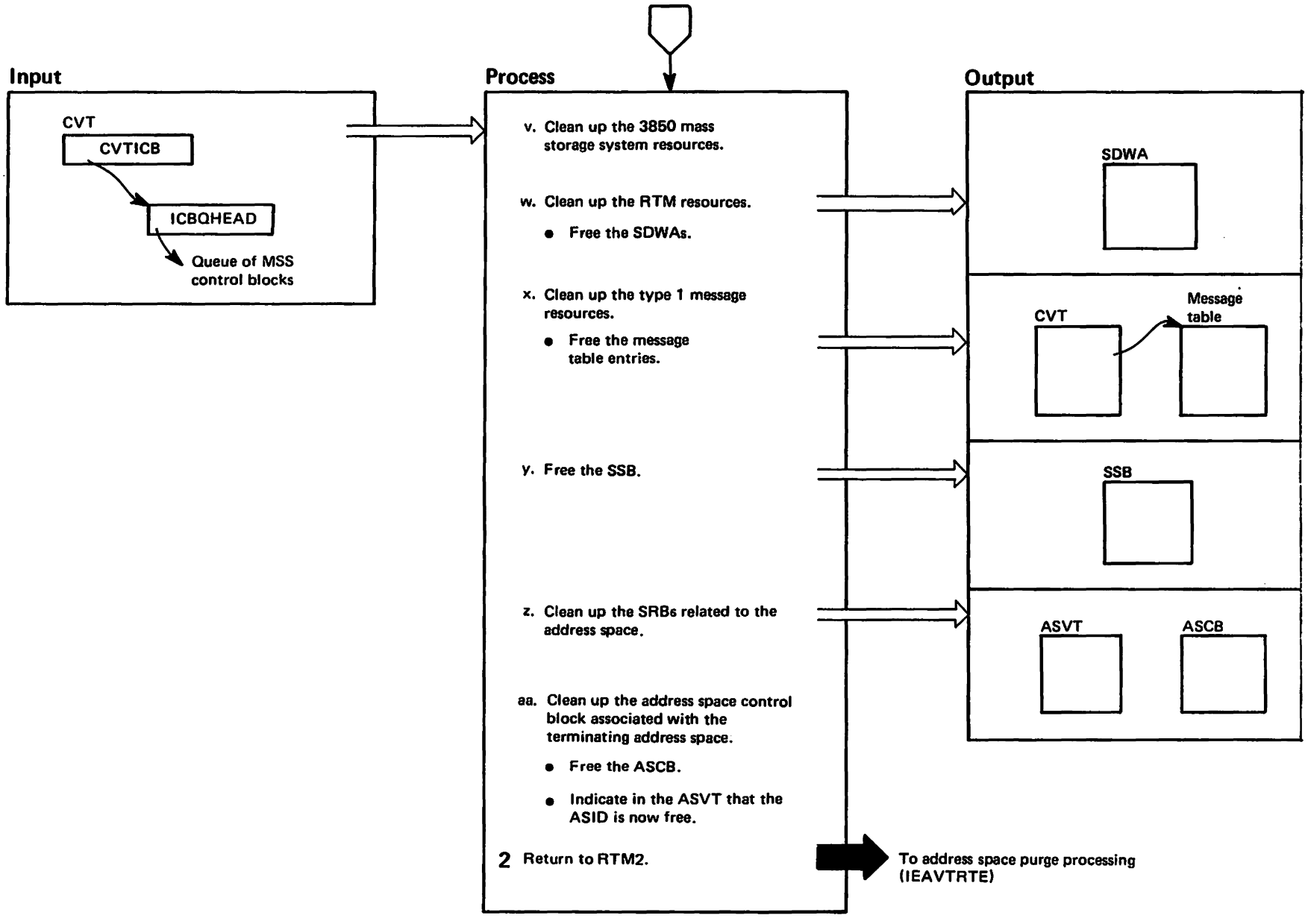
IEAVTMMT – Address Space Purge Resource Managers (Part 6 of 10)

Extended Description	Module	Label
k. The communications task resource manager cleans up WTOR (write to operator with reply) resources associated with the address space being terminated, by freeing the WWBs (write wait blocks), OREs (operator reply elements), WQEs (write queue elements), and DOMCs (delete operator message control blocks.)	IEAVMED2	
l. The subsystem interface resource manager cleans up the resources associated with the failing address space by notifying the active subsystems, via the IEFSSREQ macro, of the address space that terminated.	IEFJRECM IEFJRASP	
m. The initiator resource manager cleans up the resources associated with the address space being terminated by freeing the CSCBs (command scheduling control blocks). The resource manager also prints a message to the operator indicating which tasks in the address space are being terminated.	IEFIRECM	
n. The allocation resource manager cleans up the resources associated with the address space being terminated by unallocating the UCBs (unit control blocks). Additionally, the resource manager releases the device groups for the allocation, and then posts allocations waiting for those devices. (See the section "Allocation/ Unallocation" for a description of allocation and unallocation processing.)	IEFAB4E5	



IEAVTMMT – Address Space Purge Resource Managers (Part 8 of 10)

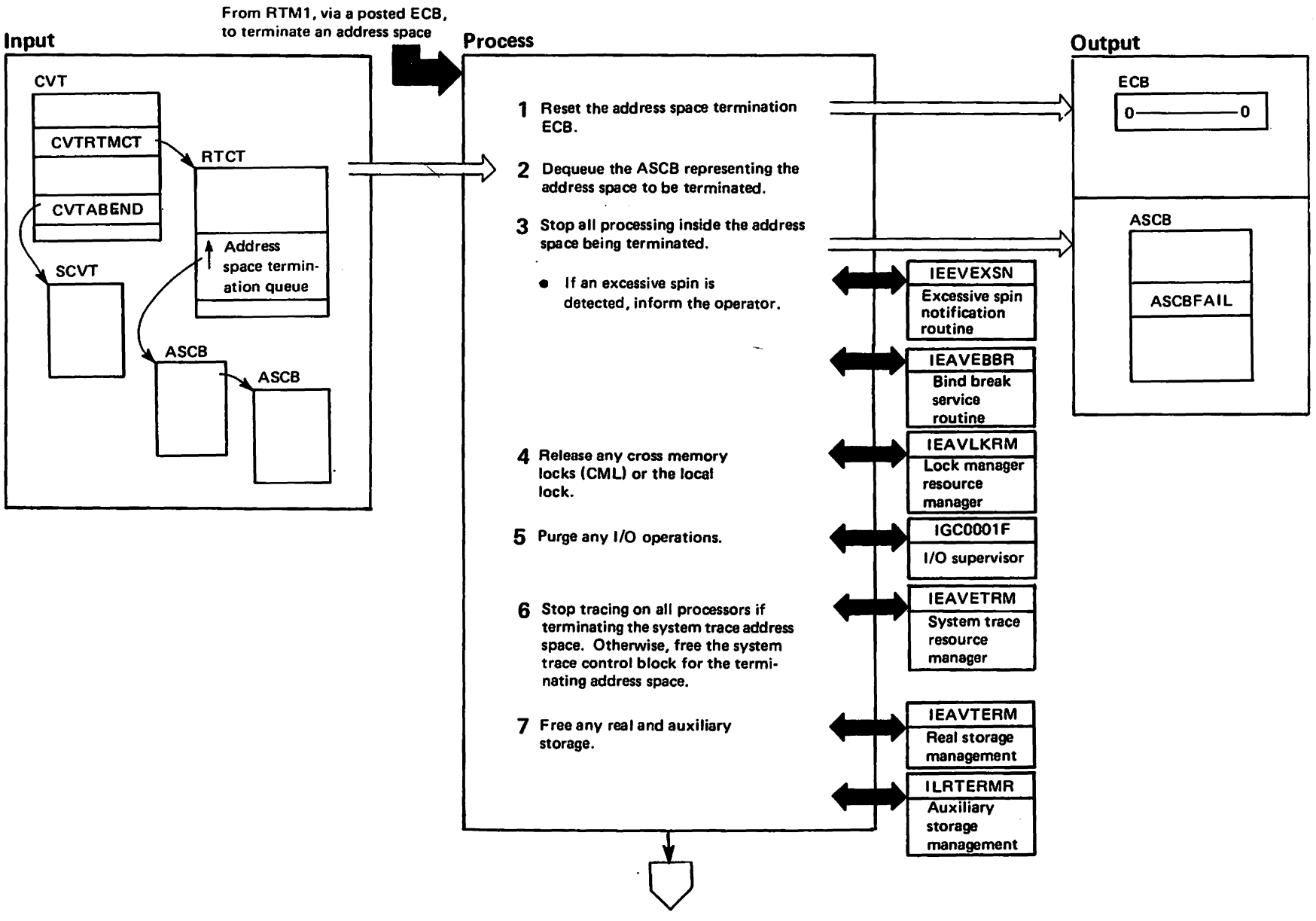
Extended Description	Module	Label
o. The contents supervisor program manager frees the globally located modules.	IEAVLK00	GXLHKEEP
p. The virtual fetch service address space termination resource manager resets the virtual fetch control block (VFCB) to indicate that the virtual fetch service address space is not active.	CSVVFMEM	
q. An inline macro (PCARM) gives the program call authorization resource manager control. This resource manager cleans up the program call resources.	IEAVXPAM	
r. The POST resource manager cleans up the resources associated with the address space being terminated by freeing the SRB associated with any cross-memory POST requests. (The M.O. diagram for IEAVEPST describes POST processing.)	IEAVEPST	
s. The virtual storage management resource manager cleans up resources associated with the address space by freeing the VRWPQEL (virtual equals real wait or post queue element). (See the section "Virtual Storage Management" for a complete description of the resource manager.)	IEAVGFAS	
t. The lock management resource manager cleans up resources associated with the address space being terminated by scheduling suspended SRBs. These SRBs will be freed after they complete their processing. (The M.O. diagram IEAVELK – SETLOCK Processing describes SETLOCK processing.)	IEAVERM	
u. The OLTEP resource manager cleans up the resources associated with the address space being terminated by freeing the OLTEP control blocks: • OLTEP common area (module IFDOLT23) • CHASCT (OLT program control table) • DEVTAB (device tables) • MCT (module control table) • OLTTAB (OLT program link table) • SECLST (test section list) • RESTAB (CDS equate resident table) (See the publication *OLTEP Logic*, for a complete description of the OLTEP resource manager.)	IFDOLT0A	



IEAVTMMT – Address Space Purge Resource Managers (Part 10 of 10)

Extended Description	Module	Label
v. The 3850 mass storage system resource manager marks invalid all delayed response queue elements relating to the terminating address space.	ICB2AIR	
w. The RTM resource manager frees all the SDWAs (system diagnostic work areas) obtained from SQA (system queue area) during FRR processing in RTM1.	IEAVTMRM	
x. The type 1 message resource manager cleans up the resources by freeing any entries in the type 1 message table associated with the address space being terminated.	IEAVTPMT	
y. The IEASMFSP memterm resource manager frees the SSB. The SSB is used to keep track of the suspended address space.	IEASMFSP	
z. The address space purge routine uses the PURGEDQ function to free the SRBs associated with the terminating address space. (The M.O. diagram IEAVEPD0 – PURGEDQ Processing in section "Supervisor Control" fully describes this processing.)	IEAVTMMT	
aa. The virtual address space terminating routine acts as a resource manager to clean up the resources held by the terminating address space by freeing the ASCB and indicating in the ASVT the ASID of the address space associated with the terminating address space.	IEAVGCAS	
2 The address space purge routine returns control to RTM2 after all the resources have been freed.	IEAVTRTE	

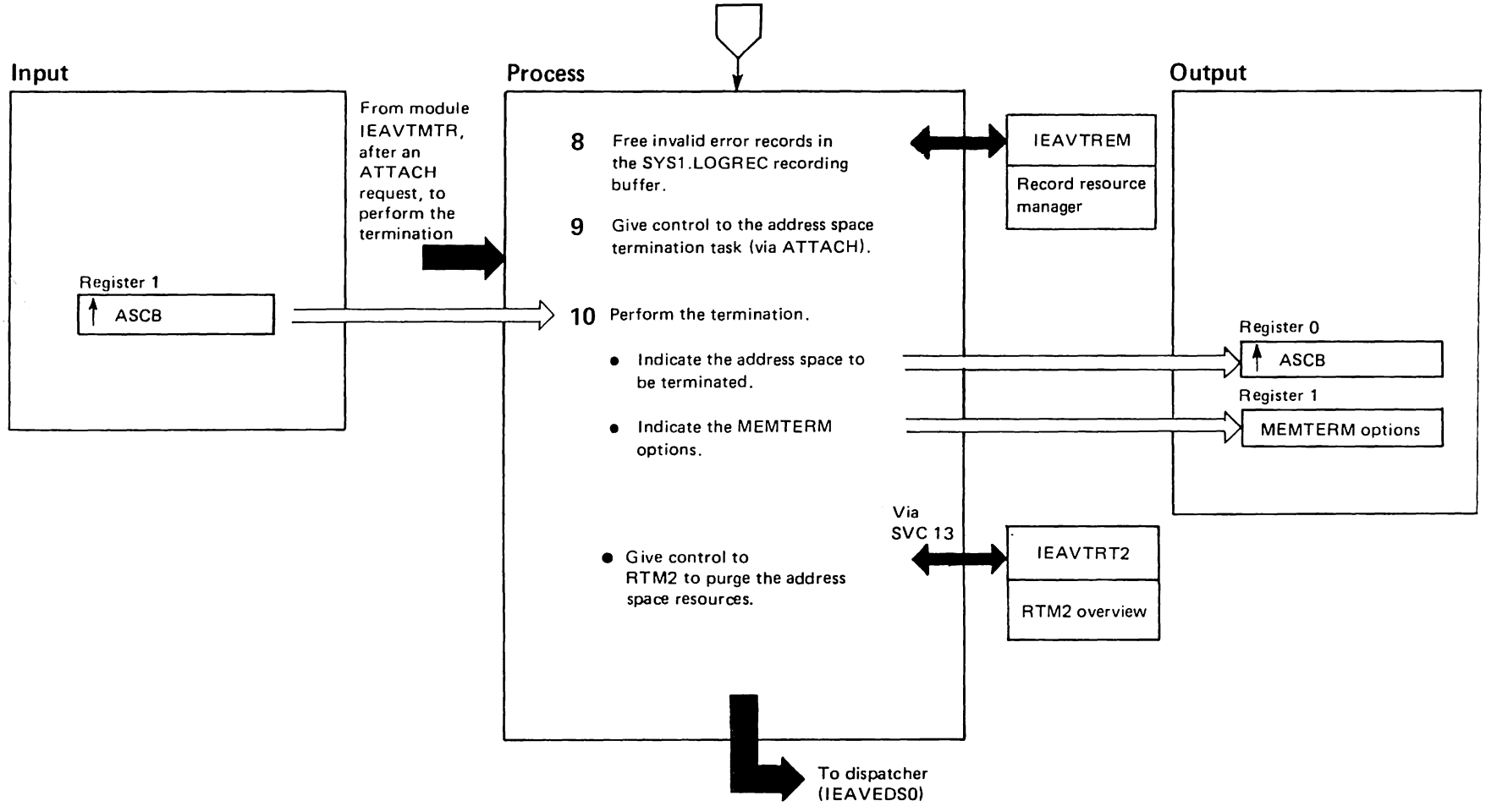
IEAVTMTC – Address Space Termination Processing (Part 1 of 4)



IEAVTMTC – Address Space Termination Processing (Part 2 of 4)

Extended Description	Module	Label
Address space termination consists of two routines, IEAVTMTC and IEAVTMTR, both of which are resident in the master address space. When a system routine issues a CALLRTM TYPE=MEMTERM request, RTM1 gives control to the address space termination routine to:		
• Find and dequeue the ASCB (address space control block) representing the address space to be terminated.		
• Stop the processing in the address space.		
• Perform the actual termination.		
• Repeat the operation for all the ASCBs on the termination queue.		
After this processing has completed for all the address spaces on the termination queue, IEAVTMTC goes into a wait state to wait for another address space termination request.		
1 Since this routine receives control after an SRB scheduled by RTM1 posts its ECB, IEAVTMTC sets the ECB to zero to allow for later entries.	IEAVTMTC	
2 IEAVTMTC uses a CS (compare and swap) instruction to remove the last ASCB from the termination queue. The RTCT (recovery termination control table) points to this queue.		
3 IEAVTMTC marks the address space non-dispatchable so that no new work can execute in the address space. If there is more than one online processor in a multiprocessing environment, additional steps are taken to stop any activity in the address space.		
• Calls memory switch (IEAVEMS0) to set every PSAANEW field to point to the master address space.	IEAVEMS0	
3 (continued)		
• Spins until no other processor is executing work in the terminating address space and no other processor is executing work that holds the local lock of the terminating address space. If the length of the spin exceeds a predetermined time limit, IEAVTMTC gives control to the excessive spin notification routine (IEEVEXSN), which issues message IEE331A to inform the operator.	IEEVEXSN	
• Calls the bind break service routine (IEAVEBBR) to ensure that no processor has an active addressing bind with the terminating address space.	IEAVEBBR	
4 IEAVTMTC gives control to the lock manager resource manager to release any cross memory locks (CML) or the LOCAL lock, if any of these are held by the terminating address space. The CVTLKRM field of the CVT contains the entry point address of the lock manager resource manager.	IEAVLKRM	
5 IEAVTMT issues an SVC 16 macro to give control to the I/O supervisor, which stops all activity for the address space being terminated.	IGC0001F	
6 IEAVTMTC gives control to IEAVETRM to:	IEAVETRM	
• Stop tracing on each processor if the terminating address space is the system trace address space.		
• Remove all trace table copy headers (TTCH) for the terminating address space from the TTCH queue and free them if an address space other than the trace address space is terminating.		
The CVT field CVTTRCRM contains the entry point address of the system trace resource manager.		
7 IEAVTMTC gives control to the real storage management routine to release all real page frames and all auxiliary storage pages belonging to the address space. The SCVTPTRM field of the SCVT contains the entry point address of the real storage management routine.	IEAVTERM ILRTERMR	

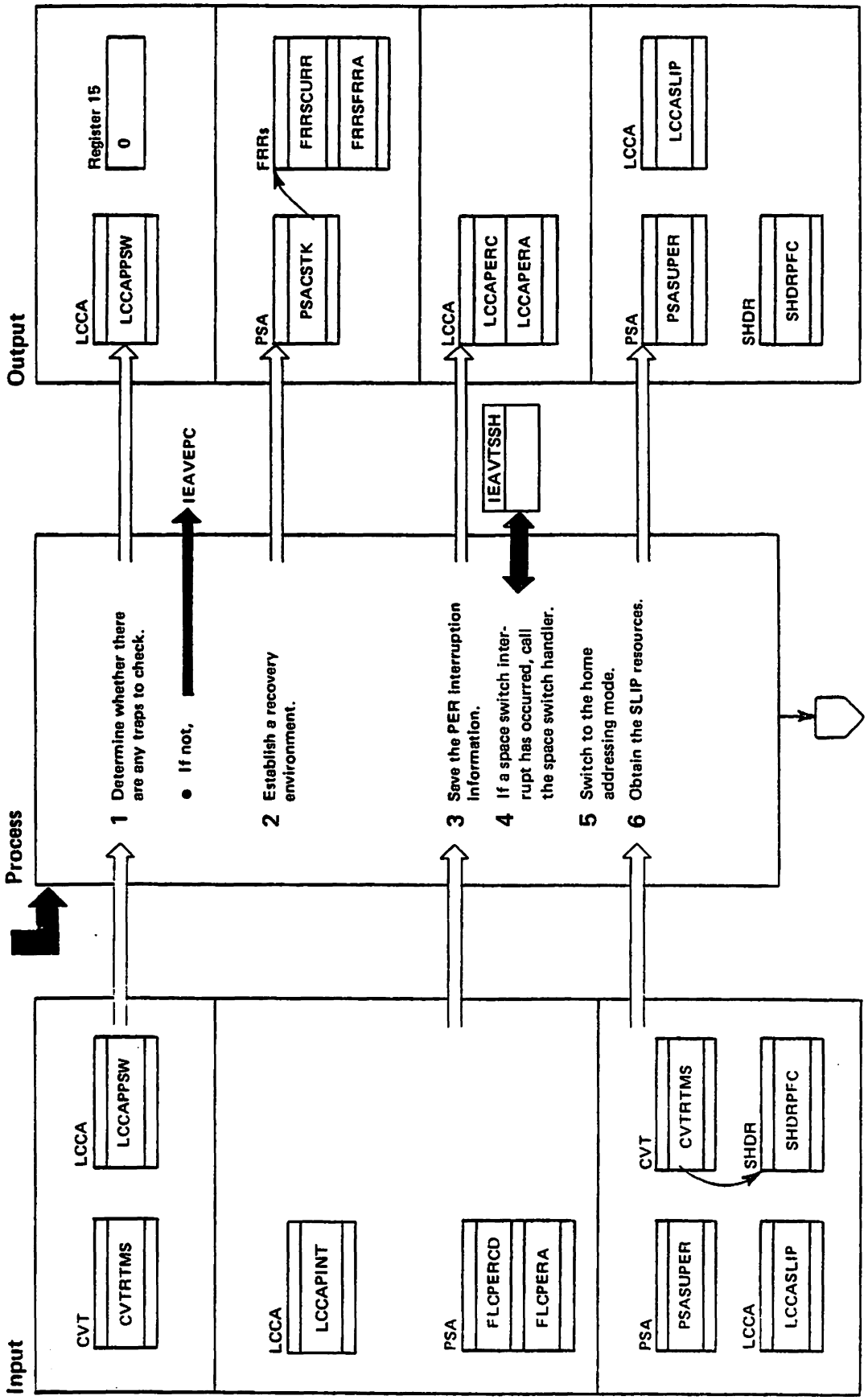
IEAVTMTC – Address Space Termination Processing (Part 3 of 4)



IEAVTMTC – Address Space Termination Processing (Part 4 of 4)

Extended Description	Module	Label
8 IEAVTMC gives control to the record resource manager to mark as invalid any incomplete entries in the SYS1.LOGREC recording buffer.	IEAVTREM	
9 Address space termination continues after module IEAVTMTC, the controller routine, attaches the address space termination task, IEAVTMTR, to perform the actual termination. (IEAVTMTR runs in the master address space.)		
10 The address space termination task indicates the address space being terminated in register 0 and the MEMTERM options in register 1. IEAVTMTR gives control to RTM2, via SVC 13, to purge the address space resources. (See the M.O. diagram IEAVTRT2 – RTM2 Overview and the M.O. diagram IEAVTRTE – Address Space Purge Processing for the description of how RTM purges address space resources.) After control comes back from RTM2, the address space termination task gives control to the dispatcher.	IEAVTMTR	

IEAVTPER - PFLIH/SLIP AND PFLIH/SPACE SWITCH HANDLER INTERFACE



IEAVTPER – PFLIH/SLIP and PFLIH/Space Switch Handler Interface (Part 1 of 4)

IEAVTPER – PFLIH/SLIP and PFLIH/Space Switch Handler Interface (Part 2 of 4)

Extended Description	Module	Label
After detecting a PER and/or space switch interruption, the program check FLIH (IEAVEPC) calls this module to provide an interface with the SLIP action processor (IEAVTSLP) and/or the space switch handler (IEAVTSSH).		
1 IEAVTPER determines whether any SLIP traps have been defined by referring to the CVTRTMS field (the pointer to the SLIP header control block). If there are no traps (CVTRTMS=0), IEAVTPER prevents future PER interruptions from occurring in the interrupted program by setting the PER bit in the resume PSW (LCCAPPSW) to zero. IEAVTPER then returns to IEAVEPC with a return code of zero. If CVTRTMS≠ 0, SLIP traps exist and need to be checked. Processing continues at the next step.	IEAVTPER	
2 IEAVTPER adds an FRR (entry point VTPERFRR in this module) to the stack and initializes an FRR parameter list.		
3 IEAVTPER saves the PER code (FLCPERCD) and the address of the instruction causing the interruption (FLCPERA) in the LCCA for use by IEAVTSLP. (See the M.O. diagram IEAVTSLP – SLIP Action Processor-Part 1 for a description of IEAVTSLP.)		
4 IEAVTPER checks the interrupt code in the LCCAPINT field. If a space switch interrupt has occurred, IEAVTPER calls the space switch handler (IEAVTSSH). (See the M.O. diagram IEAVTSSH – SLIP Space Switch Handler for a diagram and extended description of IEAVTSSH.)		
5 IEAVTPER receives control with the cross memory mode in effect at the time of the PER or space switch interrupt. Because SLIP processing is always done in the home address space, IEAVTPER issues a CMSET SET macro to PSAAOLD (the home address space).		
6 Before calling IEAVTSLP to process the PER interruption, IEAVTPER obtains several SLIP resources. IEAVTPER: • Sets the PSASLIP super bit to one for recursion control. • Obtains ownership of the processor's local SLIP work/save area to prevent this storage from being freed. • Increases the page fix counter (SHDRPFC) by one to prevent the command processor (IEECB905) from freeing the fixed portion of the IEAVTSLP load module page. If the SLIP work/save area does not exist (LCCASLIP=0) or is busy (the high order bit of the LCCASLIP=1), or if IEAVTSLP is not page-fixed (SHDRPFC <1), the interruption cannot be processed. IEAVTPER sets the PER bit in the resume PSW (LCCAPPSW) to zero to prevent future PER interruptions, then returns to IEAVEPC with a return code of zero. If the resources have been obtained, processing continues at the next step.		

IEAVTPER – PFLIH/SLIP and PFLIH/Space Switch Handler Interface (Part 3 of 4)

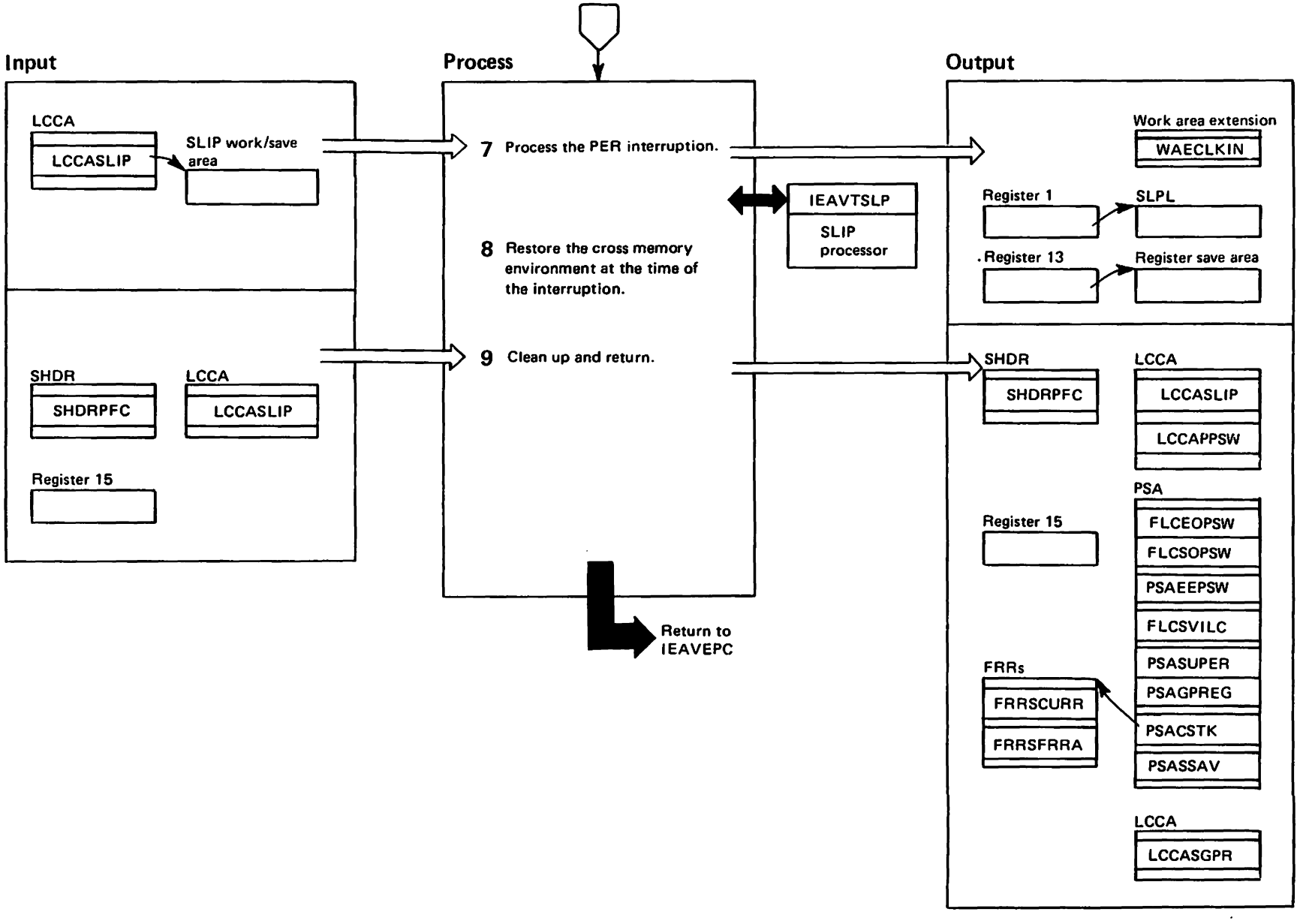


Diagram RTM-14. IEAVTPER – PFLIH/SLIP and PFLIH/Space Switch Handler Interface (Part 4 of 4)

Extended Description	Module	Label

7 IEAVTPER uses a STCK instruction to place the current time of day in the WAECLKIN field for subsequent percent limit processing by IEAVTSLP. IEAVTPER saves information that could be overlayed by a recursive call for the external and SVC FLIH. IEAVTPER also builds the IEAVTSLP parameter list (SLPL), indicating that a PER interruption is to be processed, then calls IEAVTSLP to process it. (See the M.O. diagram IEAVTSLP – SLIP Action Processor-Part 1 for a description of IEAVTSLP.)

IEAVTPER issues a CMSET RESET macro to restore the cross memory environment to what it was at the time of the PER or space switch interrupt.

8 IEAVTPER disables PSA protection to restore information saved for the external and SVC FLIH, relinquishes ownership of the SLIP work/save area, and decreases the SHDRPFC counter by one. Depending on the code returned by IEAVTSLP, IEAVTPER takes the following action:

Return code from IEAVTSLP	*IEAVTPER processing*
0	Leaves the return code unchanged.
4	Leaves the return code unchanged.
8	Turns off PER monitoring in the interrupted program by setting the PER bit of the LCCAPPSW field to zero; sets a return code of zero.
All others	Sets the PER bit of the LCCAPPSW to zero and leaves the return code unchanged.

IEAVTPER uses the return code to inform IEAVEPC that it should either resume processing where the PER interruption occurred (return code = 0), or force recovery processing (return code ≠ 0). Before returning to IEAVEPC, IEAVTPER sets the PSASLIP bit to zero and deletes the FRR.

Extended Description	Module	Label

Recovery processing

Recovery for this module is designed to allow the system to continue executing at the risk of not processing a PER interruption. When an error occurs, IEAVTPER receives control at entry point VTPERFRR, where it attempts to release all the resources obtained by IEAVTPER and to retry at a point in IEAVTPER where minimal processing is required to return to IEAVEPC. To do so, the FRR refreshes the registers necessary for the retry from the FRR parameter list. If the error occurred before IEAVTPER saved these critical registers, the FRR percolates. Percolation also occurs if various RTM footprints indicate that critical registers expected by IEAVEPC might not be available.

IEAVTPVT - MODULE DESCRIPTION

DESCRIPTIVE NAME: SLIP PVTMOD LOAD/DELETE exit routine

FUNCTION:

This module processes a contents directory entry (CDE) whenever CDEs enter or leave the job pack area queue if the CDE matches a PVTMOD PER trap.

ENTRY POINT: IEAVTPVL

PURPOSE:
Handles CDE(s) coming onto the job pack queue and receives control from LINK, LOAD, IDENTIFY, Virtual Fetch, ATTACH, and XCTL. When appropriate, IEAVTPVL starts PER monitoring by setting the PER control registers.

LINKAGE: Via BASSM 14,15

CALLERS:
By the Contents Supervisor when a CDE is placed on the job pack queue and the extent list is valid (LINK, LOAD, IDENTIFY, XCTL, and, ATTACH) or when the module is brought into memory (Virtual Fetch).

INPUT:
Key input items are:
- SHDRPVMN field - Module name associated with PVTMOD PER trap
- SHDRPER field - Pointer to the non-ignore PER trap
- SHDRPERR field - Pointer to the PER range to be monitored
- SHDRPCDE field - CDE address of PVTMOD load
- SHDRPVAS field - ASID of PVTMOD load
- SCVAAS field - ASID selections for a trap
- ASCBASID field - Address space id

OUTPUT:
SCVAMDA1 and SCVAMDA2 fields are set to the actual addresses to be monitored.
SHDRPVTA field is turned on to indicate the PER trap is active.
SHDRPVLP field is turned off.
SHDRPCDE field contains the address of the CDE for the PVTMOD PER trap.
Either SHDRPVTL or SHDRPVTG is set to indicate whether the module was loaded locally or globally.
PER control registers are set to start PER.

EXIT NORMAL: Returns to the caller.

EXIT ERROR: Percolates from entry point, PVTFRR.

ENTRY POINT: IEAVTPVD

PURPOSE:
Handles CDE(s) taken off the job pack queue and receives control from LINK, DELETE, and VIRTUAL FETCH. When appropriate, IEAVTPVD stops PER monitoring by setting the PER control registers.

LINKAGE: Via BASSM 14,15

CALLERS:
By the Contents Supervisor when a CDE is removed from the job pack queue.

INPUT:
Key input items are:
- SHDRPVMN field - Module name associated with PVTMOD PER trap
- SHDRPER field - Pointer to the non-ignore PER trap

SHDRPERR field - Pointer to the PER range to be monitored
SHDRPCDE field - CDE address of PVTMOD load
SHDRPVAS field - ASID of PVTMOD load
SCVAAS field - ASID selections for a trap
ASCBASID field - Address space ID

OUTPUT:
SHDRPVTA field is turned off.
SHDRPVLP field is turned on indicating that the PVTMOD PER trap is waiting for IEAVTPVL to be entered to restart PER monitoring.
PER control registers are reset to stop PER monitoring.

EXIT NORMAL: Returns to the caller

EXIT ERROR: Percolates from entry point, PVTFRR.

ENTRY POINT: IEAVTPVR

PURPOSE:
Receives control from Virtual Fetch's resource managers on end of job step task and end of memory. When appropriate, IEAVTPVR stops PER monitoring by setting the PER control registers.

LINKAGE: Via BASSM 14,15

CALLERS:
By Virtual Fetch resource manager at the end of a job step task or end of memory.

INPUT:
Key input items are:
SHDRPTCB field - TCB of PVTMOD load
SHDRPER field - Pointer to the non-ignore PER trap
SHDRPVAS field - ASID of PVTMOD load
RMPLASID field - Address space id
RMPLASCB field - ASCB address
RMPLTCBA field - TCB address for end of task

OUTPUT:
SHDRPVTA field is turned off.
SHDRPVLP field is turned on indicating that the PVTMOD PER trap is waiting for IEAVTPVL to be entered to restart PER monitoring.
PER control registers are reset to stop PER.

EXIT NORMAL: Returns to the caller.

EXIT ERROR: Percolates from entry point, PVTFRR.

ENTRY POINT: PVTFRR

PURPOSE:
Functions as the FRR for IEAVTPVT. If the error is non recursive, PVTFRR records the error on SYS1.LOGREC, initiates a dump to provide output of the pertinent diagnostic information, and retries to release resources held by the mainline.
If the error is recursive, PVTFRR issues a message to the system operator, zeroes the PER trap pointer in the SHDR, and then percolates. The state of PER and the resources, which may have been acquired for SLIP PER routines, are unpredictable.

LINKAGE:
Entered from RTM with the indicated register contents

IEAVTPVT - MODULE DESCRIPTION (Continued)

CALLERS: RTM

INPUT: SDWA

OUTPUT:
SDWA fields are set
SDUMP is taken

EXIT NORMAL: Returns to the caller

EXIT ERROR: Percolates to RTM from PVTFRR on a recursive error

EXTERNAL REFERENCES:

ROUTINES:
Branch enter lock manager routines
Branch enter SDUMP
Branch enter GETMAIN
Branch enter FREEMAIN
Branch enter recording facility
Branch enter RISGNL routine
Branch enter SLIP PER RISGNL routine(IEAVTSIG)
Branch enter Cross Memory POST

DATA AREAS: No data areas used.

CONTROL BLOCKS:

Common name	Macro ID	Usage	Function
ASCB	IHAASCB	read	Obtains the current ASID.
IEEBASEA		read	Obtains the master's ASCB address.
CDE	IHACDE	read	Obtains the module name, extent list, etc.
CVT	CVT	read	Establishes addressability to IEEBASEA.
PCCA	IHAPCCA	read	Obtains information about the active processor for RISGNL.
PCCAT	IHAPCCAT	read	Locates all active processors.
PSA	IHAPSA	read and write	Obtains the ASCB and FRR addresses.
SRB	IHASRB	read and write	Schedules IEAVTLCL to search the job pack queues.
SCE		read	Obtains information about the SLIP trap.
SCVA		read and write	Obtains information about the SLIP trap.
SDWA	IHASDWA	read and write	Obtains recovery information about the error.
SHDR		read and write	Obtains system SLIP information.
XTLST	IHAXTLST	read	Obtains the module's address and length.

TABLES: No tables used.

SERIALIZATION:

IEAVTPVT has the local lock on entry to serialize the manipulation of the CDEs on the job pack queue. IEAVTPVT obtains the CMS lock to serialize GETCELL/FREECELL processing from which it obtains the SRB, which is used to schedule IEAVTLCL. IEAVTPVT also obtains the dispatcher lock to serialize with IEAVTGLB. To prevent the deletion of SCEs from the SCE chain, IEAVTPVT increments the SHDR use count.

IEAVTPVT - MODULE OPERATION

IEAVTPVT receives control to process a contents directory entry (CDE) whenever CDEs enter or leave the job pack queue if the CDE matches a PVTMOD PER trap.

Entry point IEAVTPVL receives control from the Contents Supervisor when a CDE is placed on the job pack queue via a LINK, LOAD, IDENTIFY, XCTL, or ATTACH macro and the extent list is valid or when the module is brought into memory via Virtual Fetch. IEAVTPVL performs the following processing:

- Sets the control register on all the processors to start PER monitoring if these conditions are valid:
  - A PVTMOD PER trap is enabled, that is, not active (the trap has not had its address range set).
  - The module name in the CDE matches the module name in the PVTMOD trap.
  - The current address space matches one of the eligible address spaces.

Entry point IEAVTPVD receives control from the Contents Supervisor when a CDE is removed from the job pack queue via a LINK, DELETE, Virtual Fetch, or XCTL macro and the extent list is valid.
IEAVTPVD performs the following processing:

- If the CDE matches the CDE for which PVTMOD PER monitoring is currently active, IEAVTPVT resets the control registers on all the processors to stop PER monitoring.

- Indicates that for a 'local' PVTMOD trap, the local job pack queue should be searched for another occurrence of the module. IEAVTPVT schedules a local SRB.

Entry point IEAVTPVR receives control from Virtual Fetch resource manager on end of job step task or end of memory.
IEAVTPVR performs the following processing:

- Processes the end of task for the appropriate job step task and end of memory for the proper address space in the same manner as the processing at entry point IEAVTPVD. However, IEAVTPVR does not schedule an SRB if at end of memory.

RECOVERY OPERATION:

Retry is attempted only for the first entry into entry point, PVTFRR. The retry releases all obtained resources and returns to the caller. Before retrying, an SVC dump is taken, setting up a summary list to dump the relevant control blocks.

On a recursive entry into PVTFRR, the FRR frees, if possible, the resources. Then PVTFRR percolates indicating to RTM that the remaining resource (the dispatcher lock or the CMS lock) is to be freed.

IEAVTPVT - DIAGNOSTIC AIDS

ENTRY POINT NAMES: IEAVTPVL
IEAVTPVD
IEAVTPVR
PVTFRR

MESSAGES:

(Displayed by the command processor)
To the system operator:
IEA414I SLIP unable to deactivate PER.
(Issued through RECORD TYPE=WTO)
IEA415I SLIP error attempting to activate/deactivate
PER, dump scheduled.
(Posts the SLIP command processor to issue this message)

ABEND CODES: None

WAIT STATE CODES: None

RETURN CODES: None

REGISTER CONTENTS ON ENTRY:

ENTRY POINT IEAVTPVL:

Register 0	- Irrelevant
Register 1	- Address of the CDE that has been added
Registers 2-12	- Irrelevant
Register 13	- Address of 72 byte save area
Register 14	- Return address
Register 15	- Entry point (SHDRPVL1) (pointer defined)

ENTRY POINT IEAVTPVD:

Register 0	- Irrelevant
Register 1	- Address of the CDE that has been deleted
Registers 2-12	- Irrelevant
Register 13	- Address of 72 byte save area
Register 14	- Return address
Register 15	- Entry point (SHDRPVD1) (pointer defined)

ENTRY POINT IEAVTPVR:

Register 0	- Irrelevant
Register 1	- Address of a word containing the address of the RMPL passed to the resource manager
Registers 2-12	- Irrelevant
Register 14	- Return address
Register 15	- Entry point (SHDRPVR1) (pointer defined)

ENTRY POINT PVTFRR:

Register 0	- Points to a 200 byte work area in fixed SQA
Register 1	- Points to the SDWA
Registers 2-13	- Irrelevant
Register 14	- Return address
Register 15	- Entry point

REGISTER CONTENTS ON EXIT:

IEAVTPVT - DIAGNOSTIC AIDS (Continued)

ENTRY POINT IEAVTPVL:

EXIT NORMAL:

Registers 0-15 - Same as on entry

EXIT ERROR:

Registers 0-15 - Unknown

ENTRY POINT IEAVTPVD:

EXIT NORMAL:

Registers 0-15 - Same as on entry

EXIT ERROR:

Registers 0-15 - Unknown

ENTRY POINT IEAVTPVR:

EXIT NORMAL:

Registers 0-15 - Same as on entry

EXIT ERROR:

Registers 0-15 - Unknown

ENTRY POINT PVTFRR:

EXIT NORMAL:

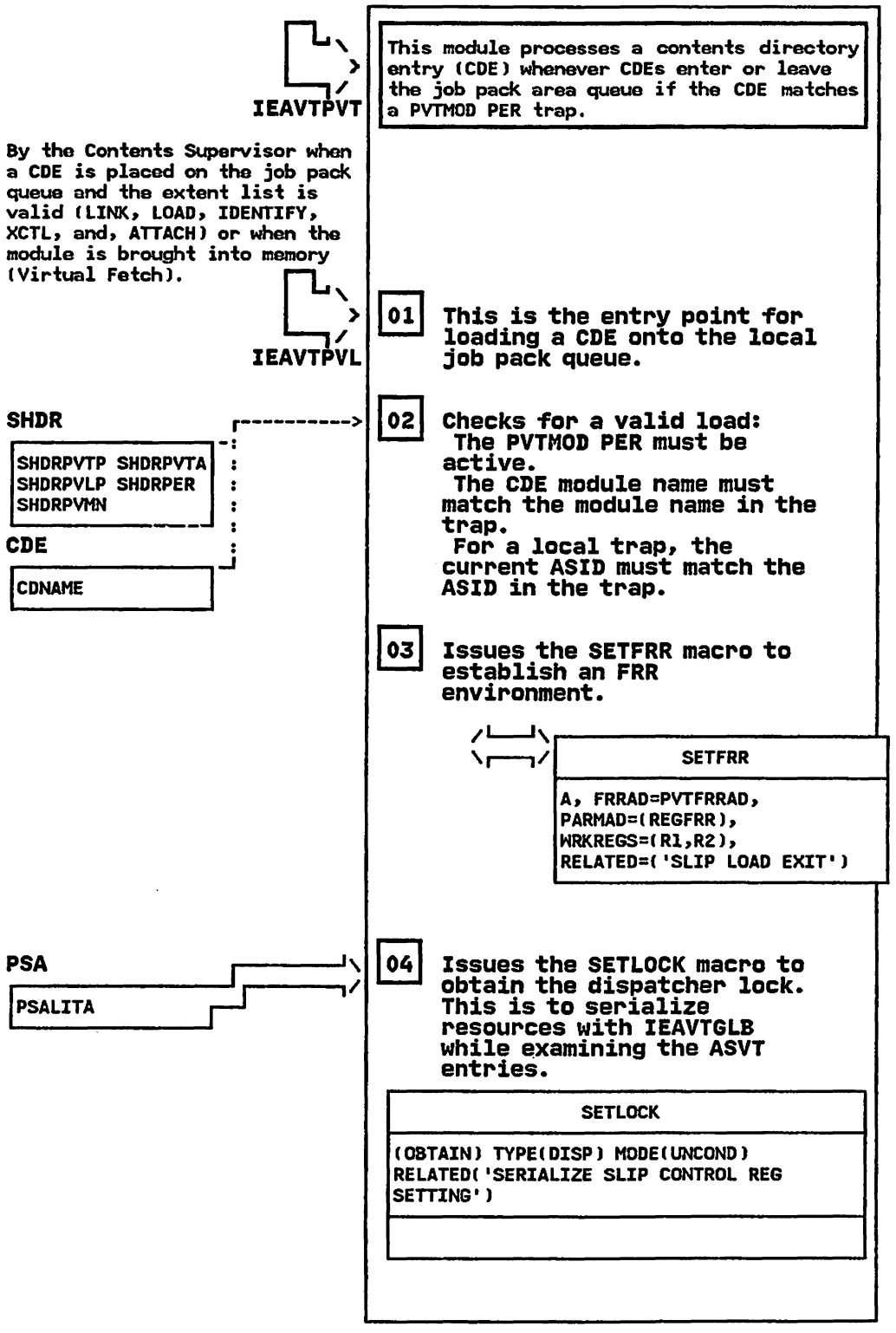
Registers 0-15 - Irrelevant

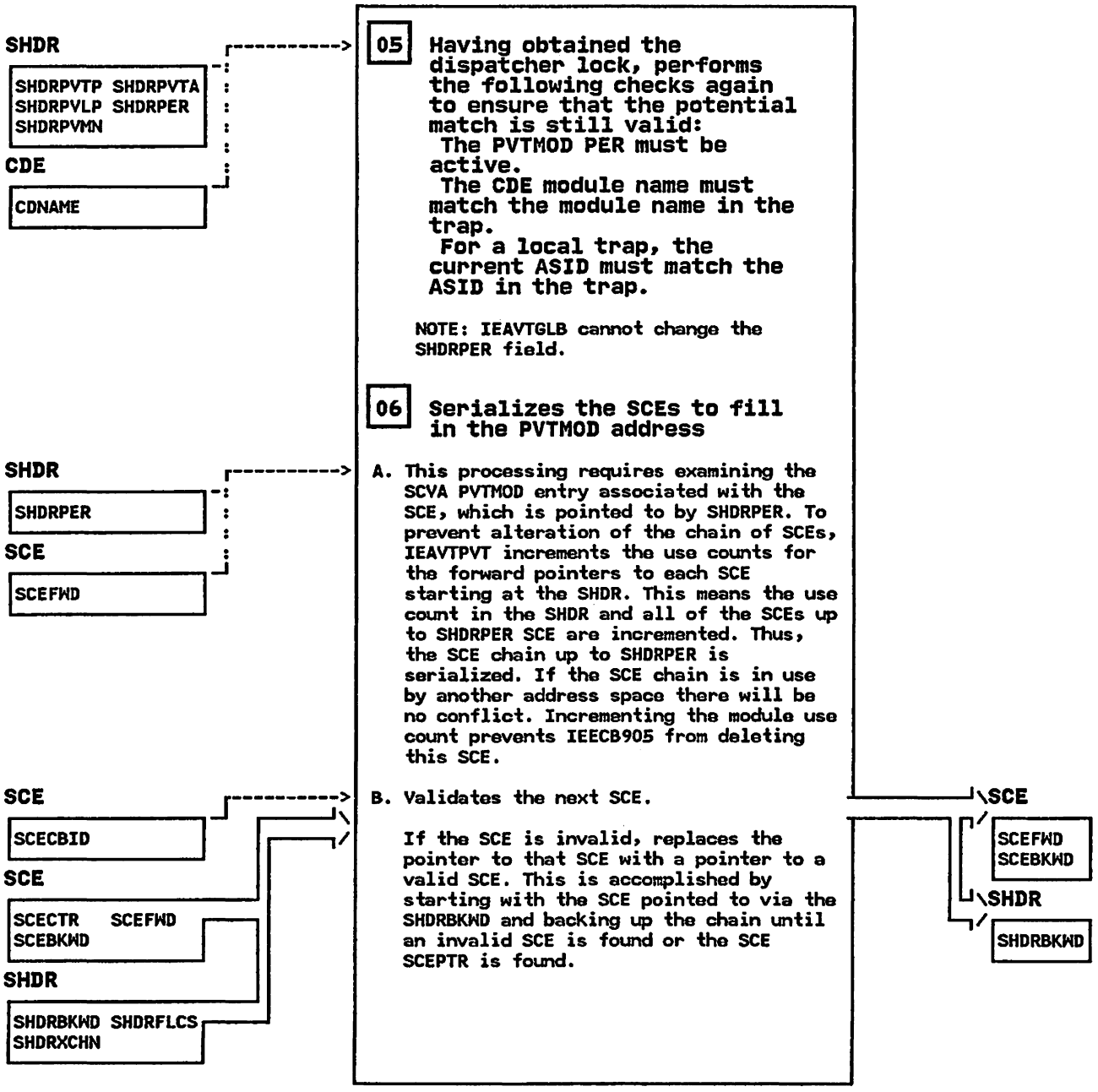
EXIT ERROR:

Registers 0-15 - Irrelevant

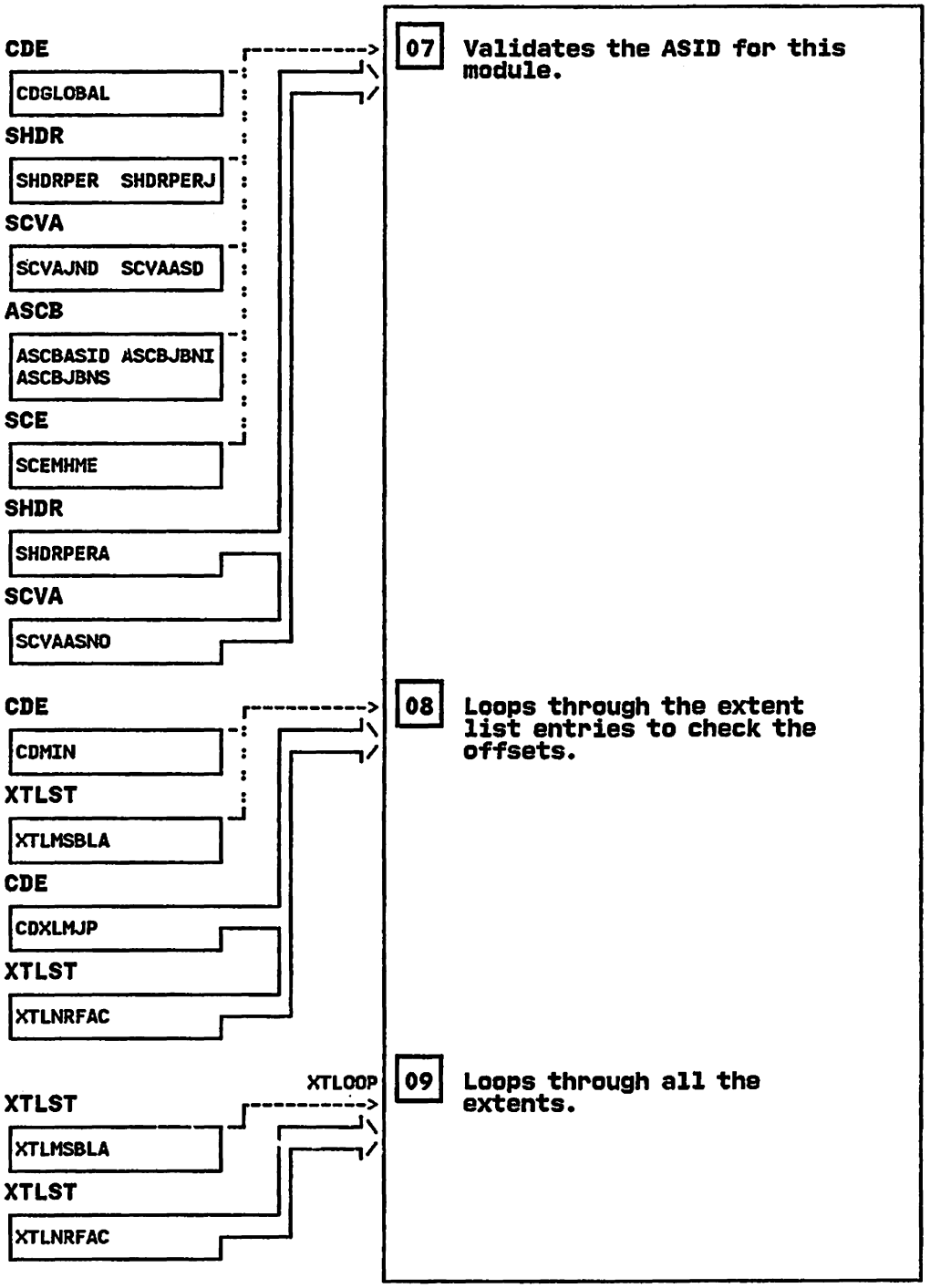
IEAVTPVT - SLIP PVTMOD LOAD/DELETE exit routine

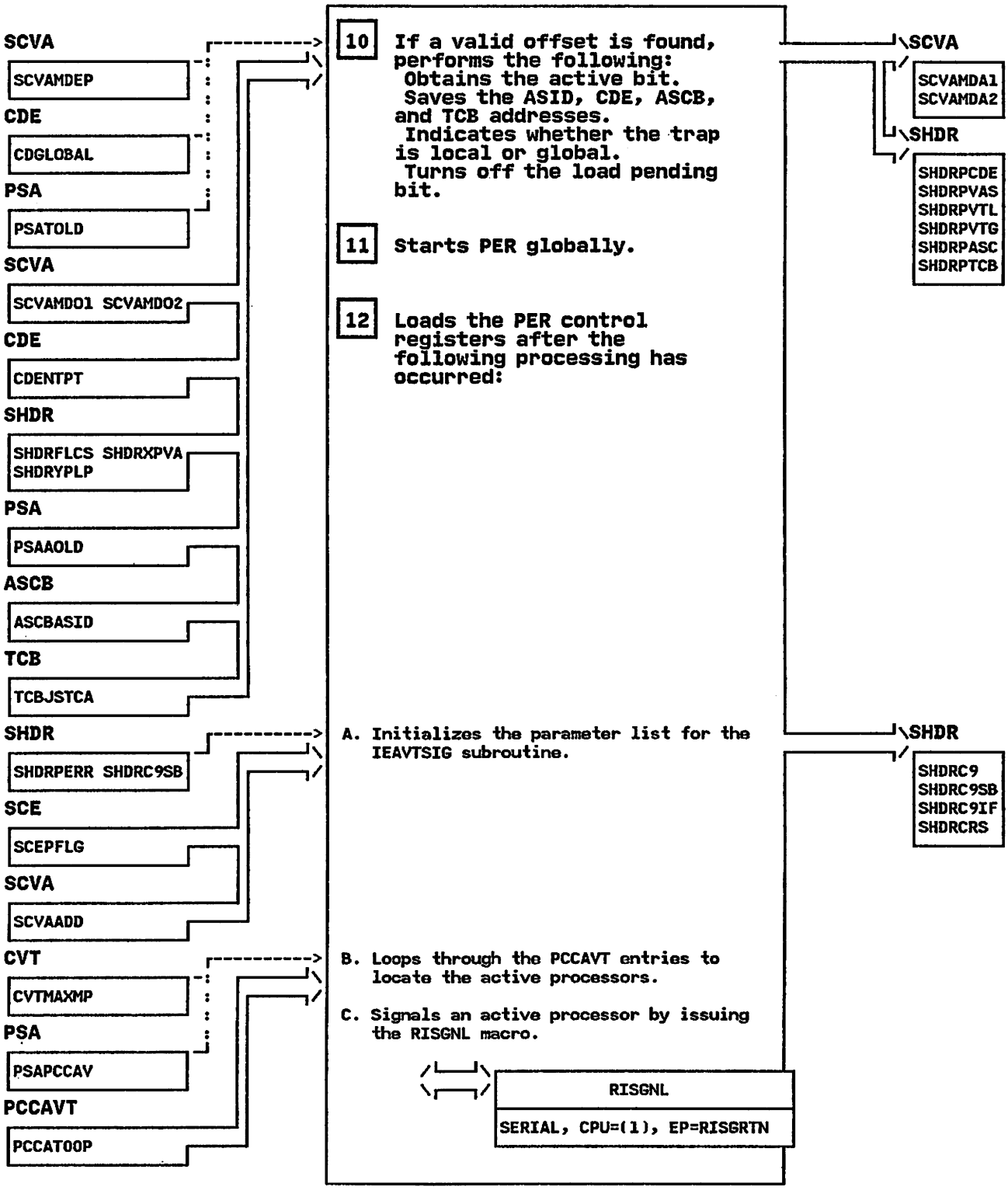
STEP 01



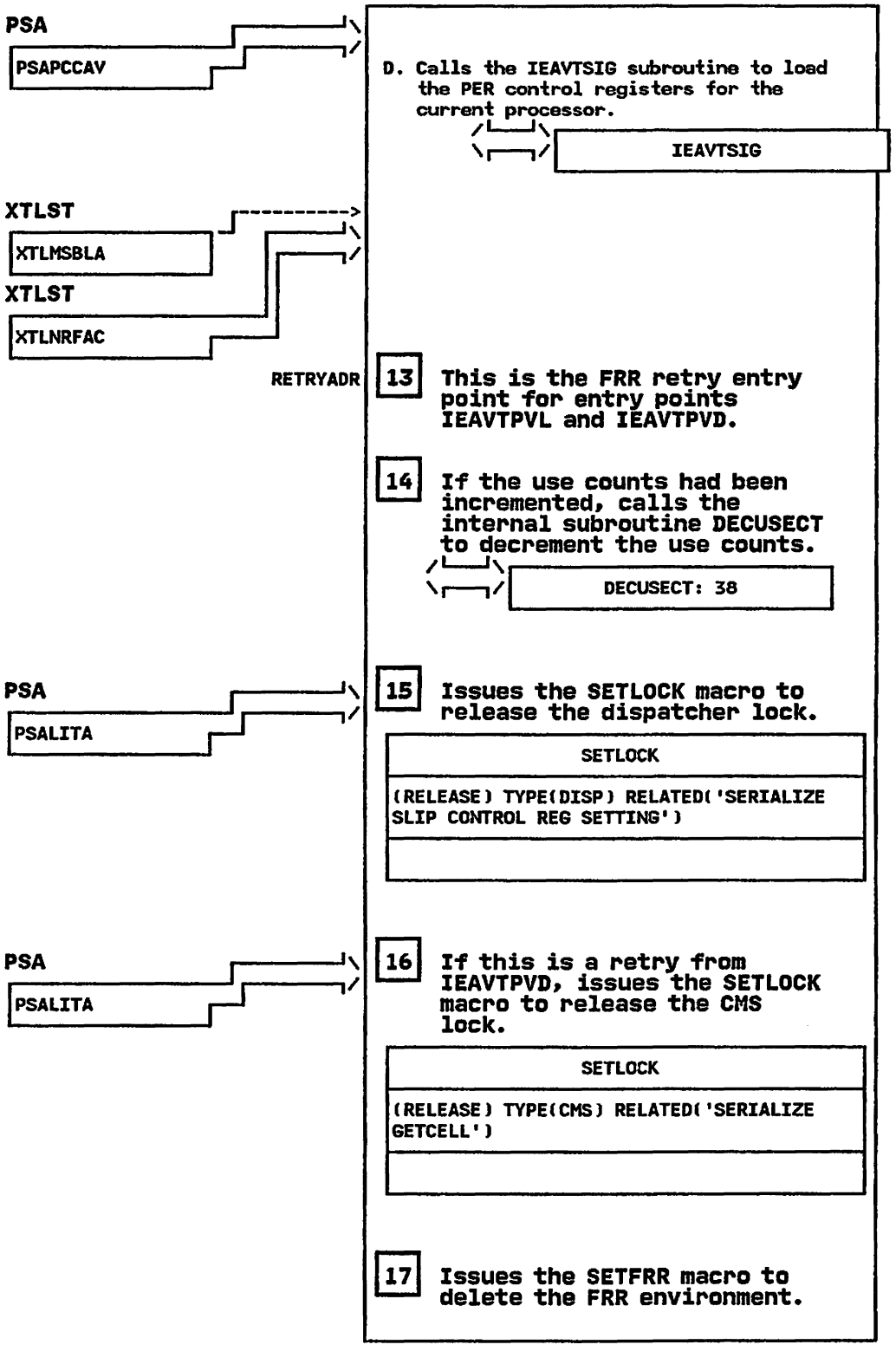


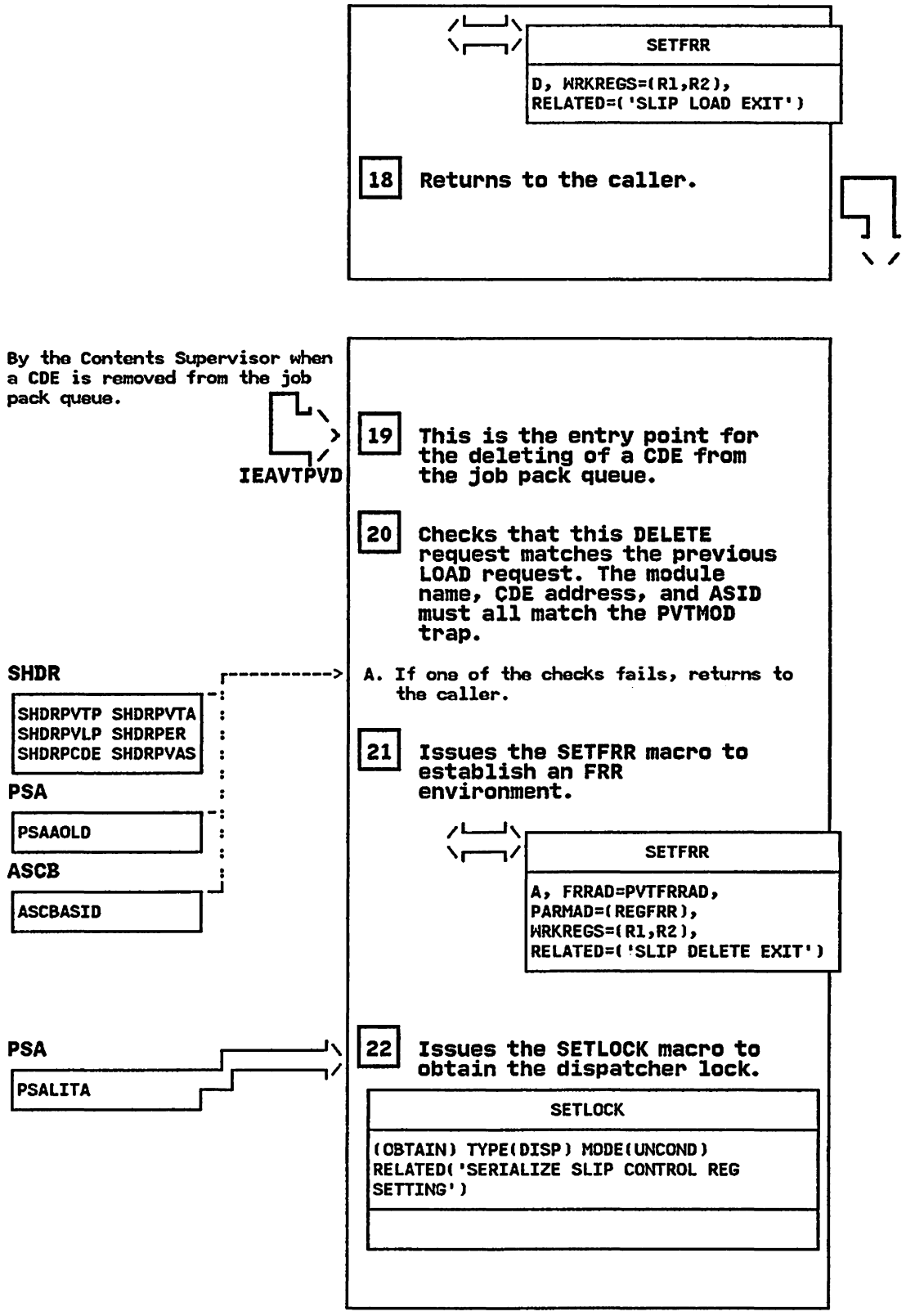
IEAVTPVT - SLIP PVTMOD LOAD/DELETE exit routine STEP 07

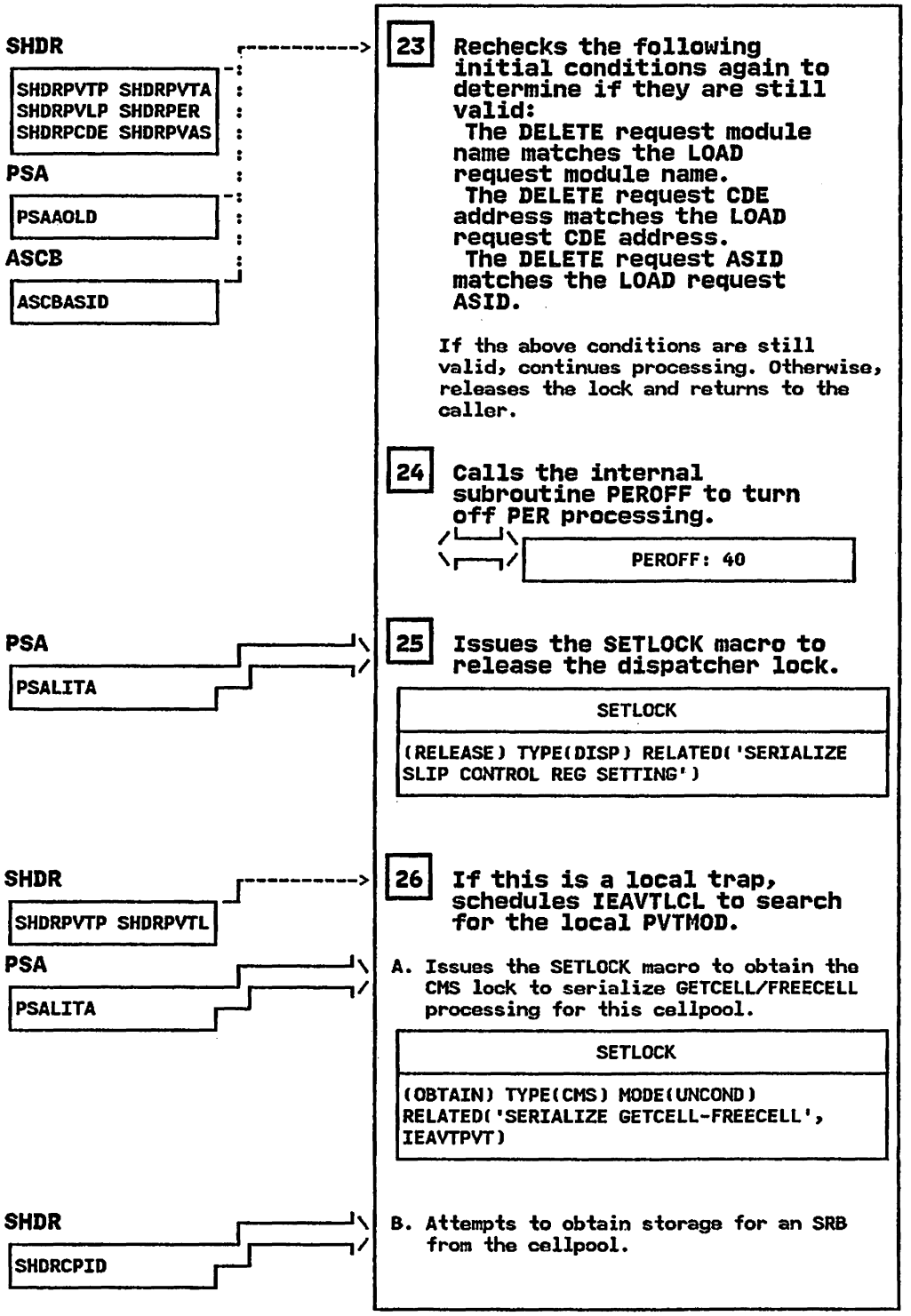


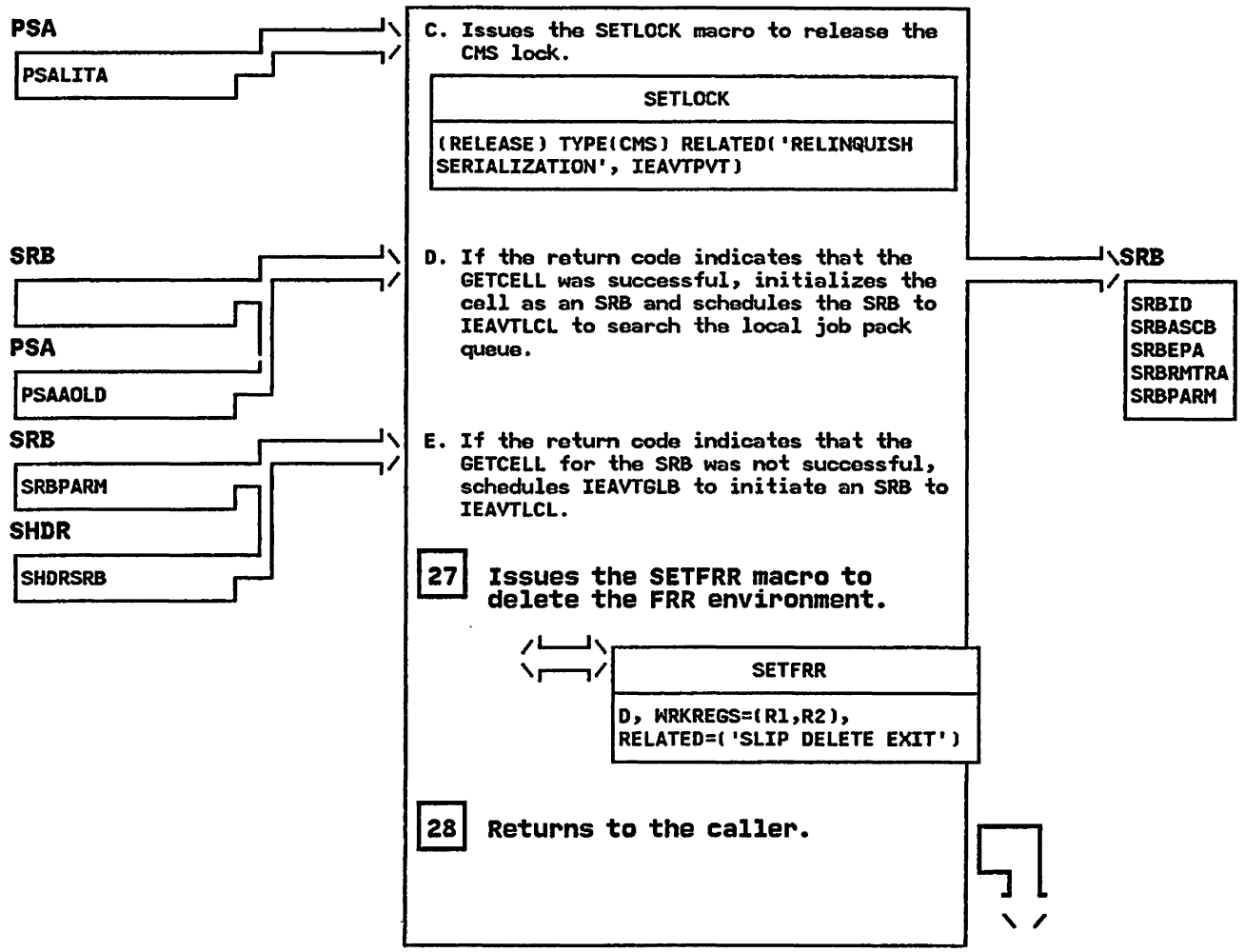


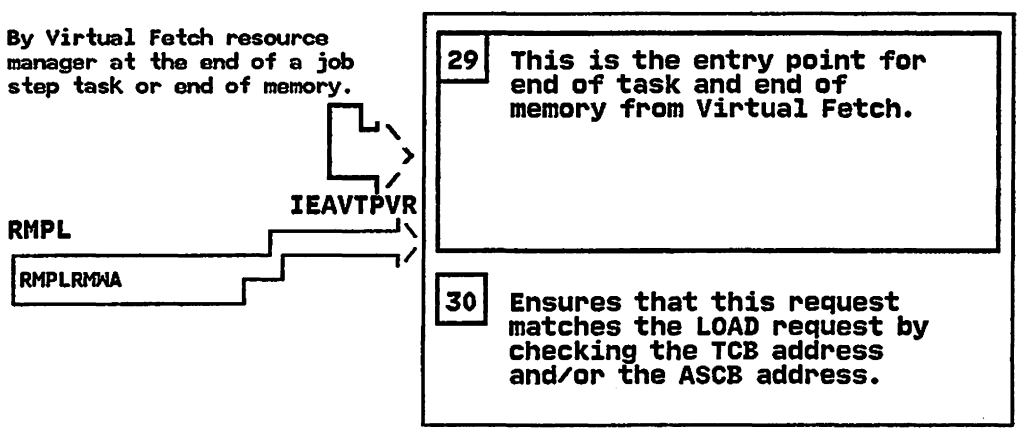
IEAVTPVT - SLIP PVTMOD LOAD/DELETE exit routine STEP 12D

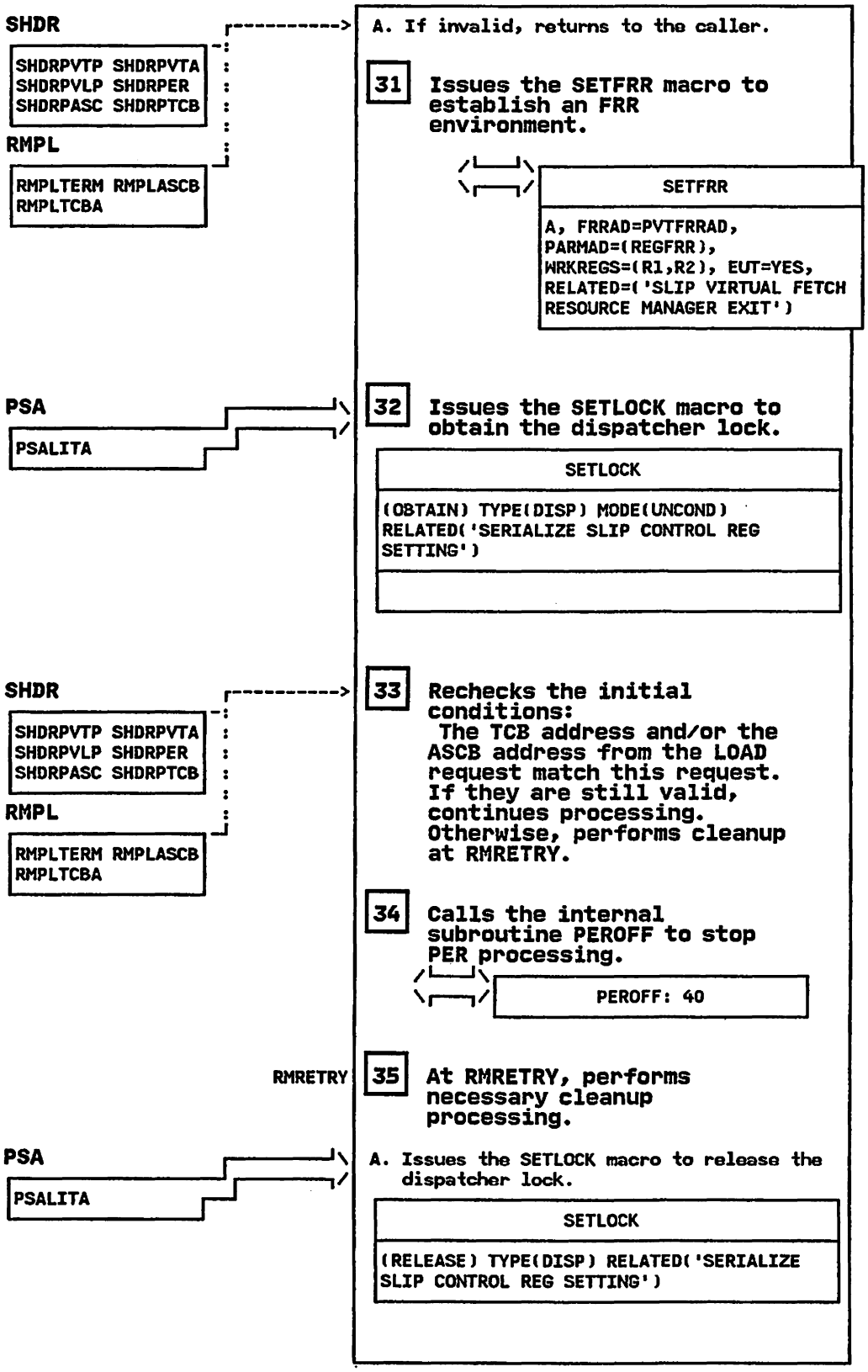


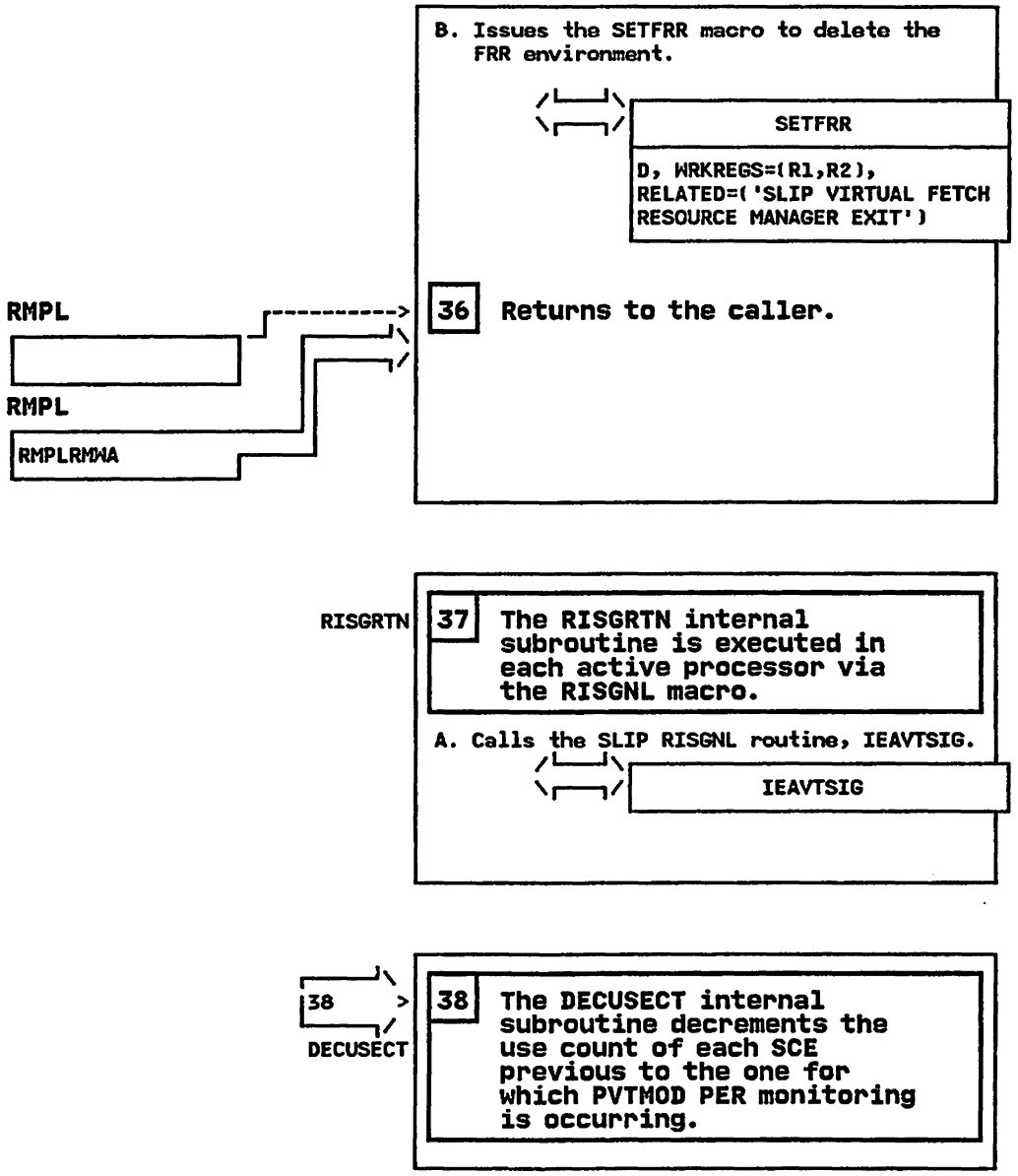




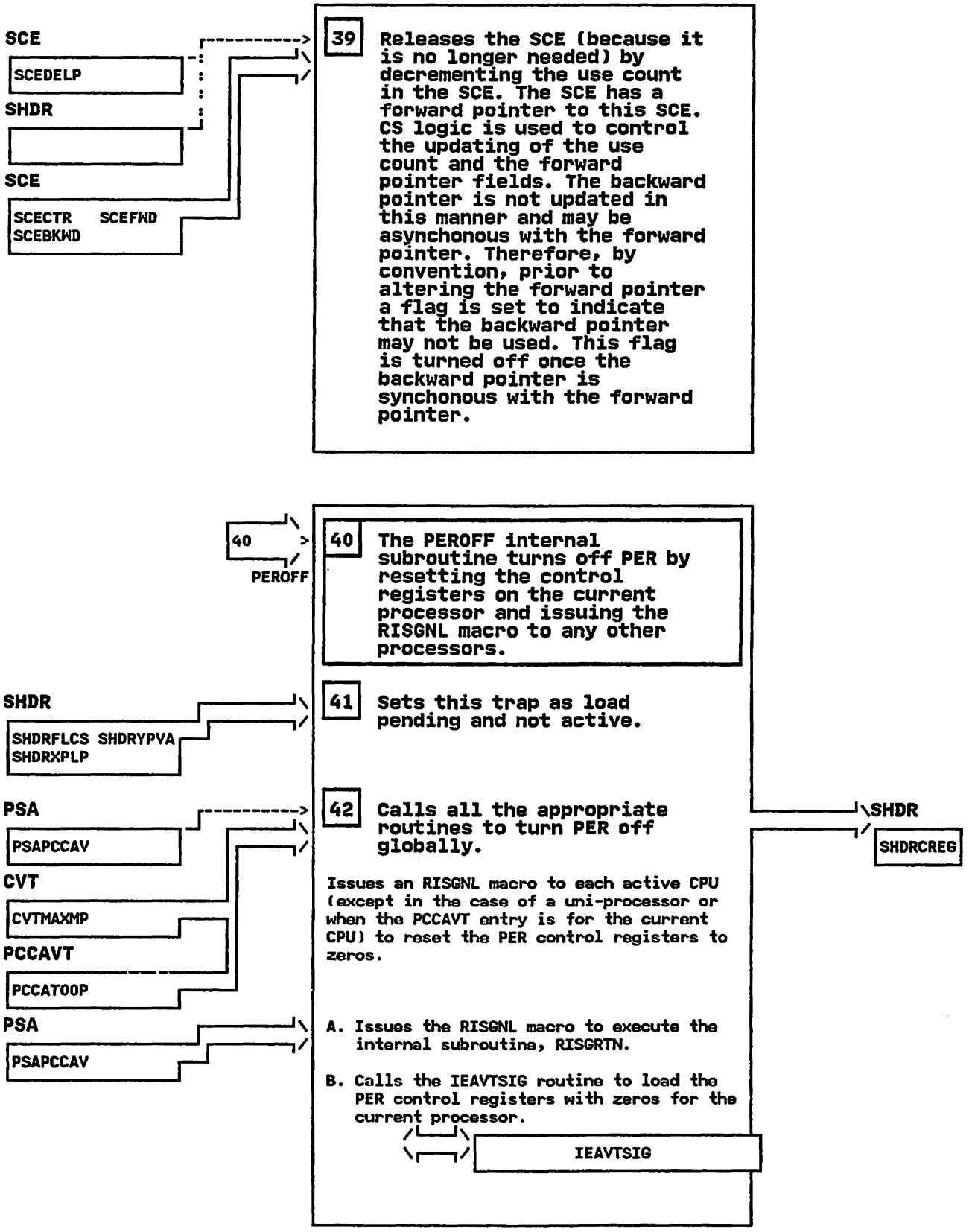


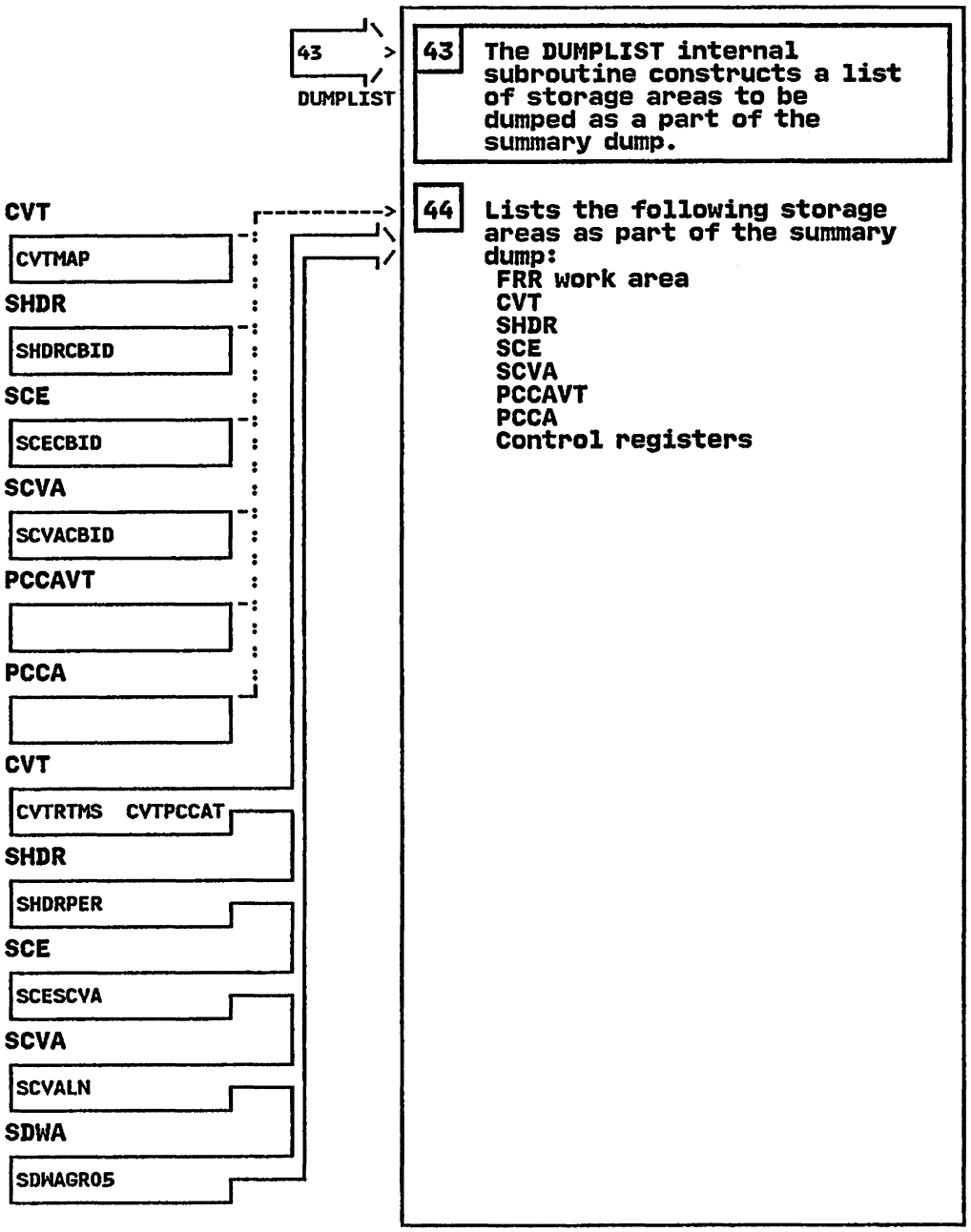




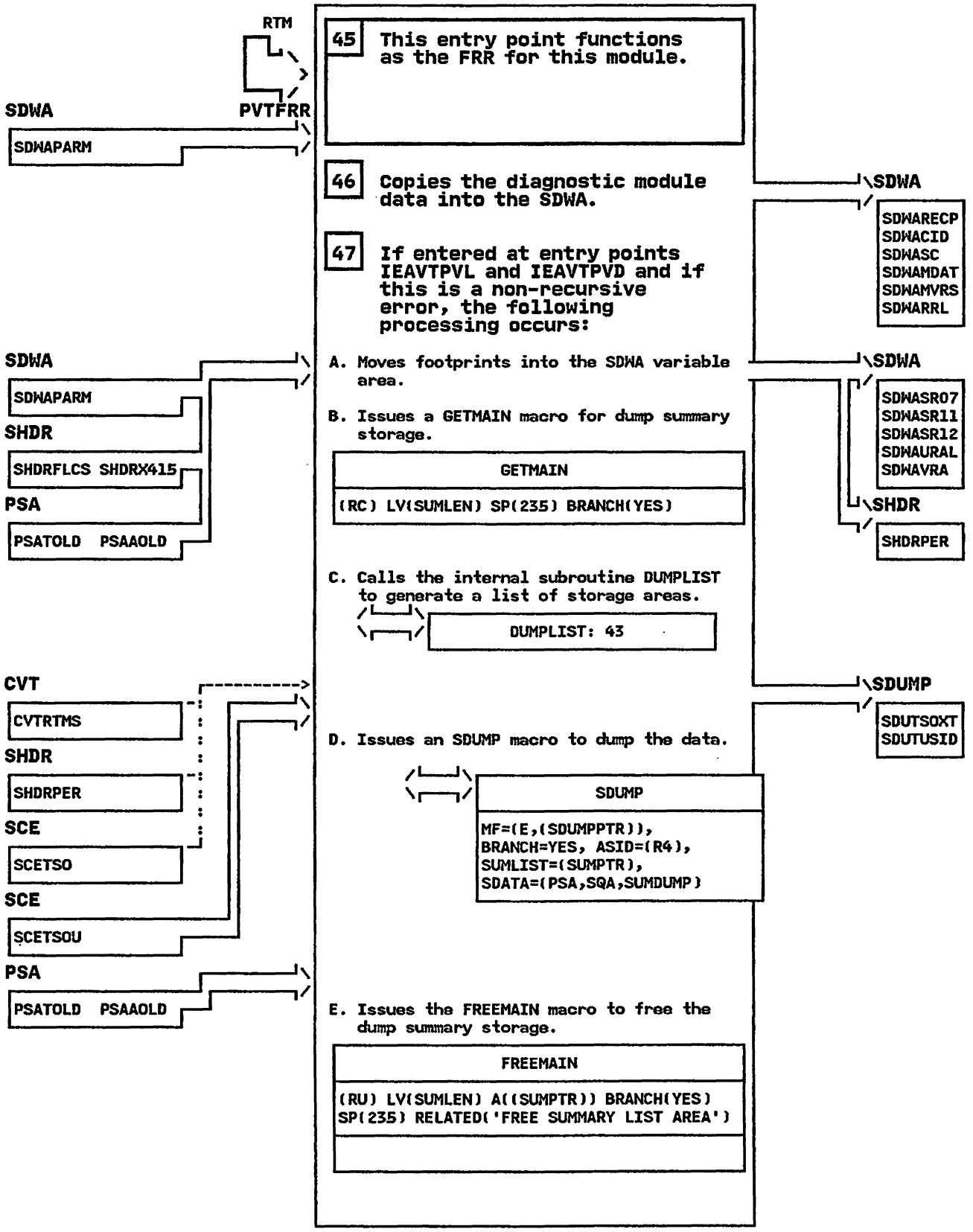


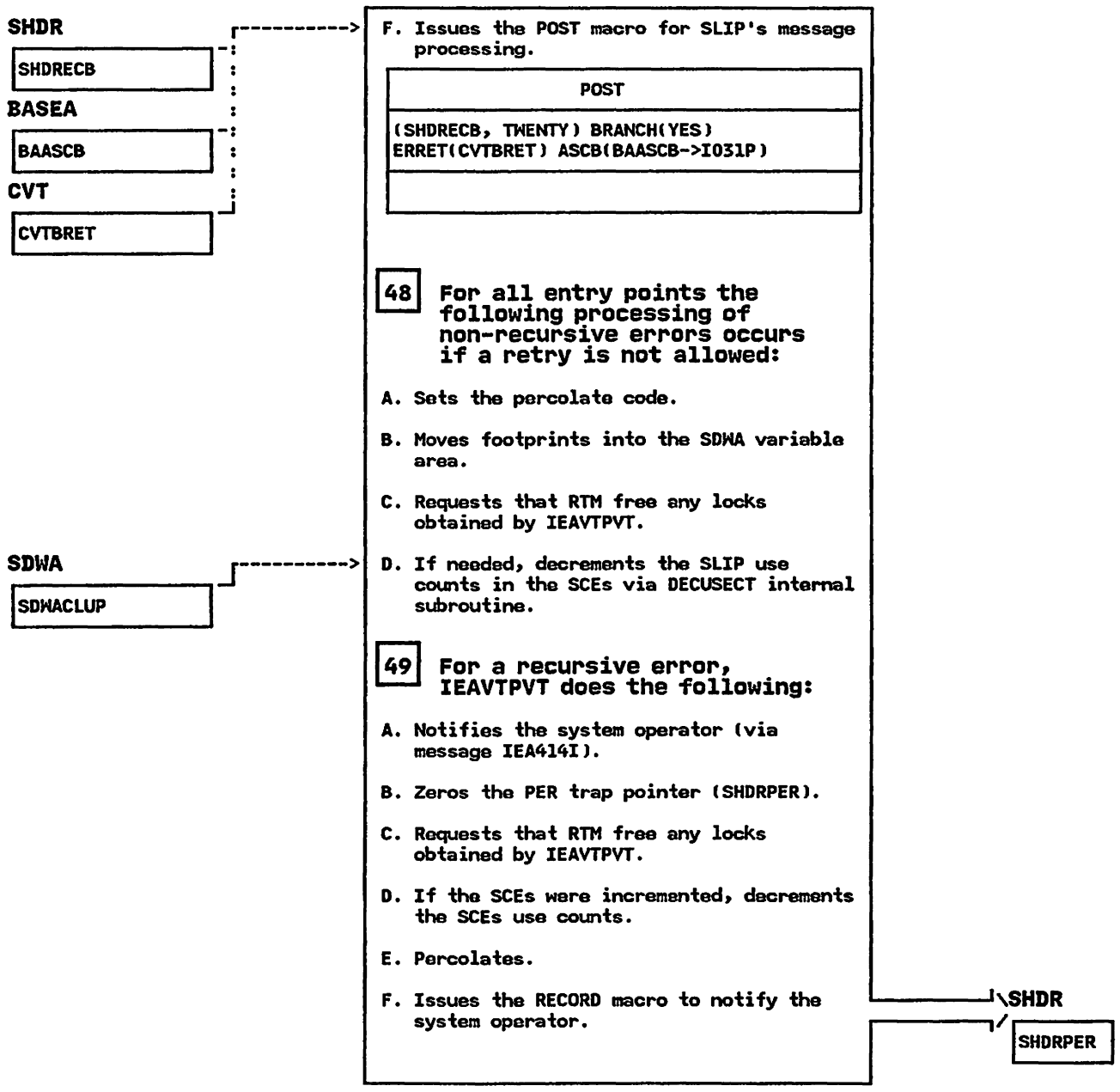
IEAVTPVT - SLIP PVTMOD LOAD/DELETE exit routine STEP 39





IEAVTPVT - SLIP PVTMOD LOAD/DELETE exit routine STEP 45





IEAVTREF - MODULE DESCRIPTION

DESCRIPTIVE NAME: LOGREC Recording Buffer Formatter

FUNCTION:
This module is a dump formatting exit that can be called from Print Dump (PRDMP) or the Interactive Problem Control Program (IPCS). IEAVTREF is invoked when the IPCS/PRDMP LOGDATA verb is executed. IEAVTREF locates the LOGREC entries that are contained in the LOGREC recording buffer and invokes the EREP program to format and print the LOGREC entries.

ENTRY POINT: IEAVTREF

PURPOSE: Formats and prints the LOGREC recording buffer

LINKAGE: LINK or ATTACH

CALLERS: PRDMP or IPCS

INPUT: The Common Exit Parameter List (BLSABDPL)

OUTPUT: The formatted contents of the LOGREC buffer

EXIT NORMAL: Returns to caller

ENTRY POINT: REFIO

PURPOSE: Prints the formatted output from IFCRCGIF

LINKAGE: BALR

CALLERS: IFCRCGIF

INPUT:
The parameter list contains the address of the 134 character formatted print line
The Common Exit Parameter List (BLSABDPL)

OUTPUT: None

EXIT NORMAL: Returns to caller

EXTERNAL REFERENCES:

ROUTINES: IFCRCGIF - EREP service to format LOGREC entries.

CONTROL BLOCKS:

Common name	macro id	usage	function
ABDPL	BLSABDPL	read	Communication area between PRDMP/IPCS and user verb exit routines
CVT	CVT	read	Establishes addressability to the RBCB.
HDR	IHAHDR	read	Required for the expansion of the RBCB.
PSA	IHAPSA	read	Establishes addressability to the CVT.
RBCB	RTMRBCB	read	Establishes addressability to the RCBs
RCB	RTMRCB	read	Contains the entries for the recording requests
RCBENTRY	RTMRCBE	read	Contains the information for a single recording request
SRB	IHASRB	read	Required for the expansion of the RBCB.

SERIALIZATION: None required

IEAVTREF - MODULE OPERATION

IEAVTREF is a dump formatting exit that is invoked to format and print the contents of the LOGREC Recording buffer. IEAVTREF locates the LOGREC buffer in the dump and passes each entry to IFCRCGIF, an EREP service that formats and prints the data. PRDMP/IPCS generates a table of contents entry for the output of the LOGDATA verb.

Entry point IEAVTREF is given control by PRDMP or IPCS and performs the following processing:

- Prints a header line identifying the output as the formatted contents of the LOGREC buffer.
- Locates the LOGREC buffer in the dump.
- Copies the LOGREC buffer into storage, and validates the information in the buffer header. If the header contains invalid data, the buffer contents will not be formatted.
- Loads IFCRCGIF
- Locates the oldest entry in the buffer by chaining backward through the entries.
- For each entry in the buffer:
  - If the entry is not a LOGREC request, or if it is not ready, the entry is not processed.
  - If the entry is buffered, IEAVTREF passes the buffered data to EREP for formatting.
  - If the entry is not buffered, IEAVTREF attempts to read the data into storage. If the data was available, it is passed to IFCRCGIF for formatting.

RECOVERY OPERATION:

Errors in IEAVTREF are handled by IPCS/PRDMP recovery routines. LOGDATA verb processing is terminated, and a message is issued indicating that IEAVTREF abended.

IEAVTREF - DIAGNOSTIC AIDS

ENTRY POINT NAMES: IEAVTREF
REFIO

MESSAGES:

Message texts issued from this module are:

IEA24001I LOGREC buffer could not be accessed,
possible cause - data not in dump

IEA24002I LOGREC buffer could not be formatted,
header information is invalid.

IEA24003I EREP enhancement is not available,
LOGREC entries formatted as hexadecimal data.

IEA24004I There are no LOGREC entries in the buffer.

IEA24005I Some entries could not be formatted due to
errors in the recording process.

IEA24006I This entry was incomplete at the time of the dump.

IEA24007I This entry was not buffered and may contain
invalid data

IEA24008I EREP formatting failed for this entry. It will
be formatted as hexadecimal data.

IEA24009I Processing errors encountered in EREP formatting.
Remaining entries formatted as hexadecimal data.

IEA24010I Unable to locate the next entry in the buffer.

IEA24011I A non-buffered entry could not be located -
processing continues with the next entry.

IEA24012I A non-buffered entry could not be retrieved from
the dump.

IEA24050I LOGDATA processing completed successfully.

IEA24060I LOGDATA processing terminated due to errors.

ABEND CODES: None

WAIT STATE CODES: None

RETURN CODES:

ENTRY POINT IEAVTREF:

EXIT NORMAL:

0 - Successful completion

ENTRY POINT REFIO:

EXIT NORMAL:

0 - Successful completion

8 - Not all entries were formatted

REGISTER CONTENTS ON ENTRY:

ENTRY POINT IEAVTREF:

Register 0	- Irrelevant
Register 1	- Address of the BLSABDPL
Registers 2-12	- Irrelevant
Register 13	- Address of the caller's register save area
Register 14	- Return address
Register 15	- Entry point address

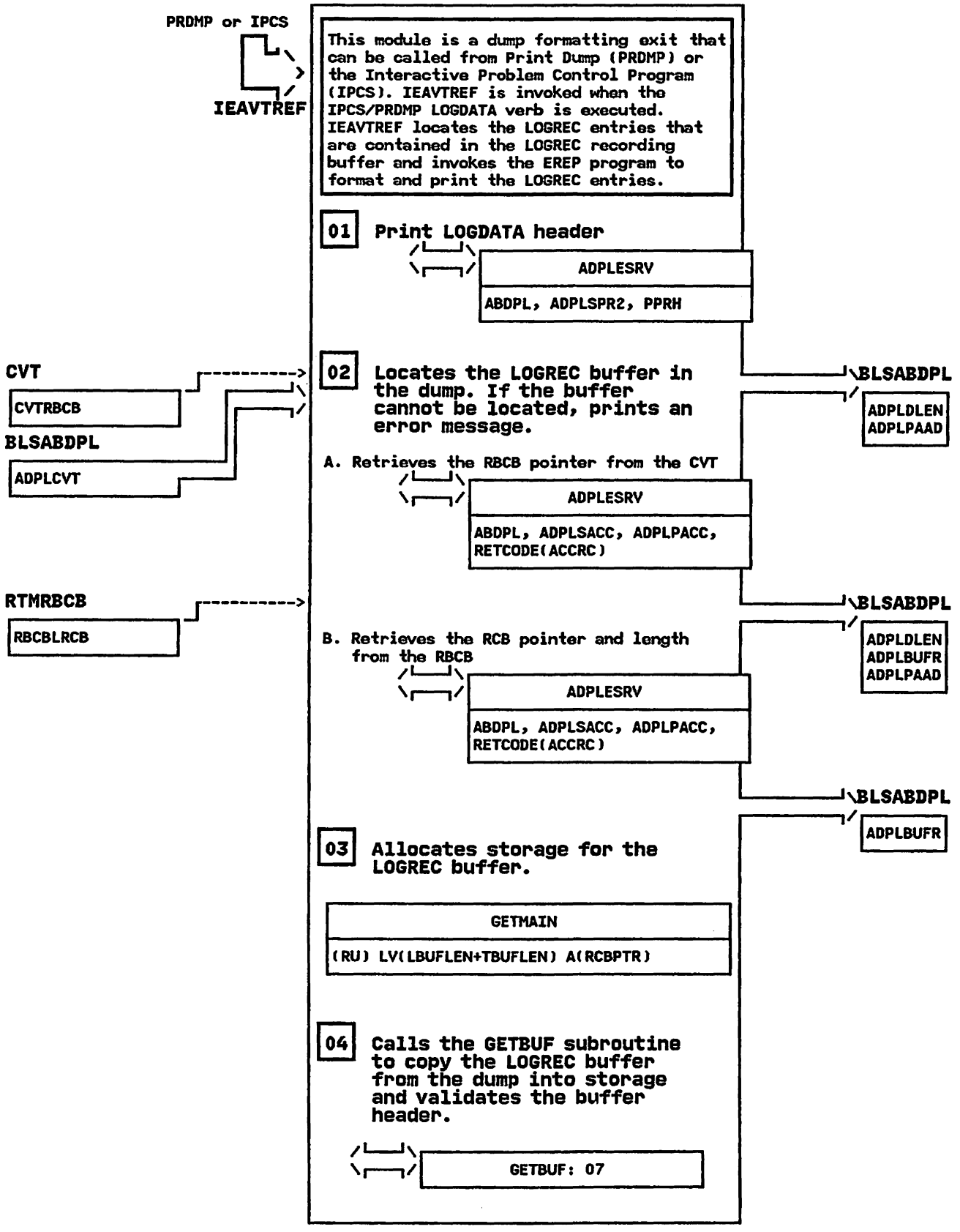
ENTRY POINT REFIO:

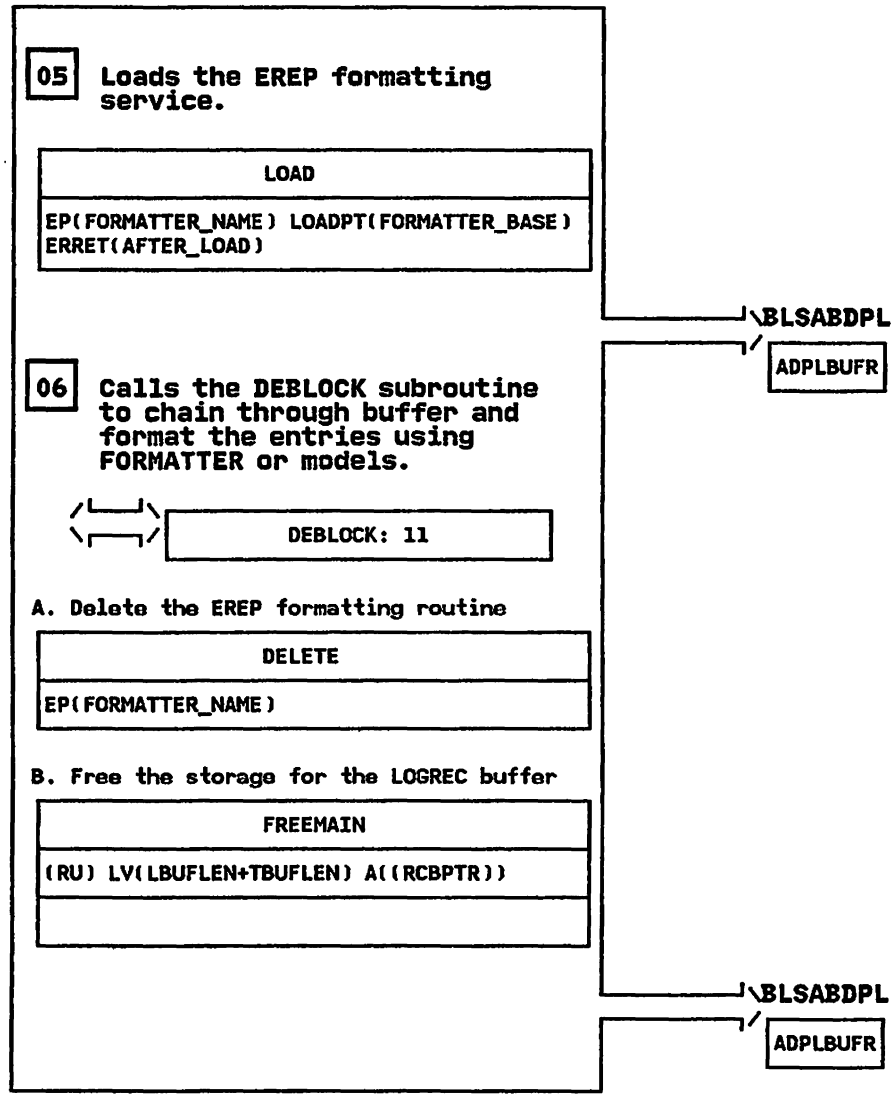
Register 0	- Irrelevant
Register 1	- Address of the parameter list
Registers 2-12	- Irrelevant
Register 13	- Address of the caller's register save area
Register 14	- Return address
Register 15	- Entry point address

REGISTER CONTENTS ON EXIT: Irrelevant

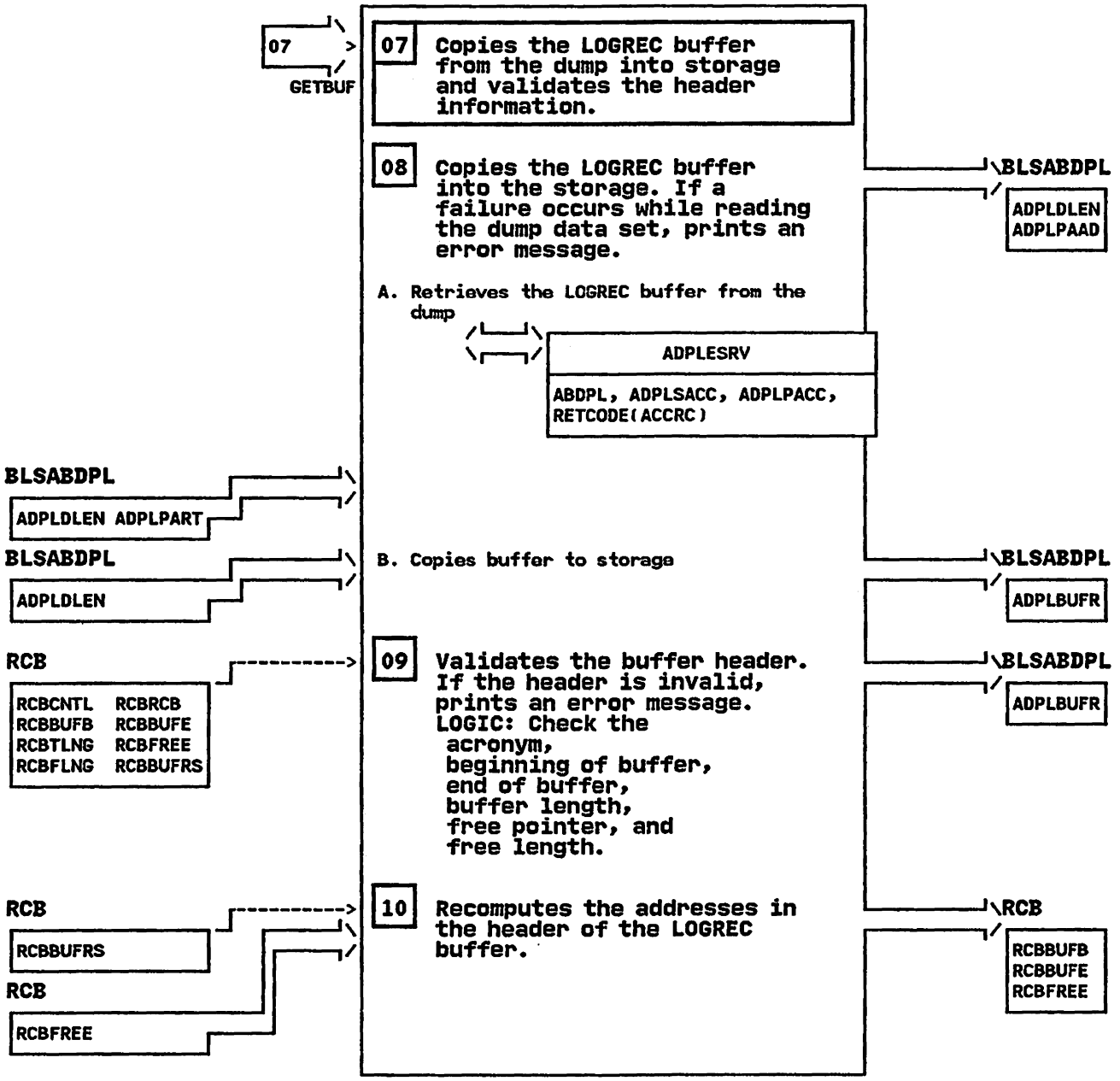
IEAVTREF - LOGREC Recording Buffer Formatter

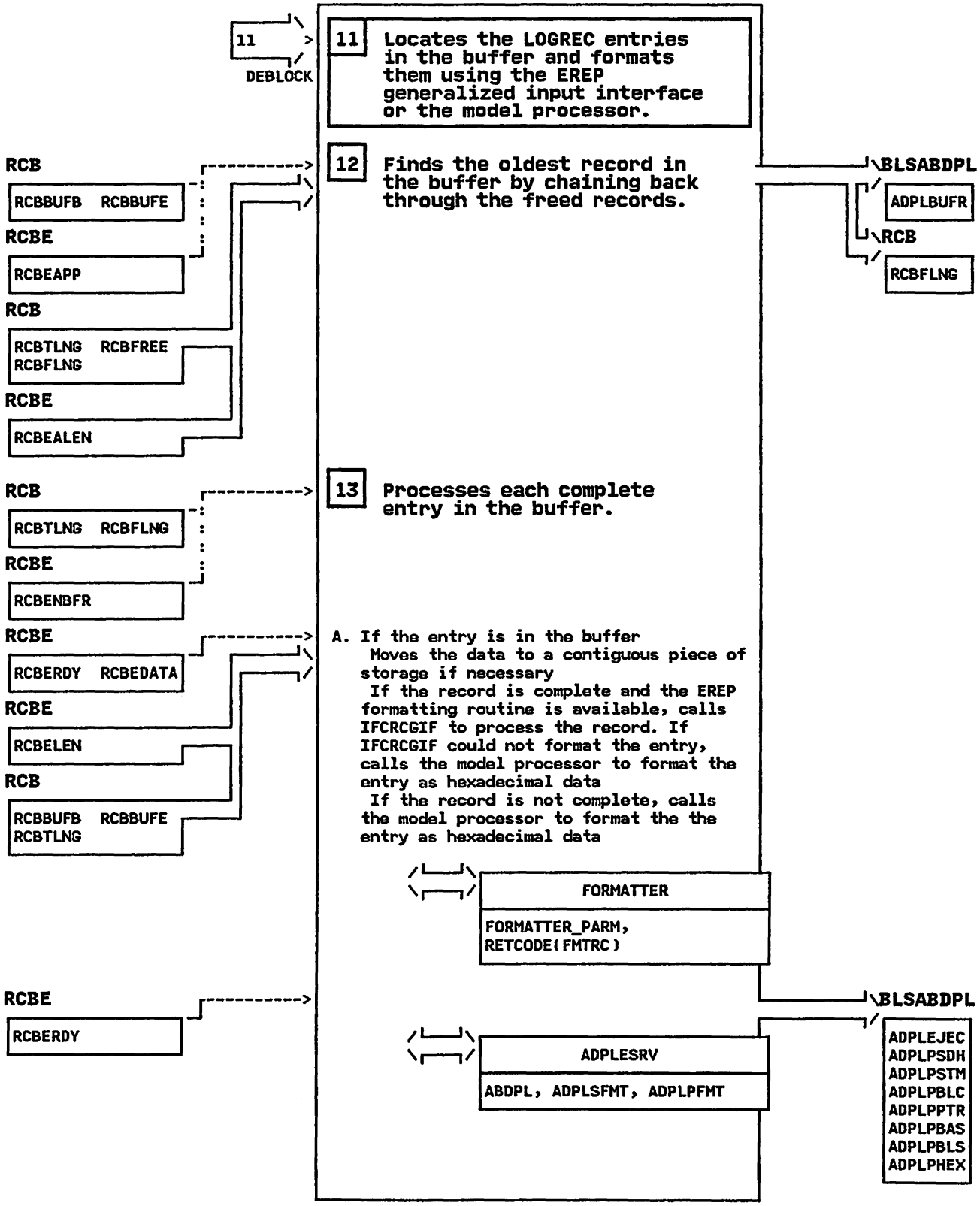
STEP 01





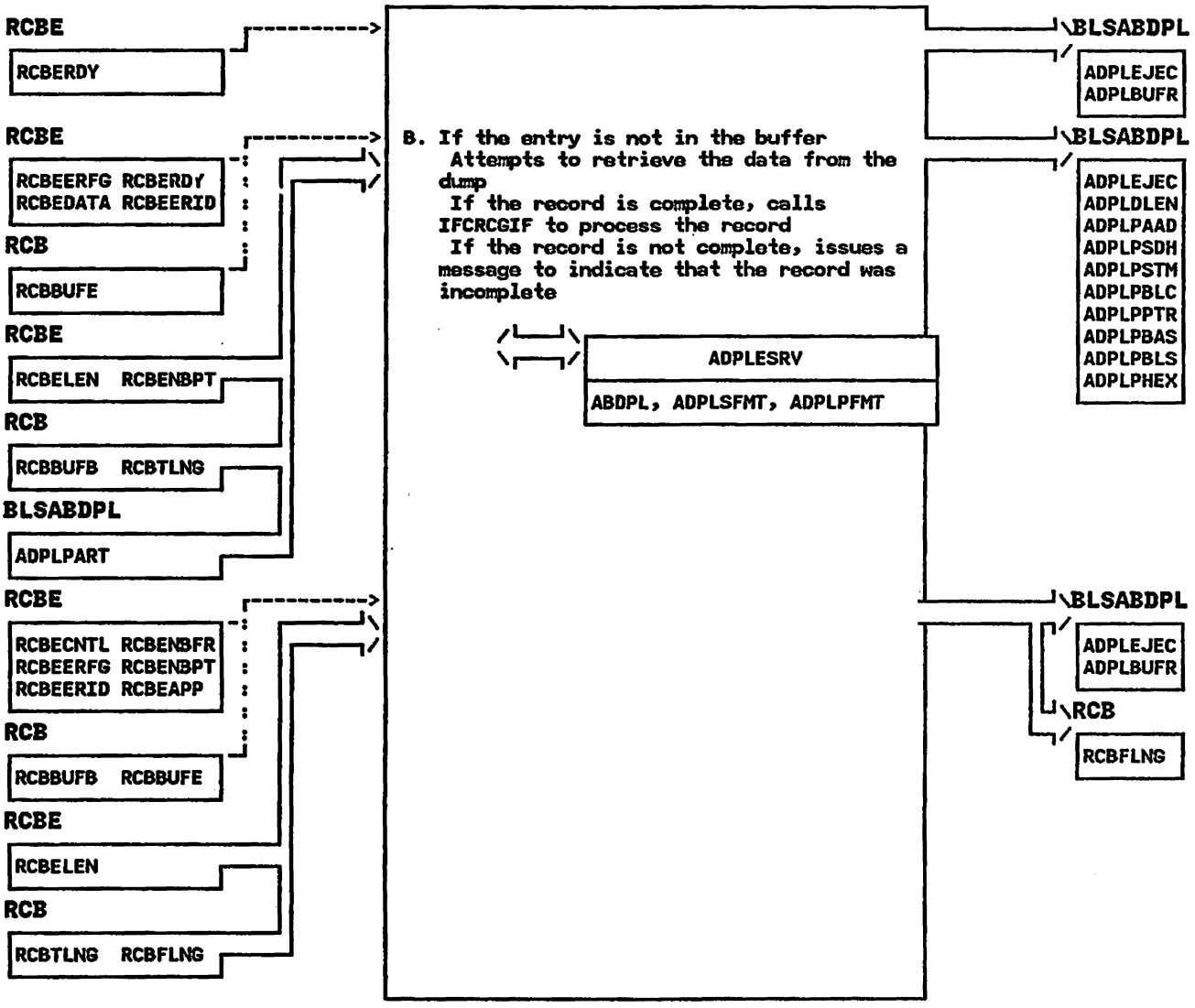
IEAVTREF - LOGREC Recording Buffer Formatter STEP 07

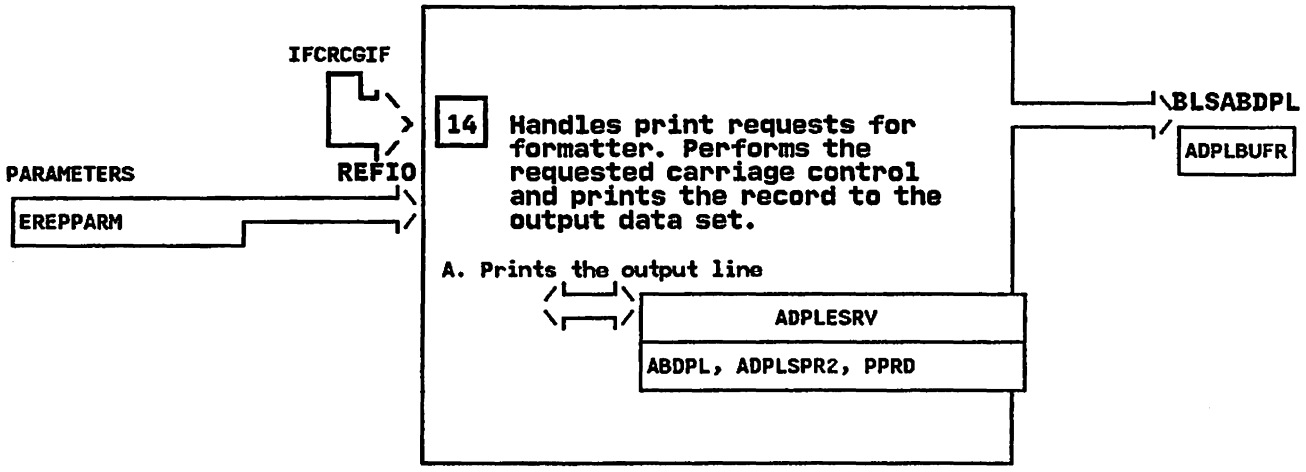




IEAVTREF - LOGREC Recording Buffer Formatter

STEP 13B





IEAVTREM - MODULE DESCRIPTION

DESCRIPTIVE NAME: Record Resource Manager

FUNCTION:
This module is a resource manager that is given control at memory termination time. IEAVTREM scans the recording control buffers (RCBs) for entries belonging to the address space being terminated. If an entry is found that has not been marked ready for termination, IEAVTREM marks the entry as invalid. IEAVTRET, the record task, does not wait for the entry to be completed and removes the entry from the buffer without processing it.

ENTRY POINT: IEAVTREM

PURPOSE: See function

LINKAGE: BALR

CALLERS: IEAVTMTC

INPUT: Resource manager's parameter list (RMPL)

OUTPUT: None

EXIT NORMAL: Returns to the caller

EXIT ERROR: No exit error conditions

ENTRY POINT: RCDRMRCV

PURPOSE:
Recovers from errors encountered during IEAVTREM's processing.

LINKAGE: Standard linkage for an ESTAE exit

CALLERS: RTM

INPUT: System diagnostic work area (SDWA)

OUTPUT: None

EXIT NORMAL: Terminates

EXIT ERROR: Percolates

EXTERNAL REFERENCES:

ROUTINES:
ESTAE service routine - Establishes the ESTAE environment.

DATA AREAS: RMPL

CONTROL BLOCKS:

Common name	Macro id	Usage	Function
CVT	CVT	read	Establishes addressability to the RBCB.
PSA	IHAPSA	read	Establishes addressability to the CVT.
RBCB	RTMRBCB	read, write	Establishes addressability to the RCB.
RCB	RTMRCB	read, write	Has recording request entries.
RCBENTRY	RTMRCBE	read, write	Maps an individual buffer entry.
RMPL	IHARMPL	read, write	Determines if an address space is terminating

IEAVTREM - MODULE DESCRIPTION (Continued)

			normally or abnormally. Also provides a work area.
SDWA	IHASDWA	read, write	Provides error information and serves as a communication area for RTM.

TABLES: No tables used

SERIALIZATION:
Serialization is required between IEAVTRET and IEAVTREM (the recording memory termination resource manager) during specific processing sections involving the recording buffers.

Module IEAVTRET obtains the local lock when:
- Releasing an entry from the recording buffers
- Processing of a temporary error

Module IEAVTREM obtains the local lock when:
- Scanning the buffer for invalid records

IEAVTREM - MODULE OPERATION

IEAVTREM receives control at memory termination time as a resource manager. IEAVTREM checks to see that entries in the recording control buffers (RCBs) that belong to the address space being terminated are marked ready for processing. If an entry is found that has not been marked ready, IEAVTREM marks the entry as invalid.

Entry point IEAVTREM receives control from IEAVTMTC and performs the following processing:

- Checks the following conditions. If one is true, returns to IEAVTMTC:
  - Address space is terminating normally.
  - The record buffers control block (RBCB) is not initialized.
  - The RECORD function encountered a permanent error.
- Establishes a recovery environment via an ESTAE.
- Attempts to serialize itself (via the local lock of masters address space) with two critical processing sections of IEAVTRET. These critical sections occur when IEAVTRET is performing temporary error cleanup and when IEAVTRET is processing an entry in the RCB.
- Scans each buffer for invalid entries, after serializing itself with IEAVTRET.
- Checks the active count in the buffer to determine if any recording requests have begun processing but have not yet completed.
  - If the active count is zero, there are no invalid entries in the buffer and IEAVTREM does not process them.
  - If the active count is not zero, IEAVTREM scans the buffer for entries that are not marked ready. If IEAVTREM finds an entry that is not ready, IEAVTREM checks whether or not the entry belongs to the address space being terminated. If it does, the entry is marked invalid. IEAVTREM continues searching until all the entries in the buffer are processed. (IEAVTREM ignores any entries that might have been put into the buffer after it began processing.)

RECOVERY OPERATION:

IEAVTREM employs an ESTAE recovery environment. If an error occurs during IEAVTREM's processing, entry point RCDRMRCV receives control. RCDRMRCV retries at RETRYPT to allow IEAVTREM to terminate normally and to allow memory termination to continue.

IEAVTREM - DIAGNOSTIC AIDS

ENTRY POINT NAMES: IEAVTREM
RCDRMRCV

MESSAGES: None

ABEND CODES: None

WAIT STATE CODES: None

RETURN CODES: None

REGISTER CONTENTS ON ENTRY:

ENTRY POINT IEAVTREM:

Register 0	- Irrelevant
Register 1	- Address of a fullword that points to the resource manager's parameter list (RMPL)
Registers 2-12	- Irrelevant
Register 13	- Address of the caller's register save area
Register 14	- Return address
Register 15	- Entry Point address

ENTRY POINT RCDRMRCV:

Register 0	- Code indication
Register 1	- Address of the SDWA or an ABEND completion code
Register 2	- Address of the parameter list specified on the ESTAE macro, if no SDWA was available
Registers 3-12	- Irrelevant
Register 13	- Address of the caller's register save area
Register 14	- Return address
Register 15	- Entry Point address

REGISTER CONTENTS ON EXIT:

ENTRY POINT IEAVTREM:

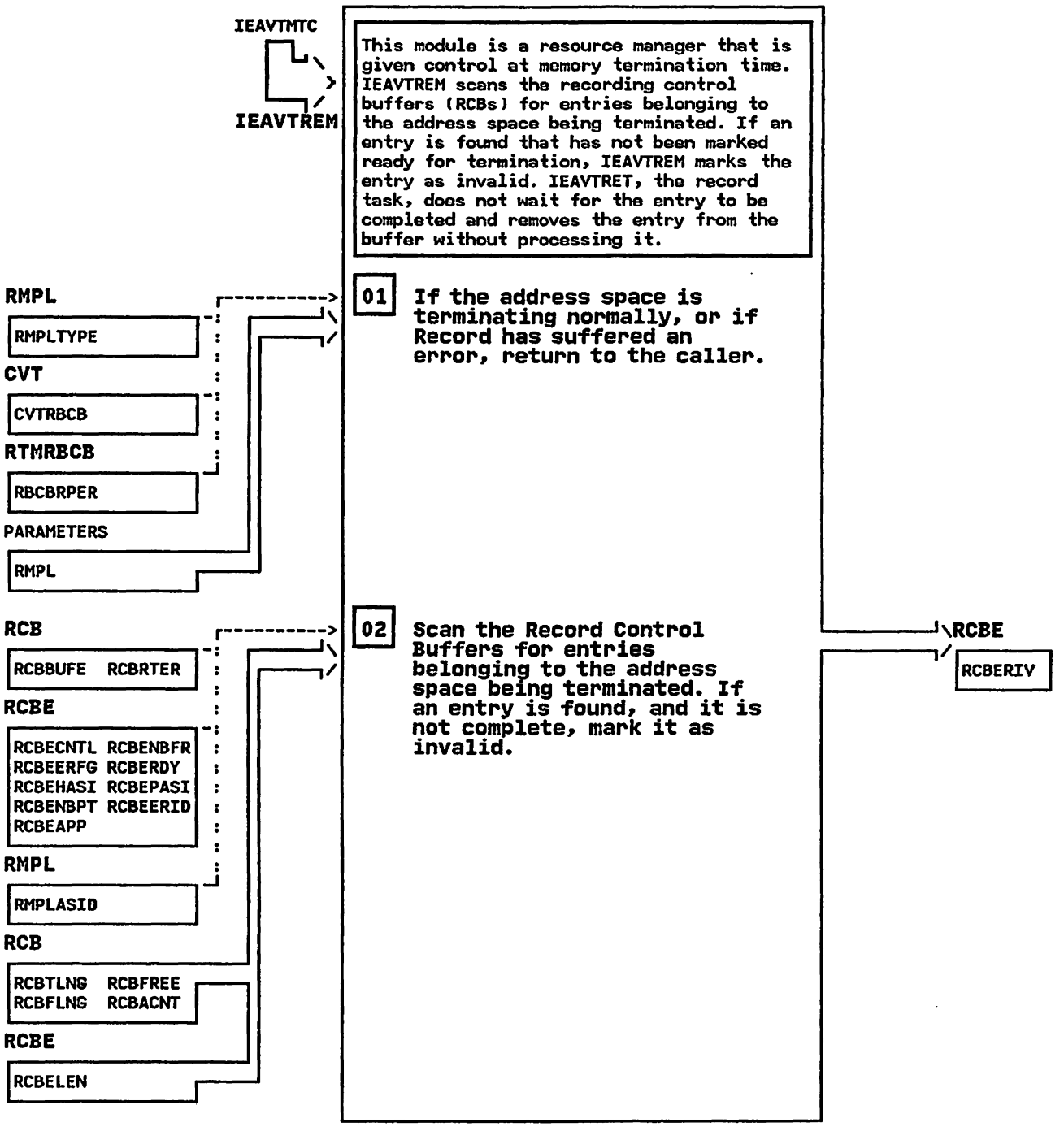
Registers 0-15 - Irrelevant

ENTRY POINT RCDRMRCV:

Registers 0-15 - Irrelevant

IEAVTREM - Record Resource Manager

STEP 01



IEAVTRER - MODULE DESCRIPTION

DESCRIPTIVE NAME: Record Request Routine

FUNCTION:
This module determines whether the caller requested the recording function (via the RECORD macro) for an emergency request (by specifying TYPE=TERM) or for a non emergency request (by specifying TYPE=LOGREC or TYPE=WTO) and prepares the buffer for recording of the error asynchronously under the recording task, IEAVTRET.

ENTRY POINT: IEAVTRER

PURPOSE:
Determines whether this is an emergency request (TYPE=TERM) or a non emergency request (TYPE=LOGREC) or (TYPE=WTO) and prepares the buffer area for the recording of errors.

LINKAGE: BSM from glue module IEAVTRGR

CALLERS:
Issuers of the RECORD macro
.that specify TYPE=LOGREC or TYPE=WTO (any key 0, supervisor routine using module IEAVTRGR)
.that specify TYPE=TERM (only module IGFPEMER)

INPUT:
Request Options (contained in register 0, bytes 0-1)
byte 0
bit 0 - Emergency request
bit 1 - Write to LOGREC
bit 2 - Write to operator
bits 3 - 5 - Reserved
bit 6 - Free unbuffered storage
bit 7 - Include errorid
byte 1
bit 0 - Recovery via ESTAE
(the recovery flags are no longer used, but have not been removed for compatability reasons)
bit 1 - Recovery via SETFRR
(the recovery flags are no longer used, but have not been removed for compatability reasons)
bit 2 - Buffer the data
bit 3 - Do not buffer the data
(if bits 2,3 both zero - Buffer if space available)
bit 4 - Prefix data with standard LOGREC header
bit 5 - Follow header with jobname
bit 6 - Return time stamp
bit 7 - POST caller when I/O completes

Length (contained in register 0, bytes 2-3)

RECORD Parameter list (address in register 1)
Header info - If header requested
Jobname - If jobname requested
ERRORID - If error id requested
ECB - If posting requested
ASID - If posting requested
Timeadr - If time stamp to be returned
Subpool - If freeing of unbuffered storage requested, subpool of user's buffer
Dataadr - Address of data to be recorded

Data (address in register 1)

OUTPUT: Register 15 contains a return code

IEAVTRER - MODULE DESCRIPTION (Continued)

EXIT NORMAL: Returns to the caller

EXIT ERROR: Returns to the caller

ENTRY POINT: RCDRCVR

PURPOSE: Recovers from errors encountered during IEAVTRER.

LINKAGE: Standard linkage for an FRR exit

CALLERS: RTM

INPUT: System diagnostic work area (SDWA)

OUTPUT: Retry/percolation options in the SDWA

EXIT NORMAL: Requests retry.

EXIT ERROR: Requests percolation.

ENTRY POINT: RERSRBEP

PURPOSE:
Issues the POST macro instruction for the ECB waited on by the Recording Task, IEAVTRET.

LINKAGE: Via schedule SRB from mainline

CALLERS: Dispatcher

INPUT: None

OUTPUT: None

EXIT NORMAL: Returns to the dispatcher.

EXIT ERROR: There are no exit error conditions.

EXTERNAL REFERENCES:

ROUTINES:
The Post service, via the POST macro
The Schedule service, via the SCHEDULE macro
The Setfrr service, via the SETFRR macro
The Setlock service, via the SETLOCK macro

DATA AREAS: There are no data areas

CONTROL BLOCKS:

Common name	Macro ID	Usage	Function
ASCB	IHAASCB	read	Contains the ASID of the caller's HOME address space.
CVT	CVT	read	Establishes addressability to the RBCB.
PSA	IHAPSA	read	Establishes addressability to the CVT.
RBCB	RTMRBCB	read, write	Establishes addressability to the RCBs.
RCB	RTMRCB	read, write	Contains the entries for the recording requests.
RCBENTRY	RTMRCBE	read, write, create, delete	Contains the information for a single recording request.
SDWA	IHASDWA	read, write	Provides error information and serves as a communication area for RTM.

IEAVTRER - MODULE DESCRIPTION (Continued)

SRB	IHASRB	read, write	Requests system processing.

SERIALIZATION:

IEAVTRER uses CS and CDS logic to serialize its use of the Recording buffers with IEAVTRET, IEAVTREM, and other concurrent recording requests.

IEAVTRER uses the CPU lock to obtain disablement during certain sections of critical processing. This is necessary to ensure that a MEMTERM does not interrupt this processing and leave the Recording buffers in an inconsistent state. If IEAVTRER is entered disabled, the CPU lock is not obtained.

IEAVTRER - MODULE OPERATION

IEAVTRER receives control after a system routine issues the RECORD macro instruction specifying the type of recording request in the parameters.

For an emergency request (TYPE=TERM), IEAVTRER performs the following:

- Removes the oldest LOGREC entry from the chain of entries ready to be written.
- Places the entry data into the caller's area specified by the DATAADR RECORD macro parameter.
- If the entry data is larger than the caller's area, IEAVTRER truncates the messages and sets a return code of 8. Otherwise, a return code of zero is set to indicate success.
- Places the length of the data in register 0.
- Returns to the caller. If there are no scheduled LOGREC requests, IEAVTRER sets a return code of 4. (Note: System termination will repeatedly call until a return code of 4 is issued.)

For a non emergency request (TYPE=LOGREC or TYPE=WTO), IEAVTRER performs the following:

- Establishes recovery via the SETFRR macro instruction.
- Determines whether the request is to write to the LOGREC dataset (TYPE=LOGREC) or to send messages to the operator (TYPE=WTO).
  Note.There are two recording buffers, one for writing data to the LOGREC dataset and one for writing messages to the operator.
- Computes the amount of buffer space required by the request as follows (buffer space is allocated in double word multiples):
  16 bytes for Record entry header (mapped by RTMRCBE)
  + 24 bytes if a standard LOGREC header must be built
  + 8 bytes if jobname is to be added
  + the length of the data supplied or 4 bytes for an address if the record is not to be buffered
  + 10 bytes if the errorid is to be added
  + 8 bytes for the entry trailer
- Determines if the buffer space partition for this type of entry have been exceeded. If so, the request is denied.
- Determines if sufficient space is available for the entry from the RCBFLNG field of the RCB.
  If sufficient space is available,
  - Allocates the space for this entry
  - Builds the entry
  - Schedules an SRB that will POST the ECB waited on by the recording task, IEAVTRET
  - Issues a return code of zero indicating the request has been scheduled normally

  If the available space is insufficient and this is a buffered LOGREC request,
  - Allocates the available space for this entry
  - Builds a truncated entry and sets a return code of eight indicating the data has been truncated because of insufficient space
    (A truncated record must consist minimally of the header information, the jobname, and the errorid.)

IEAVTRER - MODULE OPERATION (Continued)

If a truncated entry can not be built, IEAVTRER sets a return code of 12 indicating the request has not been scheduled because of a lack of buffer space.

If the available space is insufficient and this is a buffered WTO request,
- Sets a return code of 12 indicating the request has not been scheduled because of a lack of buffer space.

.Returns control to the caller.

RECOVERY OPERATION:

IEAVTRER employs an EUT FRR recovery environment. If an error occurs during IEAVTRER's processing, entry point RCDRCVR receives control. RCDRCVR sets a flag in the RCB indicating to the Recording Task, IEAVTRET, that recovery action is required. Subsequent requests involving this buffer are denied until the buffer has been reinitialized.

IEAVTRER - DIAGNOSTIC AIDS

ENTRY POINT NAMES: IEAVTRER
RCDRCVR
RERSRBEP

MESSAGES: None

ABEND CODES: None

WAIT STATE CODES: None

RETURN CODES:

ENTRY POINT IEAVTRER:

EXIT NORMAL:

0 - Successful completion
4 - For TYPE = WTO or TYPE = LOGREC, a request for optional buffering has not been buffered but scheduled directly from the callers buffer.
For emergency requests no more messages.
8 - Record for LOGREC is truncated

EXIT ERROR:

12 - A record has been lost because of the lack of space.
16 - A record has been lost because of processing errors.
20 - The recording request facility is inactive.

ENTRY POINT RCDRCVR:

EXIT NORMAL:

4 - Retry to mainline

EXIT ERROR:

0 - Percolate to the next level of recovery

ENTRY POINT RERSRBEP: None

REGISTER CONTENTS ON ENTRY:

ENTRY POINT IEAVTRER:

Register 0 - Request options and length
Register 1 - Address of data
Registers 2-12 - Irrelevant
Register 13 - Address of the caller's register save area
Register 14 - Return address
Register 15 - Entry Point address

ENTRY POINT RCDRCVR:

Register 0 - Address of a 200 byte FRR work area
Register 1 - Address of the SDWA
Registers 2-13 - Irrelevant
Register 14 - Return address
Register 15 - Entry Point address

ENTRY POINT RERSRBEP:

IEAVTRER - DIAGNOSTIC AIDS (Continued)

Register 0 - Irrelevant
Register 1 - Base register (SRB parameter pointer)
Registers 2-13 - Irrelevant
Register 14 - Return address
Register 15 - Irrelevant

REGISTER CONTENTS ON EXIT:

ENTRY POINT IEAVTRER:

Register 0-1 - Irrelevant
Registers 2-14 - Restored
Register 15 - Return Code

ENTRY POINT RCDRCVR:

Register 0-15 - Irrelevant

ENTRY POINT RERSRBEP:

Register 0-15 - Irrelevant

IEAVTRER - Record Request Routine

STEP 01

Issuers of the RECORD macro
.that specify TYPE=LOGREC or TYPE=WTO (any key 0, supervisor routine using module IEAVTRGR)
.that specify TYPE=TERM (only module IGFPEMER)

IEAVTRER

This module determines whether the caller requested the recording function (via the RECORD macro) for an emergency request (by specifying TYPE=TERM) or for a non emergency request (by specifying TYPE=LOGREC or TYPE=WTO) and prepares the buffer for recording of the error asynchronously under the recording task, IEAVTRET.

CVT
- CVTRBCB

RTMRBCB
- RBCBRPER

01 If the Recording Facility is not initialized or has suffered an unrecoverable error, sets the return code to 20 and returns.

RCBE
- RCBECNTL RCBENBFR RCBEERFG RCBEDATA RCBEAPP

RTMRBCB
- RBCBLRCB

RCB
- RCBBUFB RCBBUFE RCBTLNG RCBFREE RCBFLNG

RCBE
- RCBELEN RCBENBPT RCBEERID

02 If the request is for emergency processing, returns the oldest entry in the LOGREC RCB in the caller's buffer.

RTMRBCB
- RBCBRPER

RCBE
- RCBEERID

RCB
- RCBFLNG

PSA
- PSANSTK

RTMRBCB
- RBCBLRCB RBCBWRCB

PSA
- PSACSTK

03 Establishes an FRR recovery envirnoment and initializes the FRR parameter area.

PSA
- PSACSTK

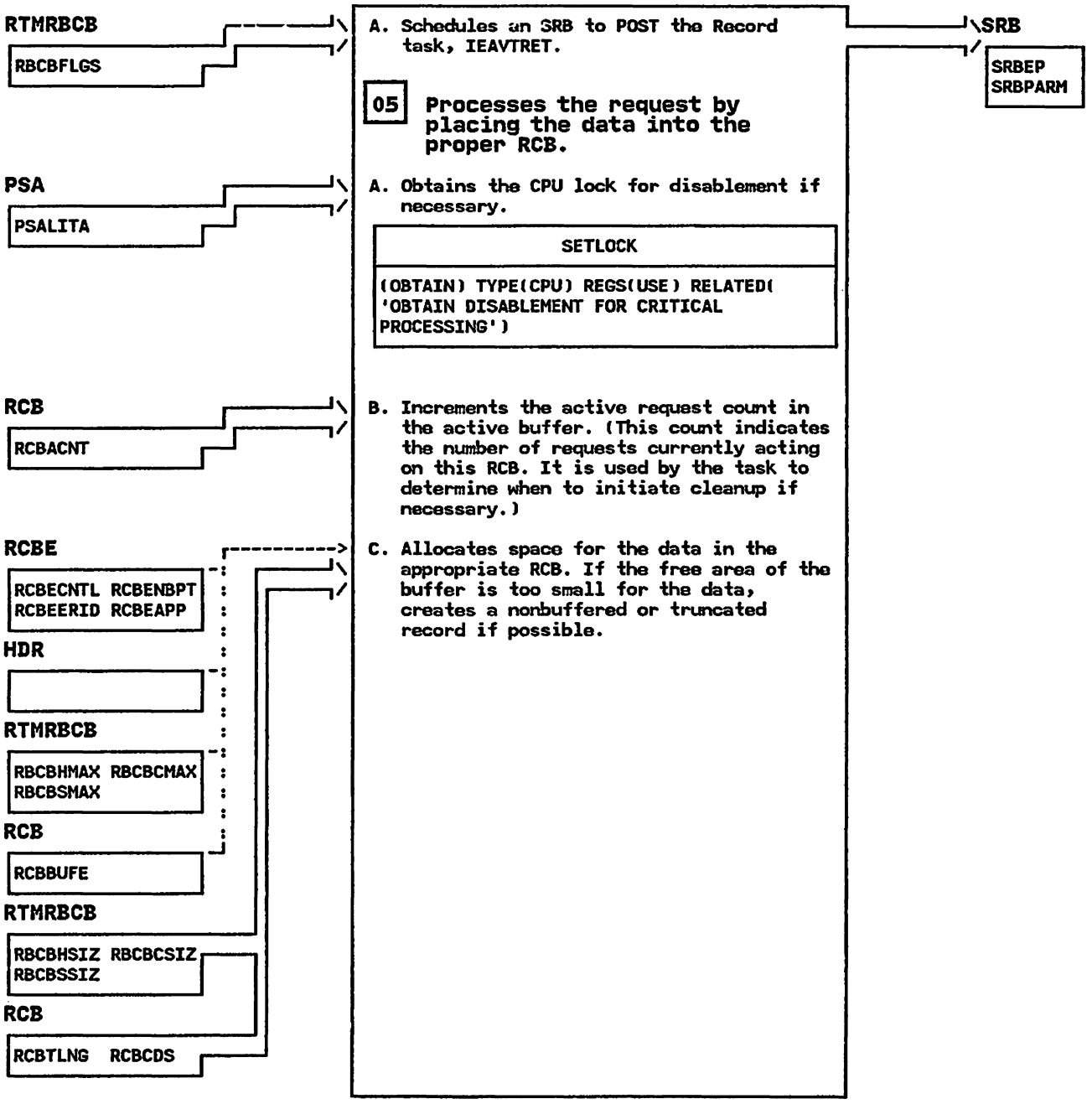
PSA
- PSANSS

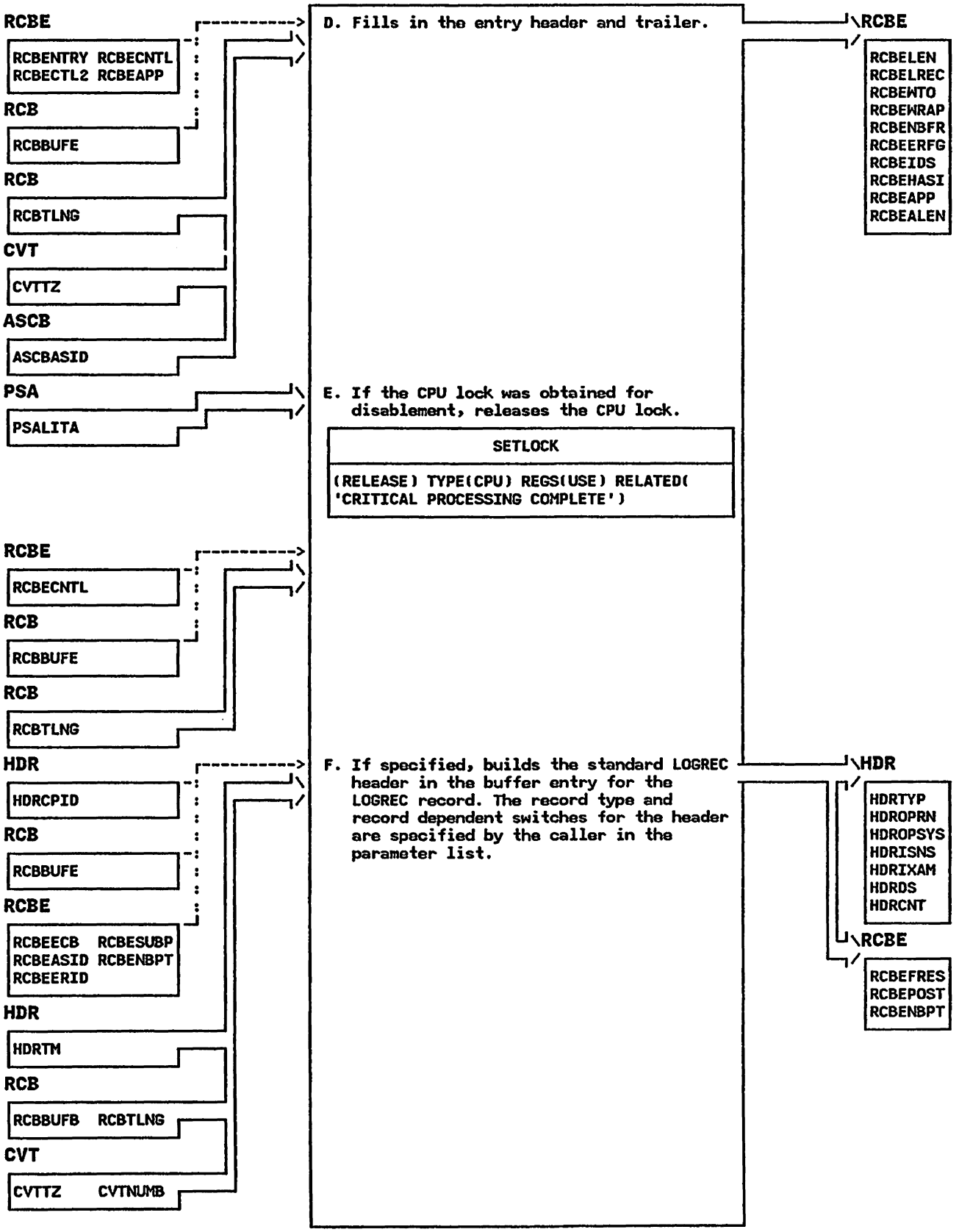
RCB
- RCBRTER

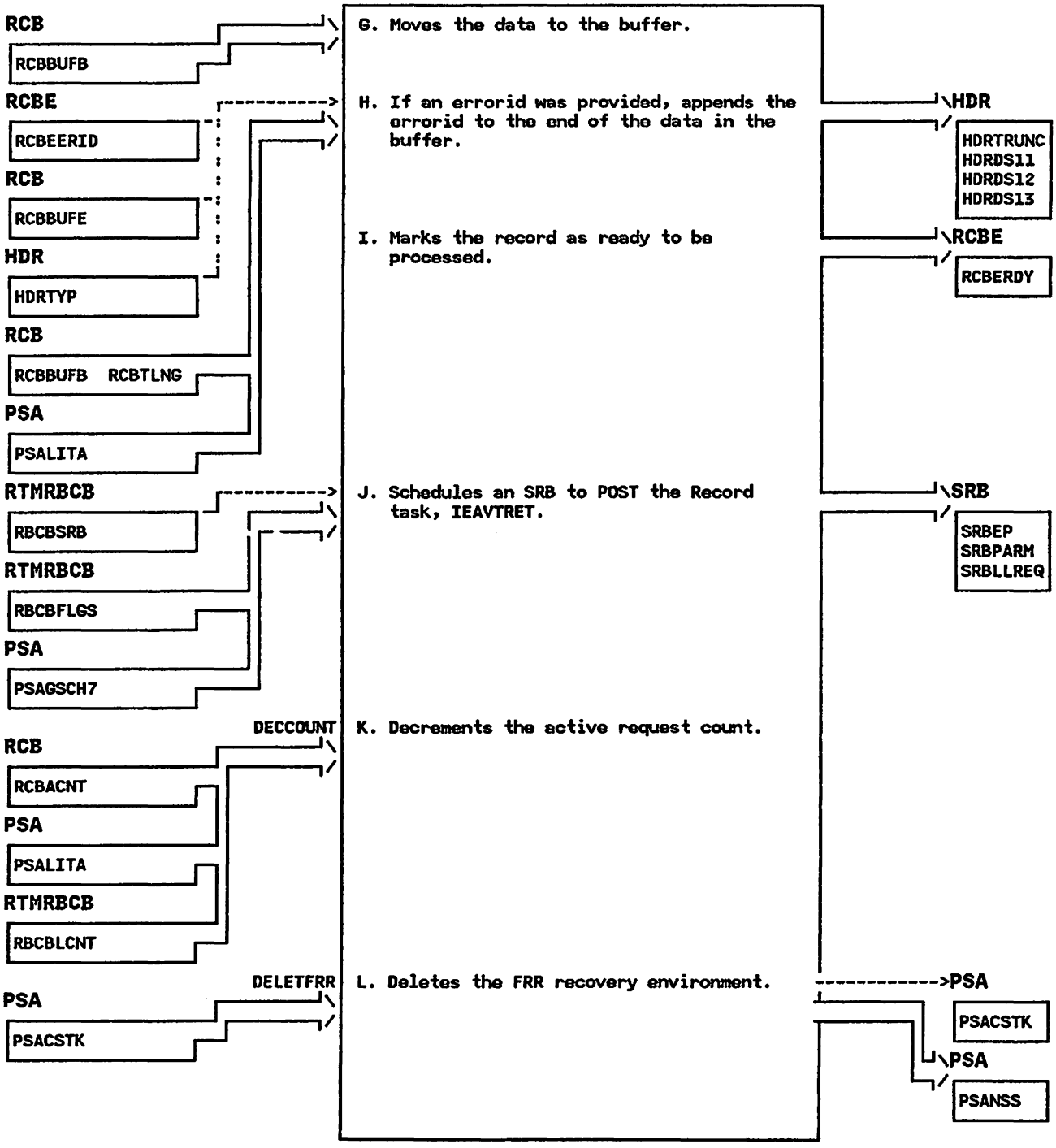
04 If the buffer for this request is marked unusable, posts the Recording task for cleanup processing and sets the return code to indicate that the request could not be scheduled because of processing errors.

IEAVTRER - Record Request Routine

STEP 04A

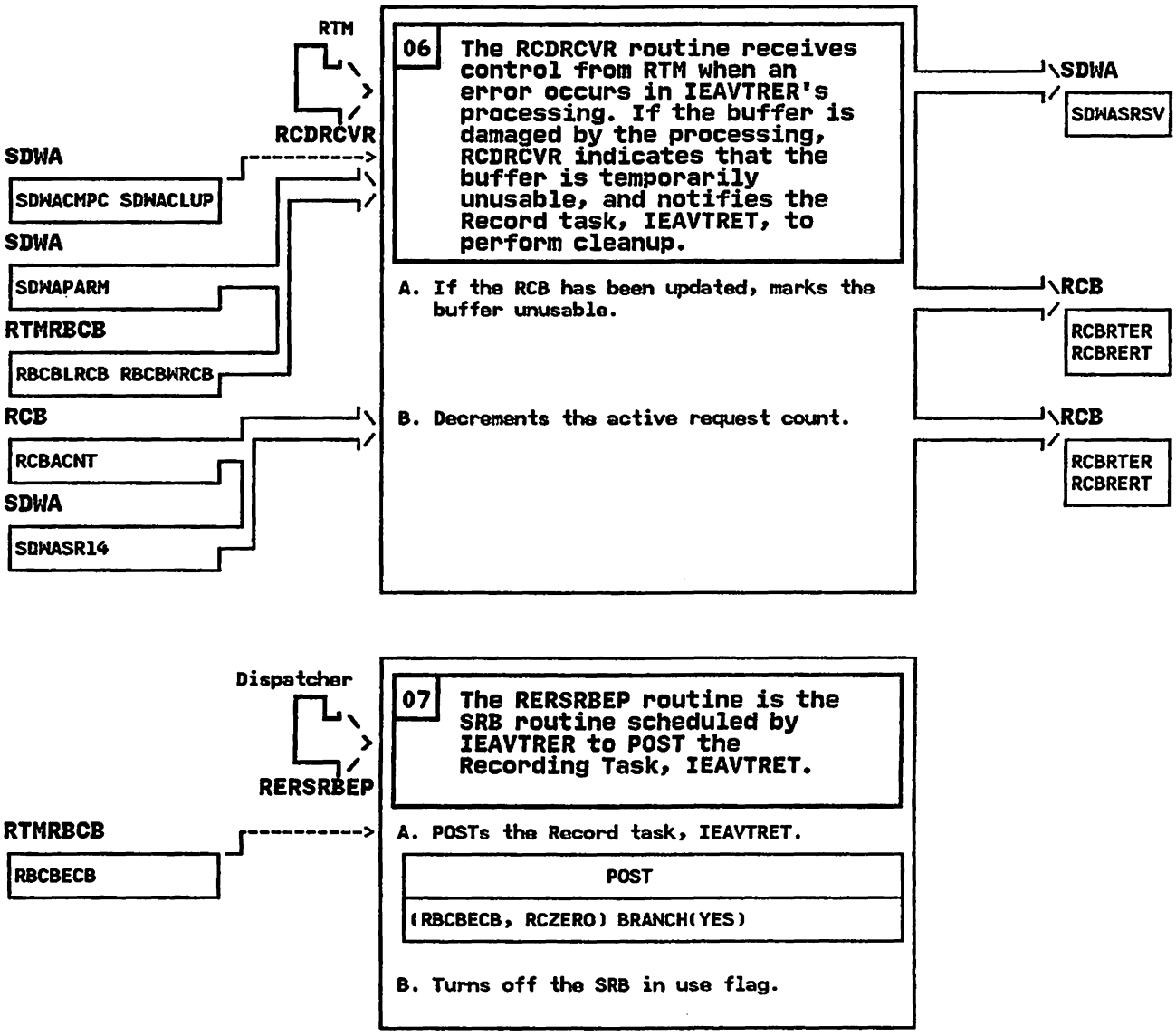






IEAVTRER - Record Request Routine

STEP 06



IEAVTRET - MODULE DESCRIPTION

DESCRIPTIVE NAME: Recording Task

FUNCTION:
This module writes records to SYS1.LOGREC or to the operator which have been scheduled via the RECORD macro. If requested, IEAVTRET will free unbuffered storage and/or post the caller after the requested processing is complete. IEAVTRET will write a lost record summary to LOGREC if records were lost, and it passes control to the MCH WTO routine for an MCH operator message (decision on whether or not a message is required is made by the MCH routine.)

ENTRY POINT: IEAVTRET

PURPOSE: See function

LINKAGE: Attach macro

CALLERS: IEAVTMSI

INPUT: Attachor's ECB

OUTPUT: None

EXIT NORMAL:

EXIT ERROR:

OUTPUT: Attachors ECB is posted with error code

ENTRY POINT: RCDTSKR

PURPOSE: Recovers from errors encountered during IEAVTRET

LINKAGE: Standard linkage for an ESTAE exit

CALLERS: RTM

INPUT: System diagnostic work area (SDWA)

OUTPUT: Retry/percolation options in the SDWA

EXIT NORMAL: Request retry

EXIT ERROR: Percolates

EXTERNAL REFERENCES:

ROUTINES:
- ESTAE service - establishes the ESTAE environment
- GETMAIN service - obtains storage for the work buffer
- FREEMAIN service - frees the caller's buffer
- MCH WTO service (IGFWMSG) - writes machine check messages to the operator if required
- PGSER service - releases the work buffer after processing
- WAIT service - waits for work to be done
- WTO service - writes messages to the operator
- POST service - signals the user that the request is complete
- SETLOCK service - obtains the local lock for serialization

CONTROL BLOCKS:

Common name	Macro ID	Usage	Function
ASCB	IHAASCB	read	Contains the ASID of the current address space
ASVT	IHAASVT	read	Indicates if an ASID is valid for posting
CVT	CVT	read	Establishes addressability

IEAVTRET - MODULE DESCRIPTION (Continued)

			to the RBCB.
LRB	IHALRB	read, write	MCH record header mapping
PSA	IHAPSA	read	Establishes addressability to the CVT.
RBCB	RTMRBCB	read, write	Establishes addressability to the RCBs
RCB	RTMRCB	read, write	Contains the entries for the recording requests
RCBENTRY	RTMRCBE	read, write, delete	Contains the information for a single recording request
SDUMP	IHASDUMP	read	SDUMP parm list
SDWA	IHASDWA	read, write	Provides error information and serves as a communication area for RTM.
SRB	IHASRB	read, write	SRB for posting IEAVTRET when a request is processed

SERIALIZATION:

Serialization is required between IEAVTRET and IEAVTREM (the recording memory termination resource manager) during specific processing sections involving the Recording buffers.

Module IEAVTRET obtains the local lock to:

- Release an entry from the Recording buffers
- Process a temporary error

Module IEAVTREM obtains the local lock to:

- Scan the buffer for invalid records

IEAVTRET - MODULE OPERATION

This module is a never ending task which runs in Master's address space.
IEAVTRET is given control by IEAVTMSI via the ATTACH macro.

When IEAVTRET is attached, it performs the following processing.

- Establishes recovery via ESTAE
- Builds an SRB in the RBCB
- Obtains storage for rebuffering of the RCBs

If any errors occur in this initialization phase, IEAVTRET posts the attachor with a return code of 4, and goes into a wait state.

Otherwise, the attachor is posted with a return code of zero to indicate successful initialization.

After successful initialization, the task processes any records that may have already been put into its buffer. It then goes into a wait state.

When IEAVTRET is posted, it performs the following processing for each Recording buffer:

- Moves all ready records to the work buffer
- Writes the records to SYS1.LOGREC or the operator
- If requested, frees the users buffer and/or posts the users ECB specified in the request

If records have been lost as indicated by the RBCBLCNT field of the RBCB, the lost record indication is written to SYS1.LOGREC.

IEAVTRET then calls the MCH WTO routine to issue an operator message related to MCH LOGREC records. All logic relative to whether or not a message is required is performed by the MCH routine.

The task returns to the wait state.

RECOVERY OPERATION:

IEAVTRET employs an ESTAE recovery environment. If an error occurs during IEAVTRET's processing, entry point RCDTSKR receives control. Each time an external routine is entered the address of the instruction following the routine and required interface is stored in the ESTAE parmlist. If we are percolated to by an external routine we retry at the point following this routine.

RCDTSKR determines if it was in an external routine if so retry is effected following the call to this routine. If the error was an operation exception (SDWAABCC) or if record has already suffered a temporary error (RCBRTER), a dump of the code and work areas is taken, recording is turned off, and a message is written to the operator to this effect. If the error was not an operation exception, the interface is established from the ESTAE parmlist, and retry is initiated at at the point where the task wakes up from the wait state.

IEAVTRET - DIAGNOSTIC AIDS

ENTRY POINT NAMES: IEAVTRET
RCDTSKR

MESSAGES: IEA896I - RECORDING FUNCTION NO LONGER ACTIVE

ABEND CODES: None

WAIT STATE CODES: None

RETURN CODES:

ENTRY POINT IEAVTRET:

0 - Successful completion

4 - Initialization failed

ENTRY POINT RCDTSKR:

EXIT NORMAL:

4 - Retry to recording task mainline

EXIT ERROR:

0 - Percolate to the next level of recovery

REGISTER CONTENTS ON ENTRY:

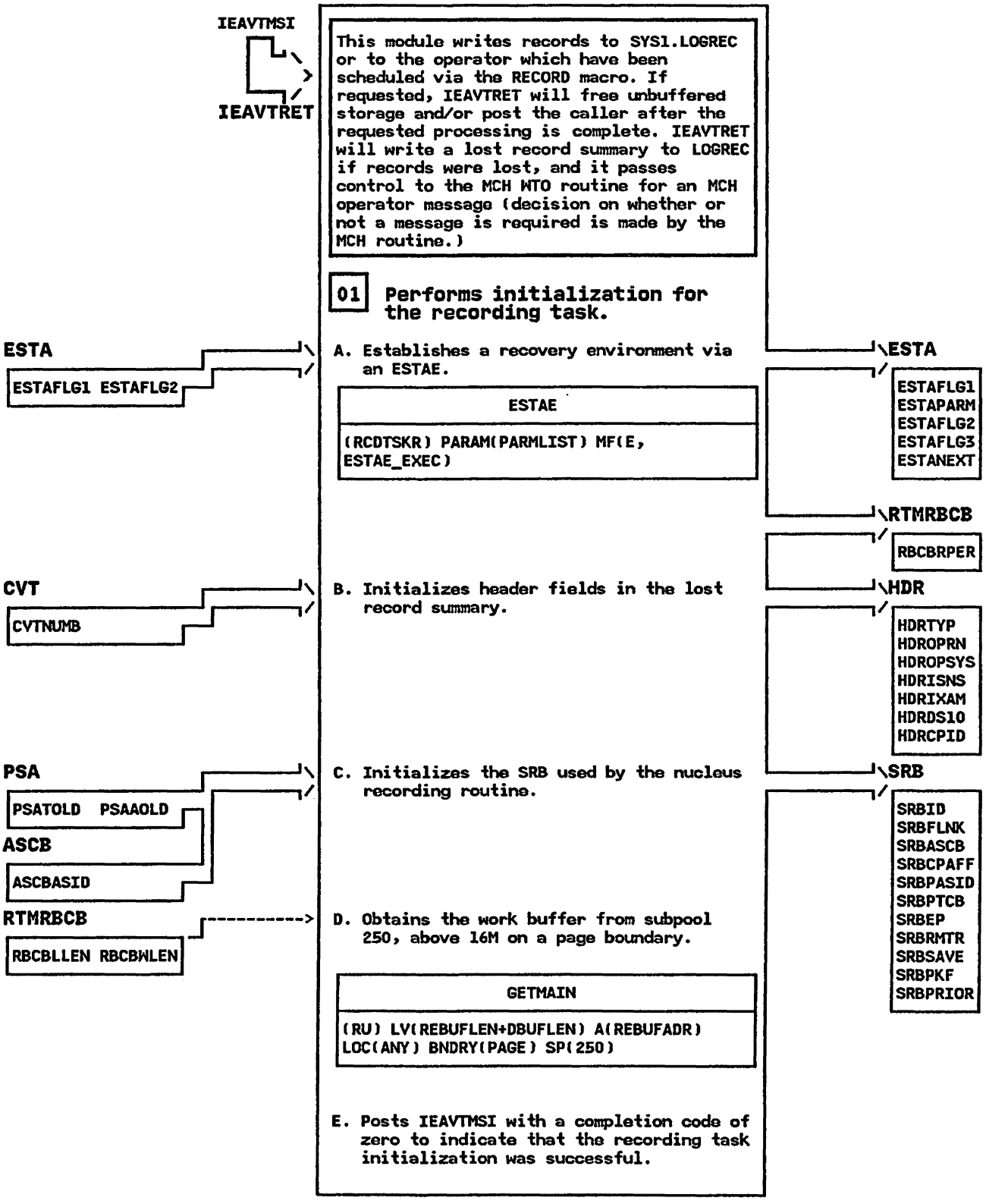
ENTRY POINT IEAVTRET:

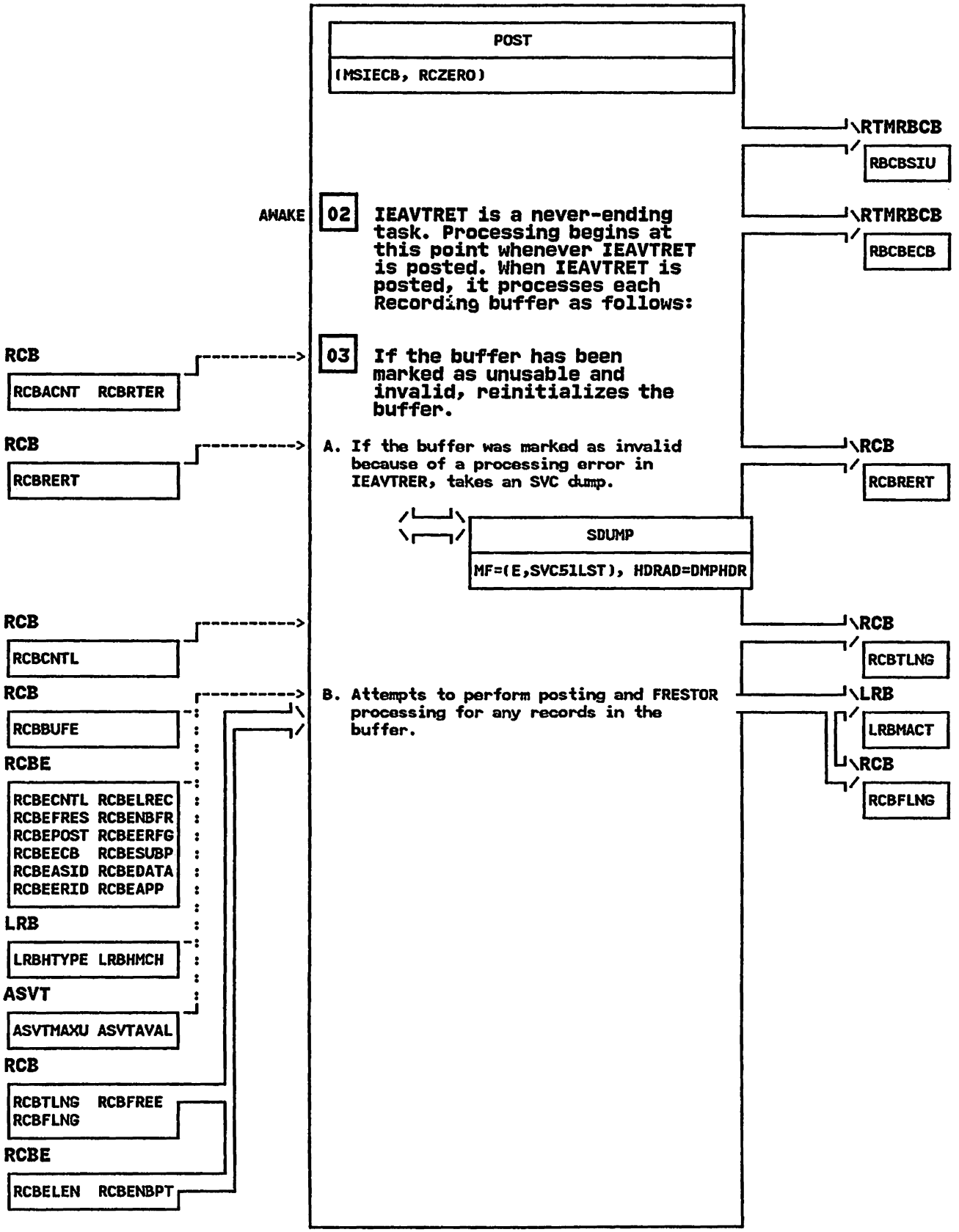
Register	0	- Irrelevant
register	1	- Address of the attachor's ECB
registers	2-12	- Irrelevant
register	13	- Address of the caller's register save area
register	14	- Return address
register	15	- Entry point address

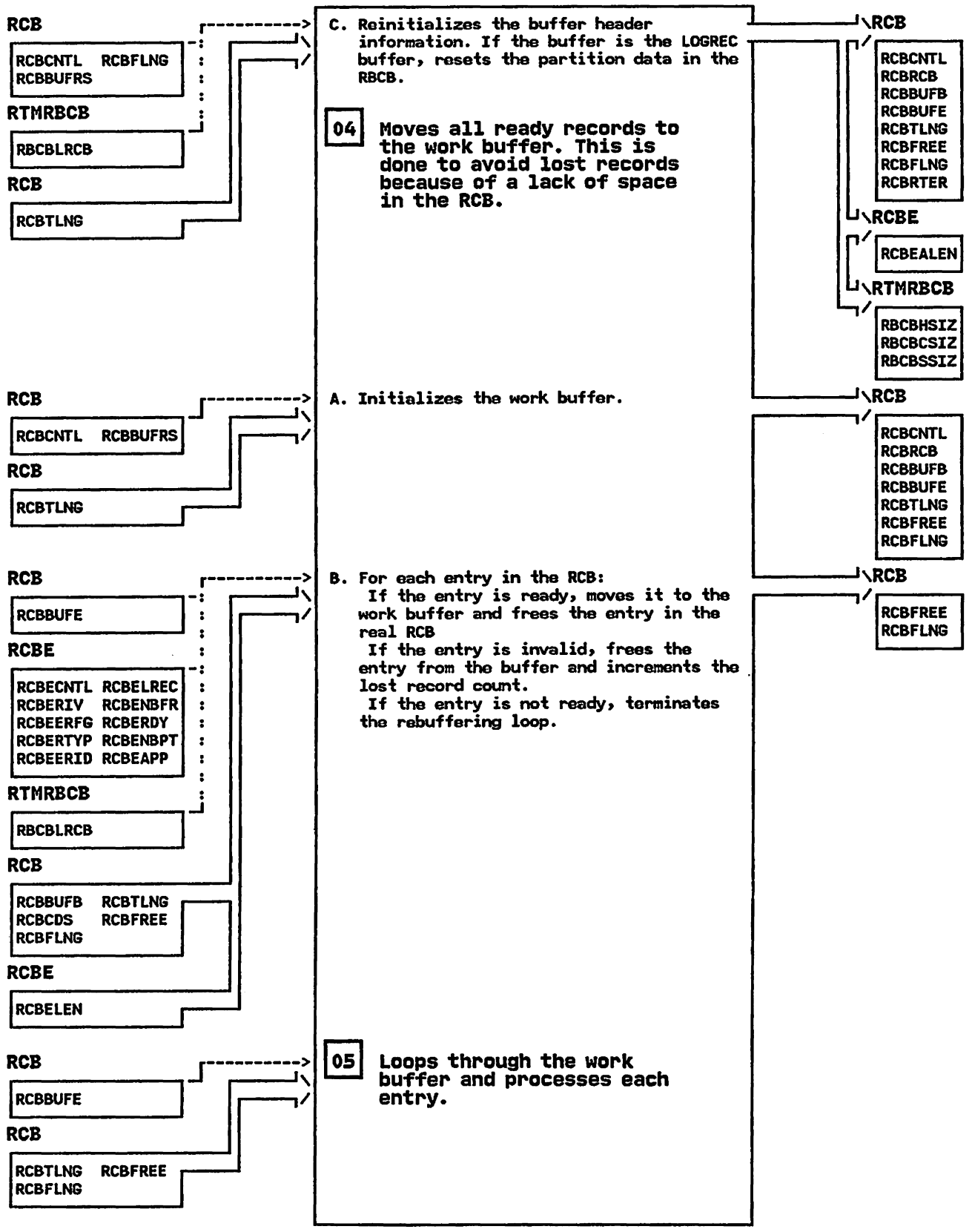
ENTRY POINT RCDTSKR:

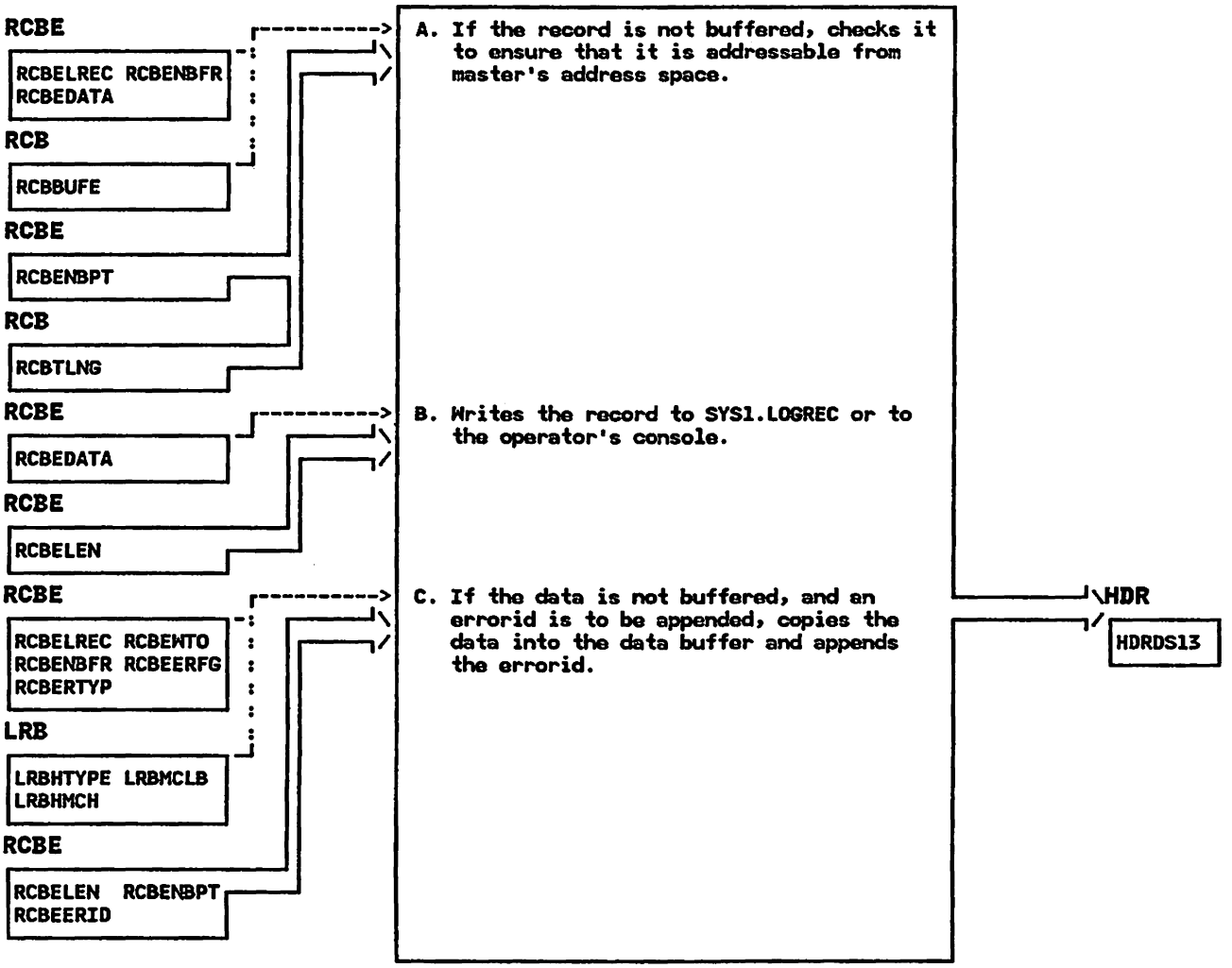
Register	0	- Code indication
register	1	- Address of the SDWA or an ABEND completion code
registers	2	- Address of the recovery parameter area if no SDWA was available
registers	3-12	- Irrelevant
register	13	- Address of the caller's register save area
register	14	- Return address
register	15	- Entry point address

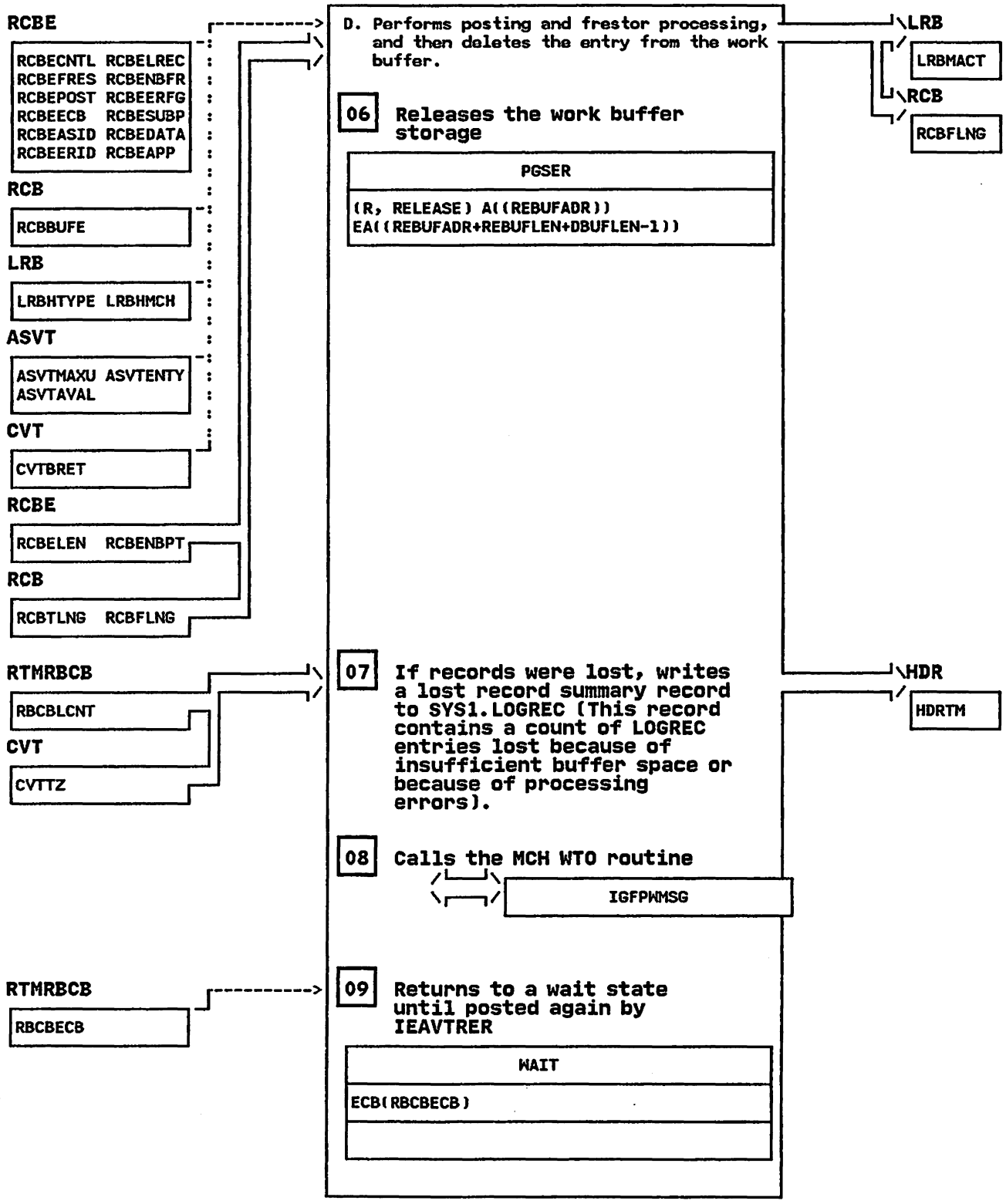
REGISTER CONTENTS ON EXIT: Irrelevant





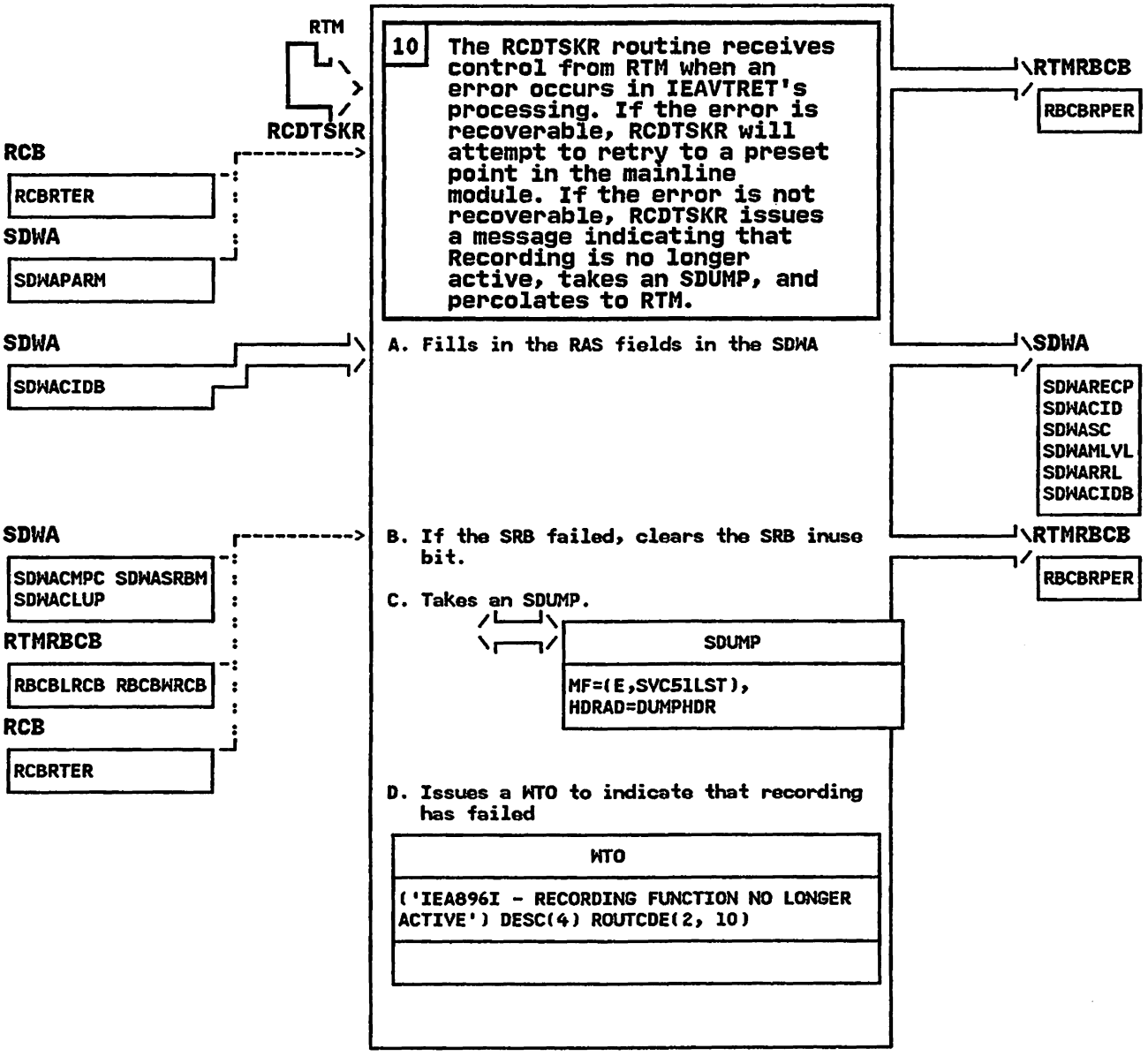


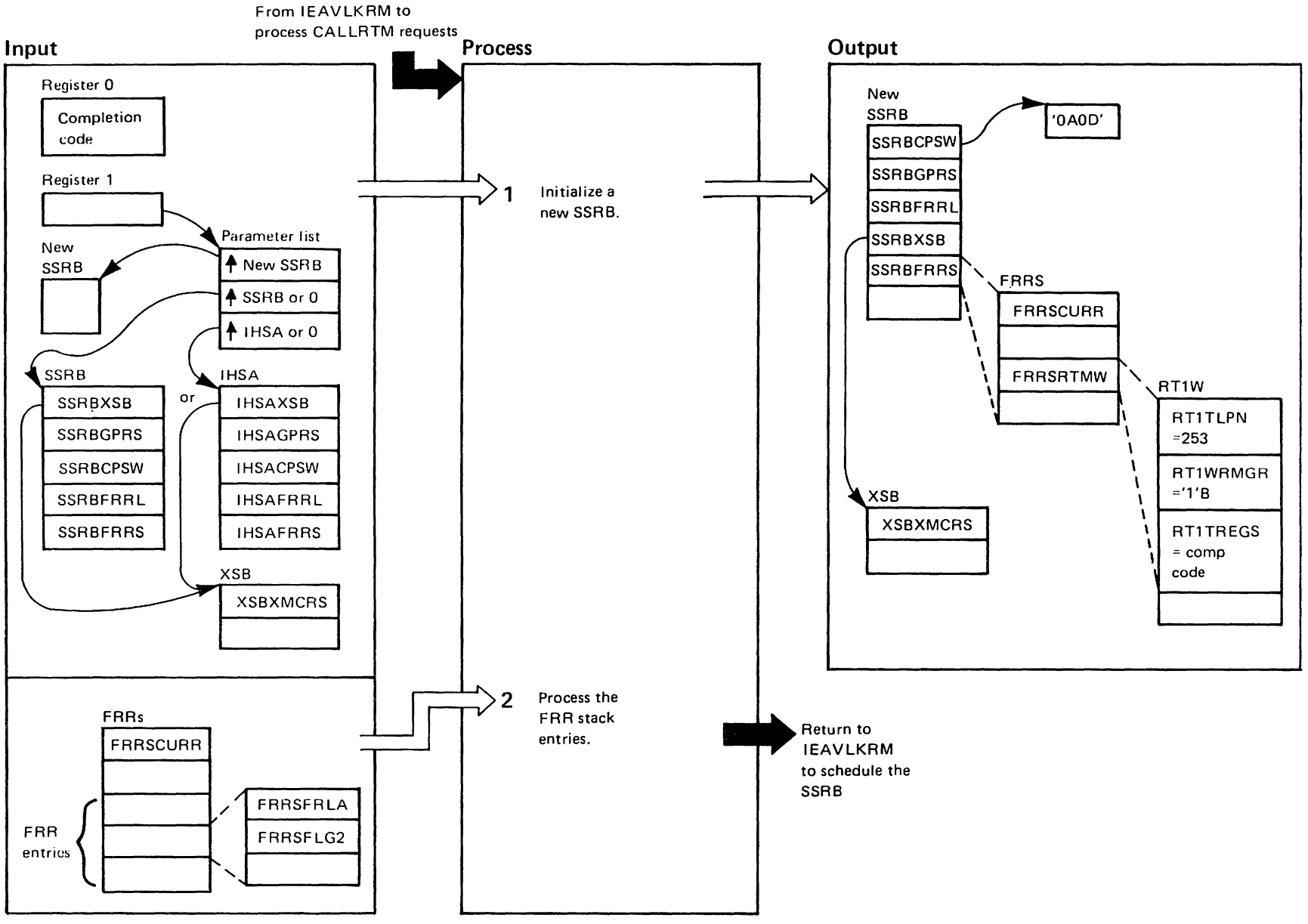




IEAVTRET - Recording Task

STEP 10





IEAVTRMC – CALLRTM TYPE=RMGRCML Processor (Part 2 of 2)

Extended Description | **Module** | **Label**

The lock resource manager (IEAVLKRM) issues a CALLRTM TYPE=RMGRCML at address space termination time for each SRB or task in the terminating address space which holds the local lock of another address space. IEAVTRMC initializes an SSRB with information from an input SSRB (for SRBs) or from an IHSA (for tasks) and sets the resume PSW in the SSRB to point to an SVC 13. When the SSRB is dispatched, it abends, causing RTM1 to be entered. RTM1 can then route control to MODE=LOCAL FRRs for resource cleanup.

1 IEAVTRMC initializes the new SSRB with the PSW, general purpose registers, cross memory registers, and the FRR stack from the input SSRB or IHSA. IEAVTRMC initializes the RTM1 workarea portion of the FRR stack with control information so that RTM1, when entered for the SRB's abend, knows that the processing is on behalf of a CALLRTM TYPE=RMGRCML.

2 IEAVTRMC processes the FRR stack entries for RTM1. IEAVTRMC deletes FRRs from the FRR stack until an FRR with MODE=LOCAL is found or until the stack is empty.

IEAVTRRR - MODULE DESCRIPTION

DESCRIPTIVE NAME: RTM1 FRR Routines

FUNCTION:
IEAVTRRR is a collection of FRRs that protect several functions of RTM1's FRR processing. See the individual entry point descriptions for the specific RTM1 function protected by each FRR.

ENTRY POINT: R1RFRR

PURPOSE:
Protects IEAVTR1R from errors while IEAVTR1R is using the software recording facility, RECORD, to write an SDWA to SYS1.LOGREC.

LINKAGE: BALR

CALLERS: RTM FRR Processing

INPUT:
Obtains error information and RTM control information from the SDWA.

OUTPUT: Sets RAS and retry information in the SDWA.

EXIT NORMAL: Returns to the caller.

EXIT ERROR: There are no exit error conditions.

ENTRY POINT: R1SFRR

PURPOSE:
Protects IEAVTR1S from errors while IEAVTR1S is obtaining an SQA SDWA.

LINKAGE: BALR

CALLERS: RTM FRR Processing

INPUT:
Obtains error information and RTM control information from the SDWA.

OUTPUT: Sets RAS and retry information in the SDWA.

EXIT NORMAL: Returns to the caller.

EXIT ERROR: There are no exit error conditions.

ENTRY POINT: R1IFRR

PURPOSE:
Protects IEAVTR1I from errors while IEAVTR1I is copying dump options, dump ranges and subpool lists to the SDWA.

LINKAGE: BALR

CALLERS: RTM FRR Processing

INPUT:
Obtains error information and RTM control information from the SDWA.

OUTPUT: Sets RAS and retry information in the SDWA.

EXIT NORMAL: Returns to the caller.

EXIT ERROR: There are no exit error conditions.

ENTRY POINT: R1FFRR

PURPOSE:
Protects IEAVTR1F from errors when IEAVTR1F is processing MODE=LOCAL FRRs.

LINKAGE: BALR

CALLERS: RTM FRR Processing

INPUT:
Obtains error information and RTM control information from the SDWA.

OUTPUT: Sets RAS and retry information in the SDWA.

EXIT NORMAL: Returns to the caller.

EXIT ERROR: There are no exit error conditions.

ENTRY POINT: R1XFRR

PURPOSE:
Protects IEAVTR1X from errors while IEAVTR1X is establishing an FRR's cross memory addressing environment via a CMSET instruction.

LINKAGE: BALR

CALLERS: RTM FRR Processing

INPUT:
Obtains error information and RTM control information from the SDWA.

OUTPUT: Sets RAS and retry information in the SDWA.

EXIT NORMAL: Returns to the caller.

EXIT ERROR: There are no exit error conditions.

ENTRY POINT: R1GFRR

PURPOSE:
Protects IEAVTR1G from errors when IEAVTR1G is using GTF to trace the return of an FRR.

LINKAGE: BALR

CALLERS: RTM FRR Processing

INPUT:
Obtains error information and RTM control information from the SDWA.

OUTPUT: Sets RAS and retry information in the SDWA.

EXIT NORMAL: Returns to the caller.

EXIT ERROR: There are no exit error conditions.

EXTERNAL REFERENCES:

ROUTINES: None

DATA AREAS: No data areas are used.

IEAVTRRR - MODULE DESCRIPTION (Continued)

CONTROL BLOCKS:

Common name	Macro ID	Usage	Function
FRRS	IHAFRRS	write	Sets the current FRR address in the FRR stack header.
LCCA	IHALCCA	read	Obtains address of CPU work save area.
PSA	IHAPSA	read	Obtains addresses of various FRR stacks and the LCCA.
RT1W	IHART1W	read and write	Obtains RTM control information.
SDWA	IHASDWA	read and write	Obtains error information and sets retry information.
WSAVT	IHAWSAVT	read	Obtains RTM work save area address.

TABLES: No tables used.

SERIALIZATION:
IEAVTRRR does not obtain any locks. IEAVTRRR runs disabled to serialize the RTM control information in the RTM1 work area.

IEAVTRRR - MODULE OPERATION

IEAVTRRR receives control to protect various modules via FRRs from errors that might occur during their processing.

The general processing of an FRR contained in IEAVTRRR follows:

- Calls the internal procedure, INITSDWA, to initialize the SDWA with standard RAS information, to set up for abort processing and to initialize the retry registers in the SDWA.
- If the error is not recoverable, sets up for percolation. The variable recording area (VRA) of the SDWA is initialized with a message indicating the function that was forced to percolate.
- If the error is recoverable, sets up for retry. The VRA of the SDWA is initialized with a message indicating the function that was recovered.
- Returns to RTM.

RECOVERY OPERATION:

Default recovery processing, contained in module IEAVTRTR, protects IEAVTRRR's processing.

IEAVTRRR - DIAGNOSTIC AIDS

ENTRY POINT NAMES: R1RFRR
R1SFRR
R1IFRR
R1FFRR
R1XFRR
R1GFRR

MESSAGES: None

ABEND CODES: None

WAIT STATE CODES: None

RETURN CODES: None

REGISTER CONTENTS ON ENTRY:

ENTRY POINT R1RFRR:

Register 0 - 200 byte FRR work area address
Register 1 - SDWA address
Registers 2-13 - Irrelevant
Register 14 - Return address
Register 15 - Entry point address

ENTRY POINT R1SFRR:

Register 0 - 200 byte FRR work area address
Register 1 - SDWA address
Registers 2 - 13 - Irrelevant
Register 14 - Return address
Register 15 - Entry point address

ENTRY POINT R1IFRR:

Register 0 - 200 byte FRR work area address
Register 1 - SDWA address
Registers 2 - 13 - Irrelevant
Register 14 - Return address
Register 15 - Entry point address

ENTRY POINT R1FFRR:

Register 0 - 200 byte FRR work area address
Register 1 - SDWA address
Registers 2 - 13 - Irrelevant
Register 14 - Return address
Register 15 - Entry point address

ENTRY POINT R1XFRR:

Register 0 - 200 byte FRR work area address
Register 1 - SDWA address
Registers 2 - 13 - Irrelevant
Register 14 - Return address
Register 15 - Entry point address

ENTRY POINT R1GFRR:

Register 0 - 200 byte FRR work area address
Register 1 - SDWA address
Registers 2 - 13 - Irrelevant

Register 14 - Return address
Register 15 - Entry point address

REGISTER CONTENTS ON EXIT:

ENTRY POINT R1RFRR:

EXIT NORMAL:

Registers 0 - 13 - Not restored
Register 14 - Restored
Register 15 - Not restored

ENTRY POINT R1SFRR:

EXIT NORMAL:

Registers 0 - 13 - Not restored
Register 14 - Restored
Register 15 - Not restored

ENTRY POINT R1IFRR:

EXIT NORMAL:

Registers 0 - 13 - Not restored
Register 14 - Restored
Register 15 - Not restored

ENTRY POINT R1FFRR:

EXIT NORMAL:

Registers 0 - 13 - Not restored
Register 14 - Restored
Register 15 - Not restored

ENTRY POINT R1XFRR:

EXIT NORMAL:

Registers 0 - 13 - Not restored
Register 14 - Restored
Register 15 - Not restored

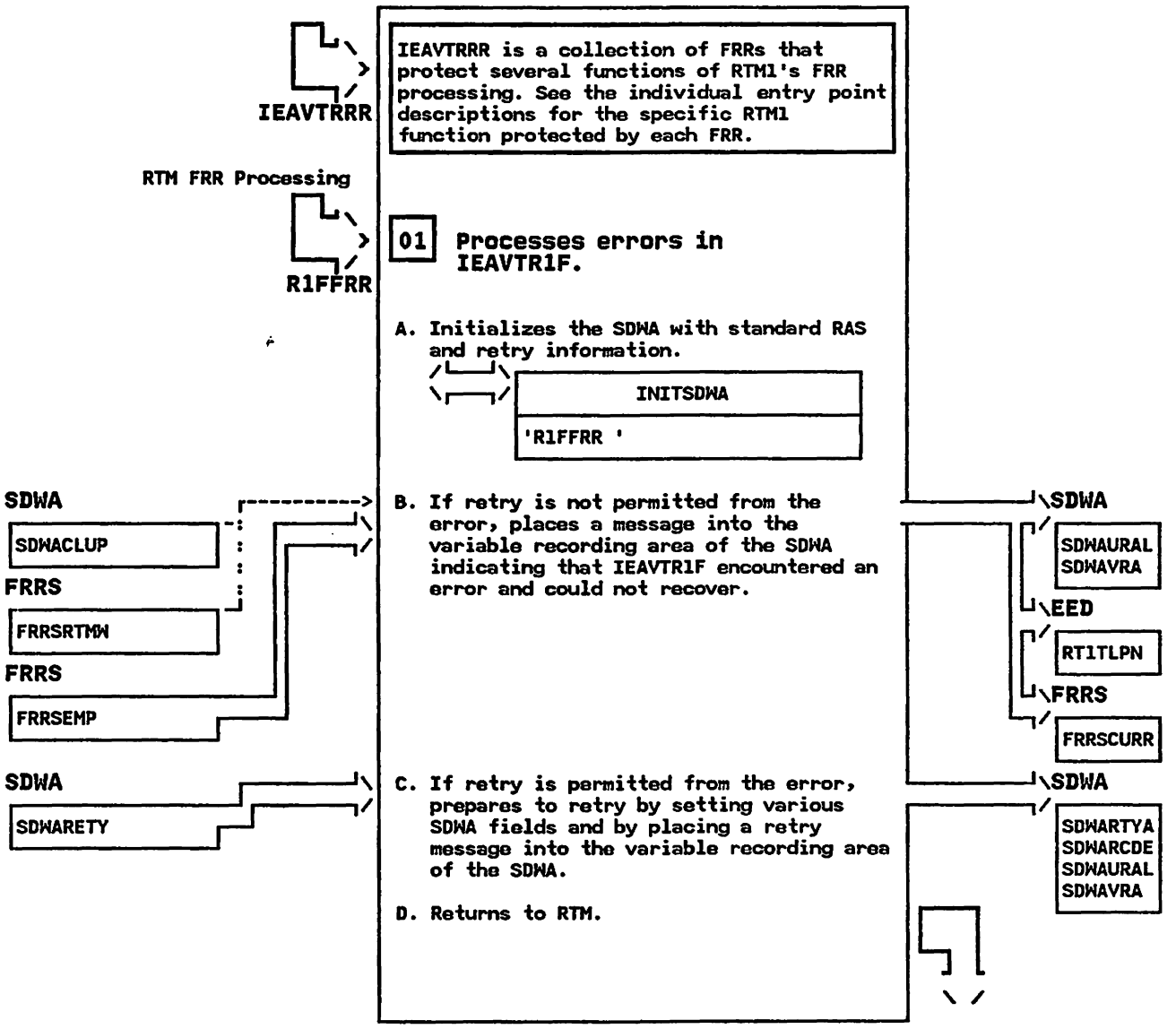
ENTRY POINT R1GFRR:

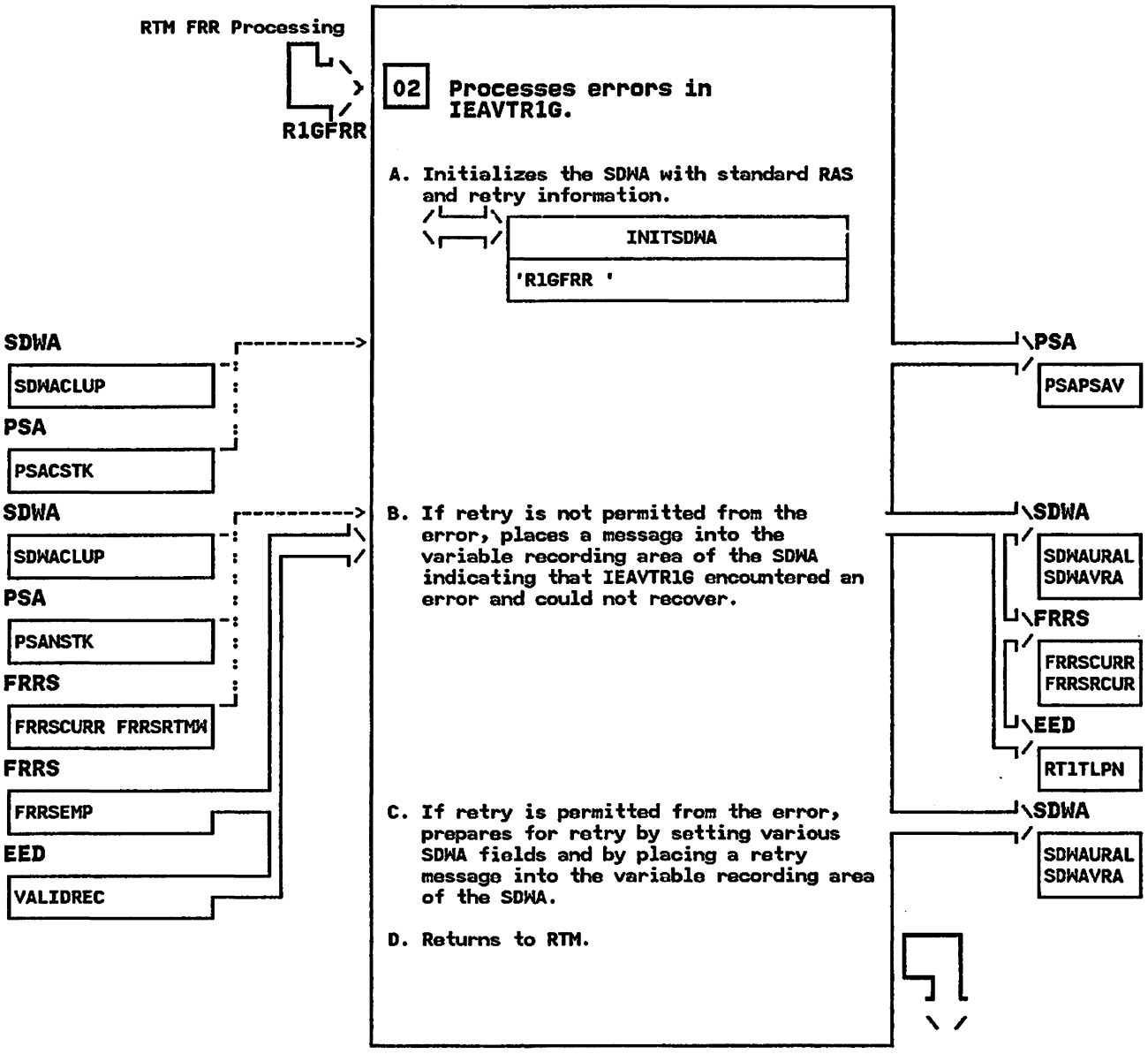
EXIT NORMAL:

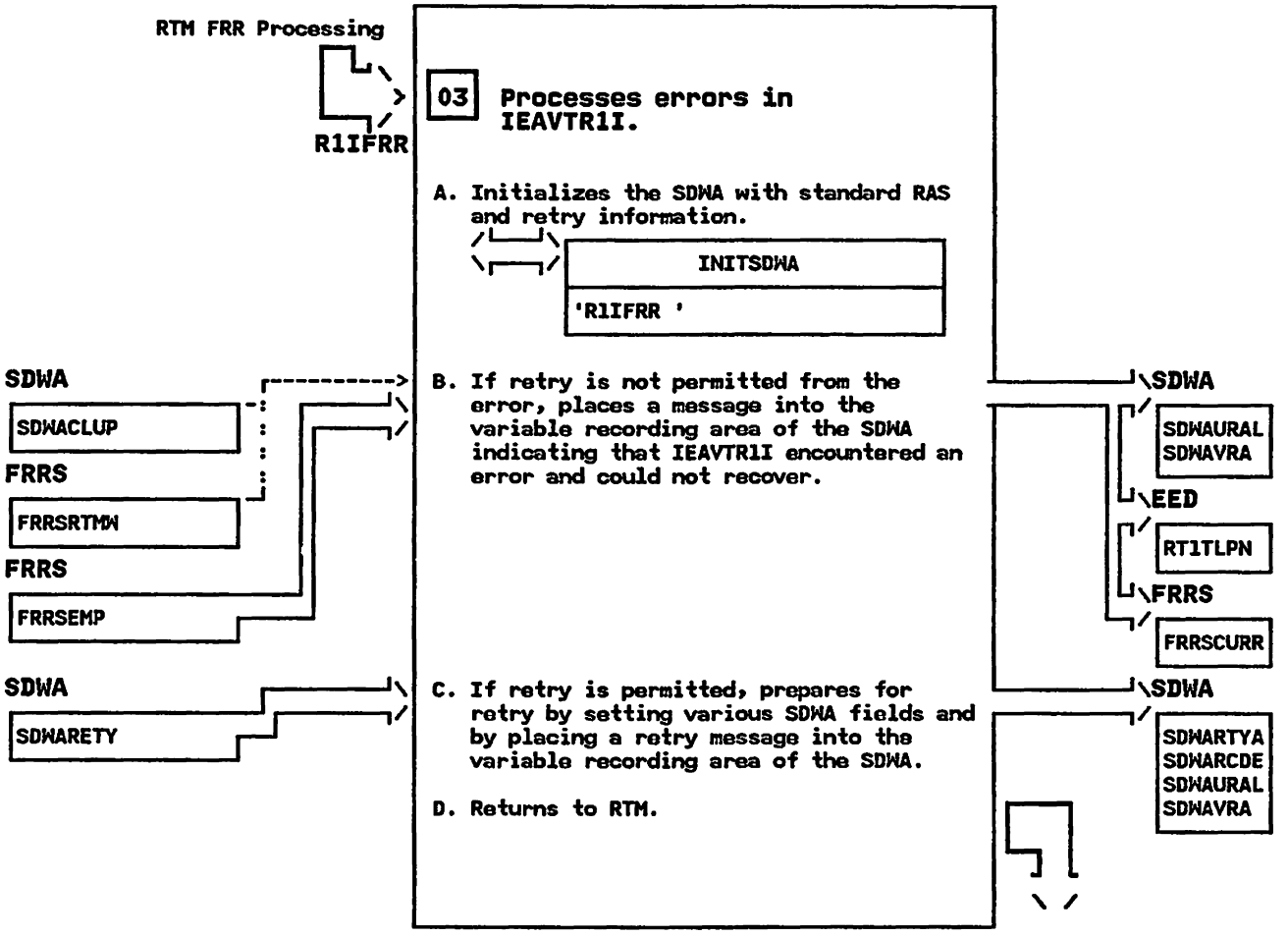
Registers 0 - 13 - Not restored
Register 14 - Restored
Register 15 - Not restored

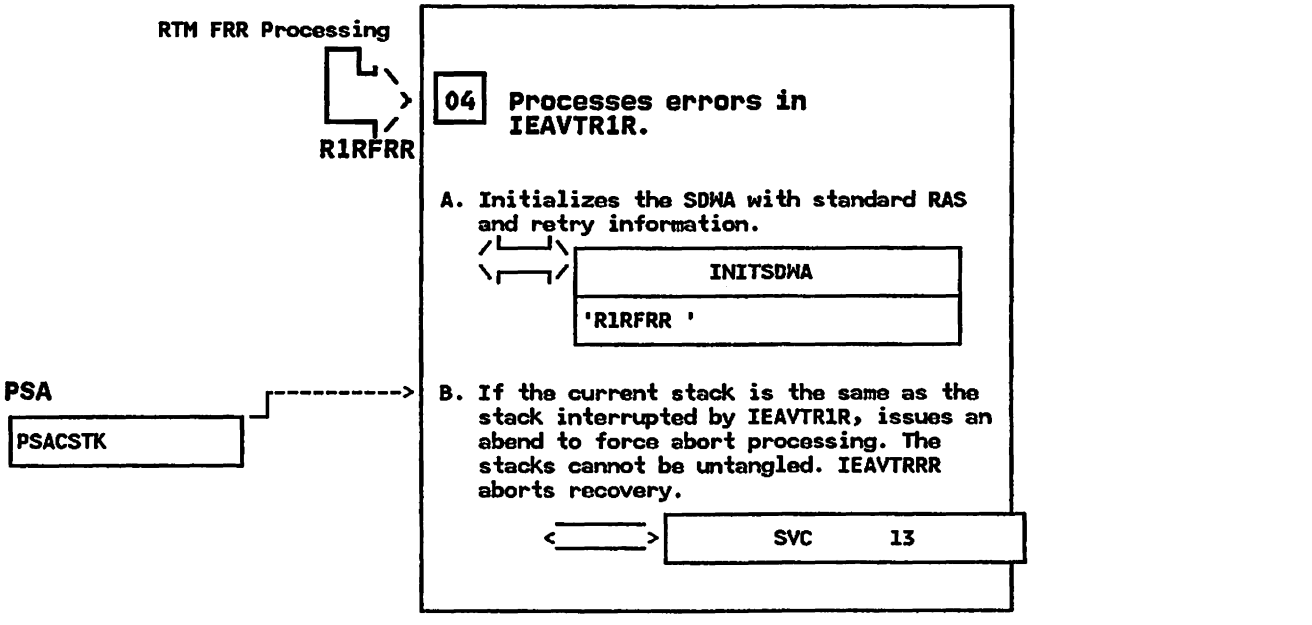
IEAVTRRR - RTM1 FRR Routines

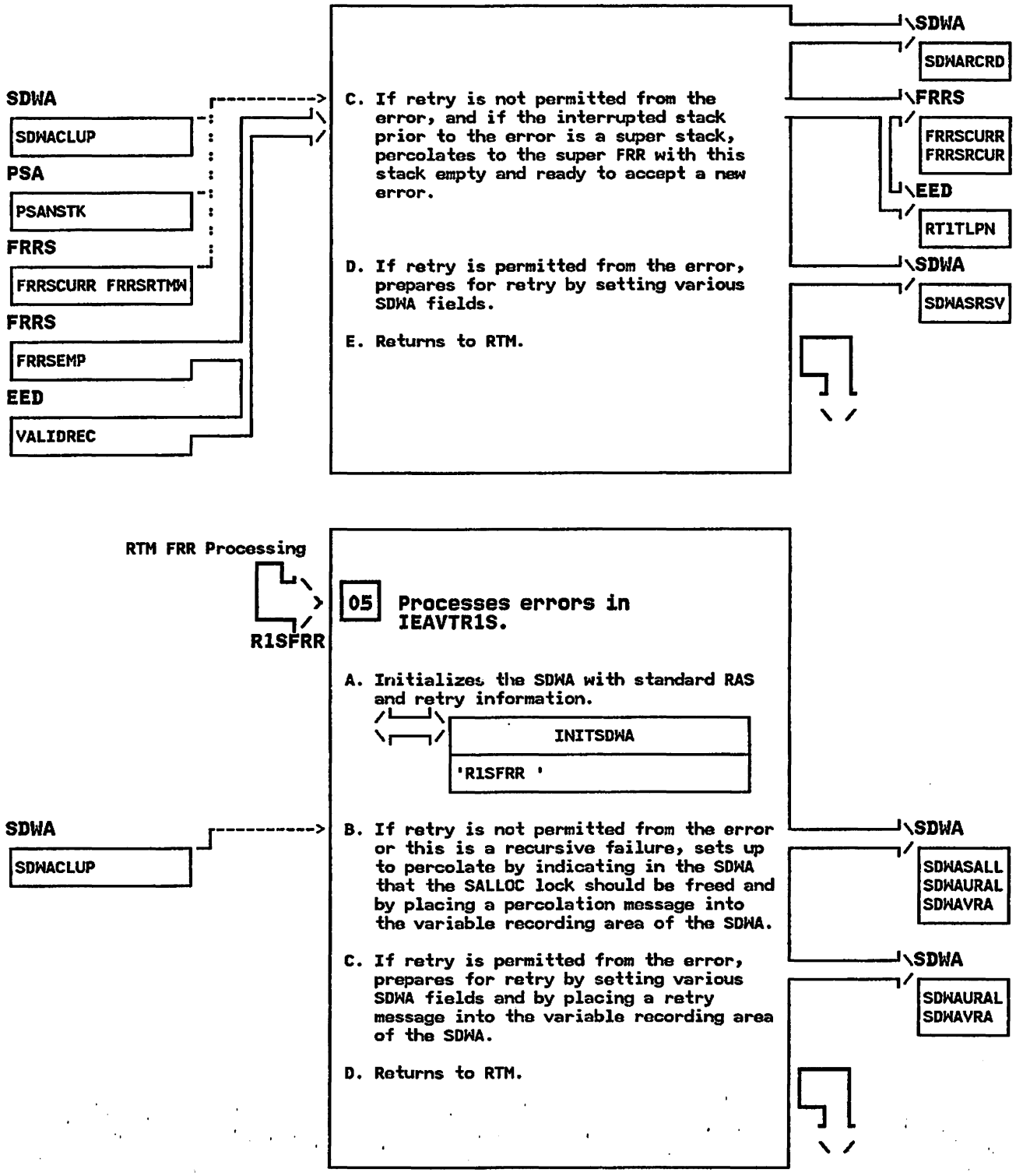
STEP 01





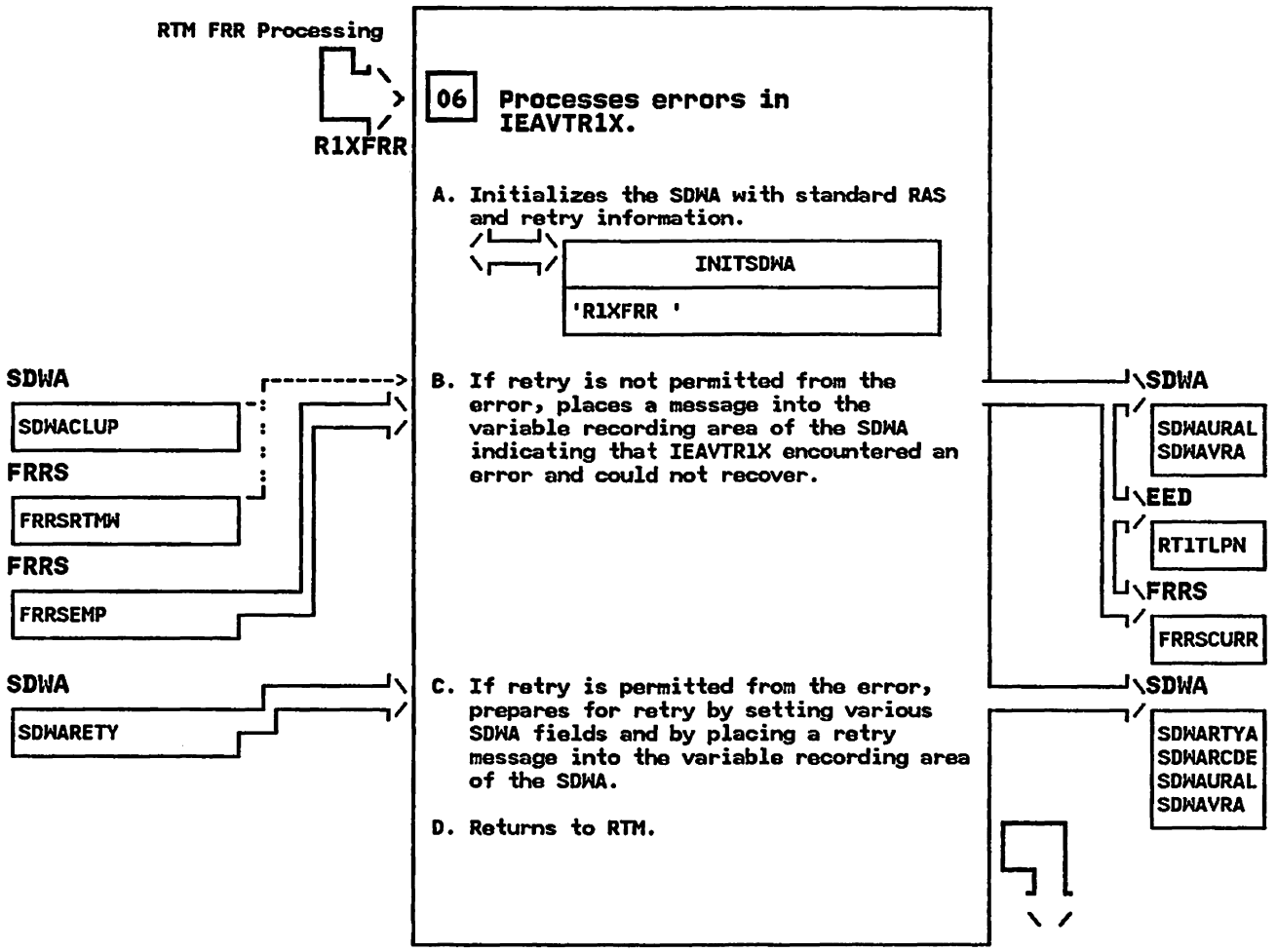






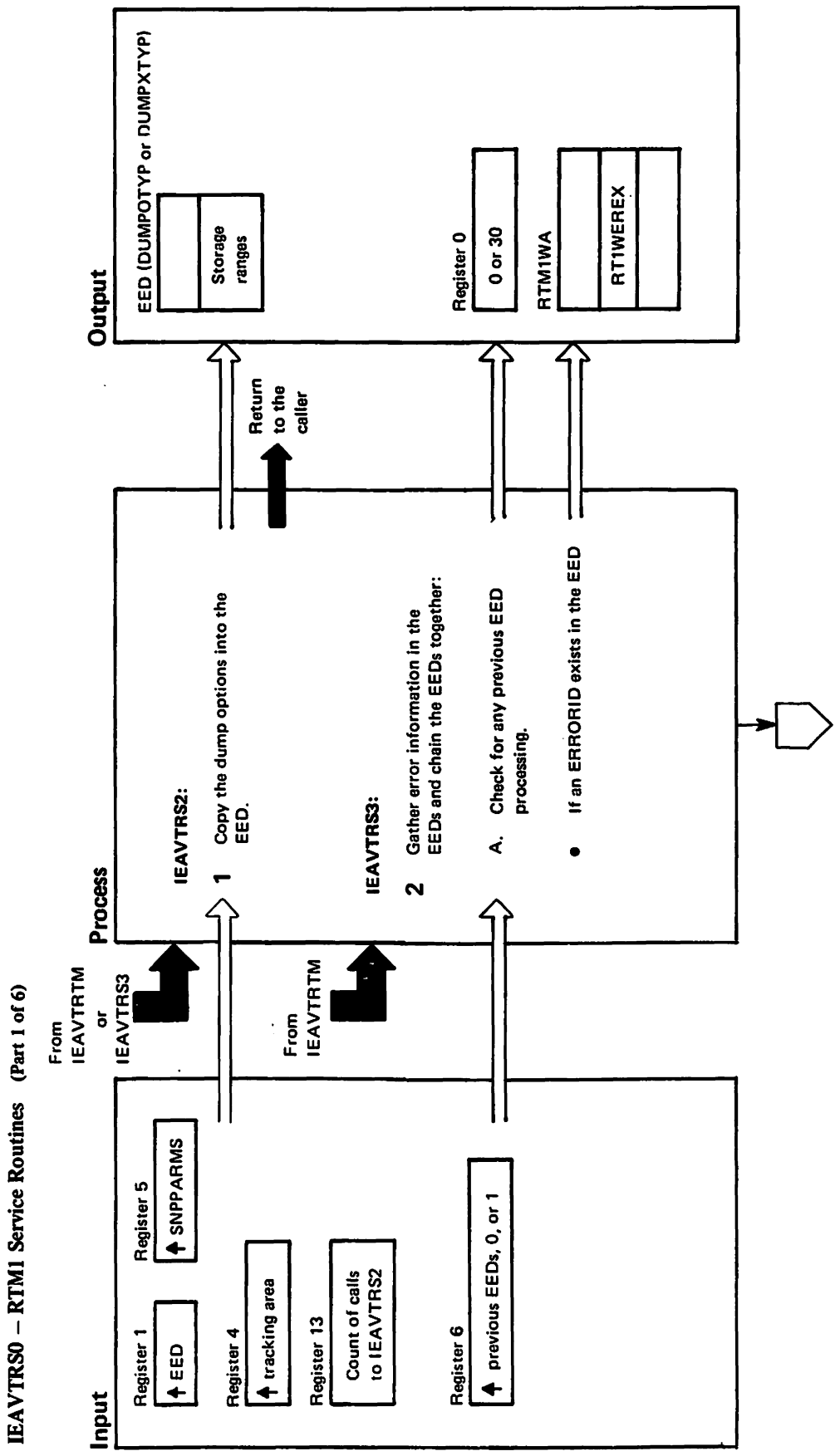
IEAVTRRR - RTM1 FRR Routines

STEP 06



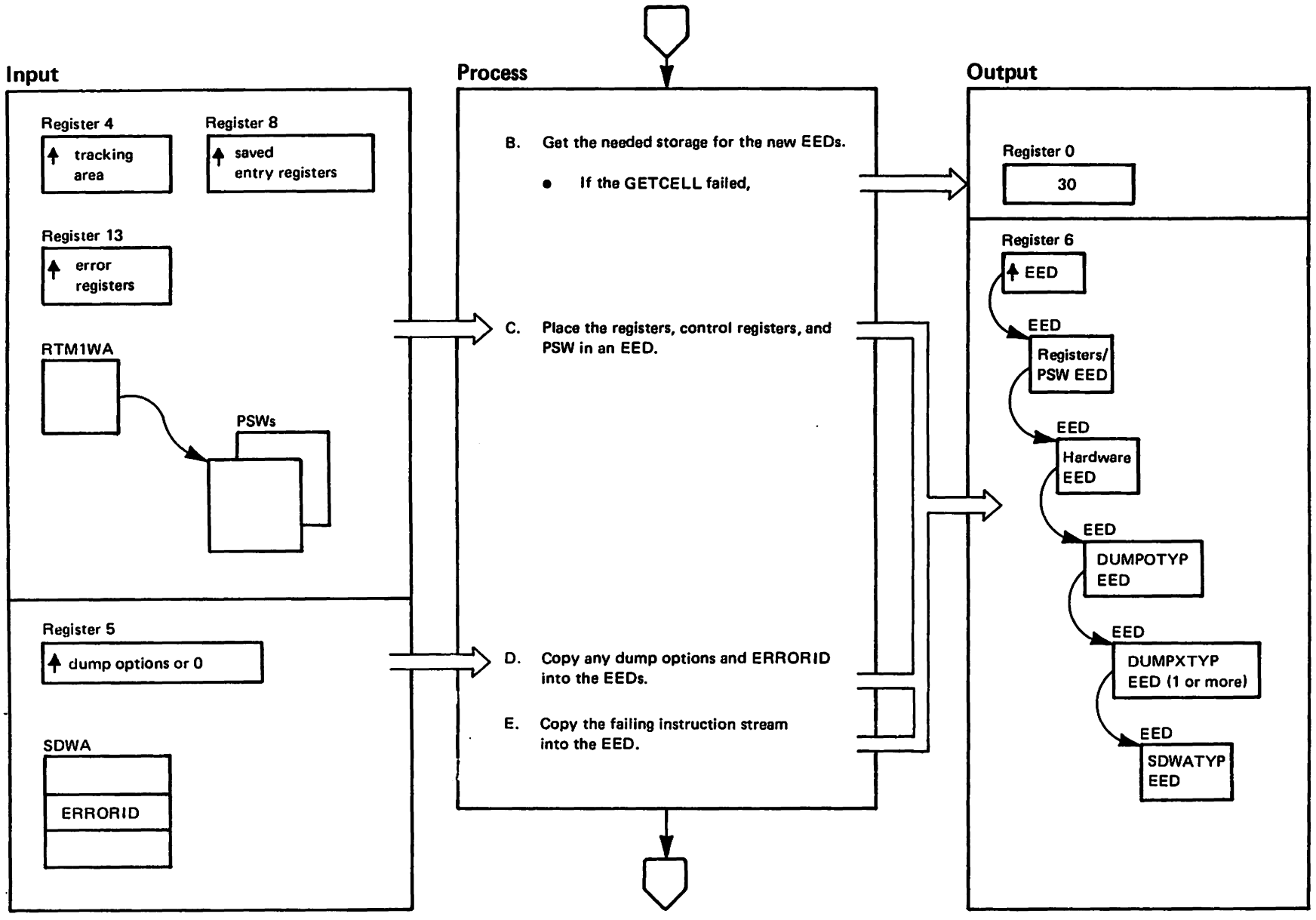
This page left blank intentionally.

IEAVTRS0 - RTM1 SERVICE ROUTINES



Extended Description	Module	Label
IEAVTRS0 consists of several entry points called either internally or by IEAVTRTM. These entry points are called to copy information into EEDs or an SDWA, act as an interface to the STATUS routine to make a task dispatchable, or act as an interface to IGFPTERM to terminate the system.		
1 The DUMPEED subroutine in IEAVTRS3 and the SRBTASKQ segment in IEAVTRTM call IEAVTRS0 at entry point IEAVTRS2 to place dump options passed by the invoker of an ABEND, CALLRTM, or SETRP macro into an extended error descriptor (EED). On entry, register 1 points to an EED, register 4 points to the tracking area (a six-word parameter area of RTM's FRR), register 5 points to the dump options field (SNPPARMS), and register 13 contains a count of the number of times this entry point has been called. The output from IEAVTRS2 is an EED containing the dump options and the storage ranges to be dumped. The EEDTYPE is either a DUMPOTYP or DUMPXTYP.		IEAVTRS2
2 Segments XMABTERM and SCHDRTM2 in IEAVTRTM call IEAVTRS0 at entry point IEAVTRS3 to place error information into EEDs and to chain the EEDs together as a single-threaded, forward-pointing queue with its origin in register 6. IEAVTRS3 determines if there are any EEDs. To do this IEAVTRS3 checks register 6. Register 6 either points to the EED chain, contains a one to indicate that a previous attempt to get an EED cell failed, or contains a zero to indicate no request for EED cells has been previously attempted.		IEAVTRS3 OLDEEDS

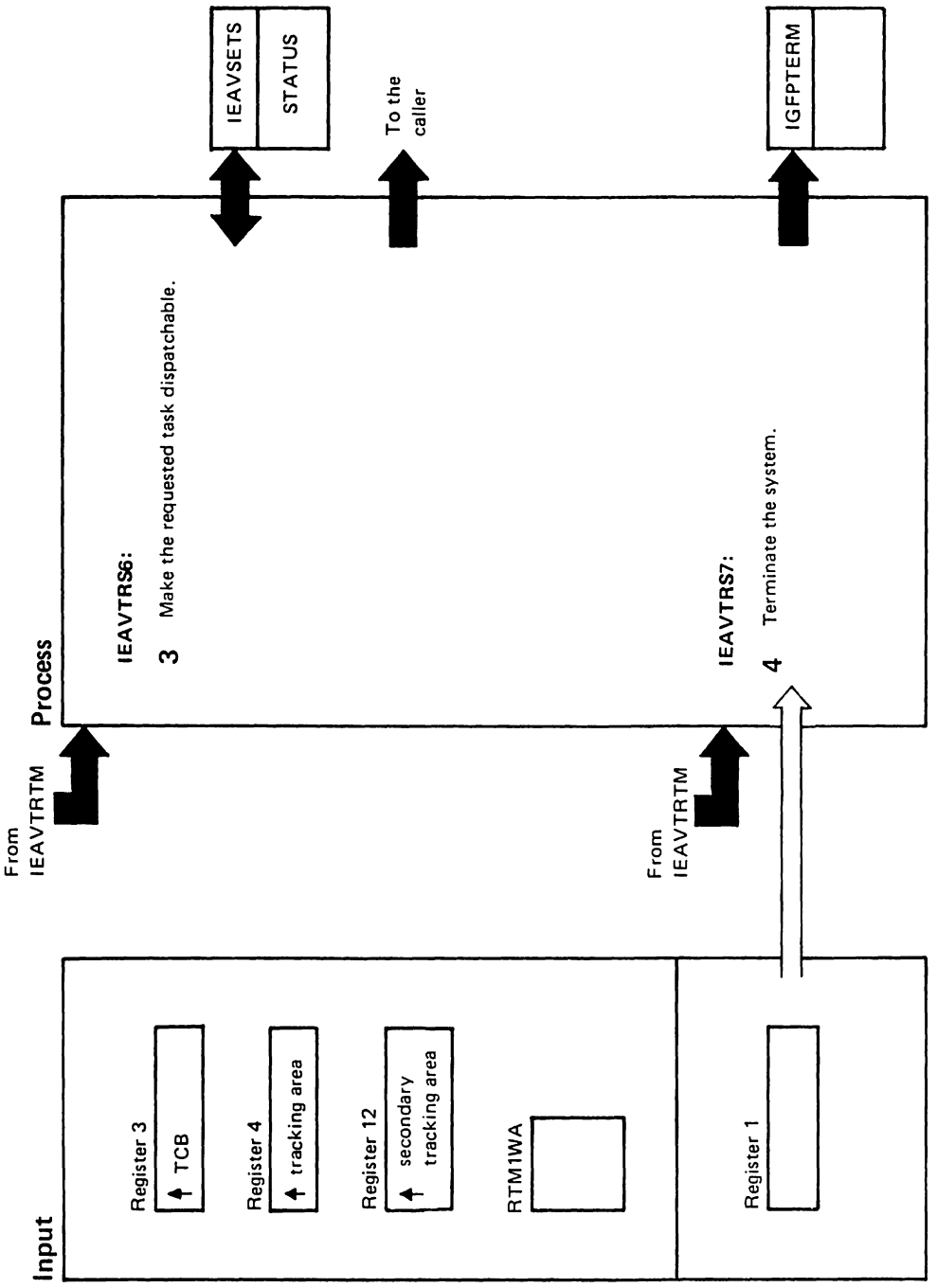
Extended Description	Module	Label
A. There are two situations where EEDs might have been previously requested. First, if a machine check occurred, it placed the hardware repair data, registers, PSW, and control registers three and four into EEDs. Second, if RTM1 set up a locally locked task or an SRB that suffered an STERM error for ABEND processing, it placed the registers, PSW, and control registers three and four into EEDs. If no previous attempts to get an EED cell failed, IEAVTRS3 sets register 0 to a zero to indicate that processing is to continue. If a previous attempt to get an EED cell failed, IEAVTRS3 sets register 0 to a 30 so that RTM will skip further processing. IEAVTRS3 checks the EED chain to see if an error ID (ERRORID) already exists. If one does, IEAVTRS3 sets the RT1WEREX flag in the RTM1WA to one.		



IEAVTRS0 – RTM1 Service Routines (Part 4 of 6)

Extended Description	Module	Label
B. For new EEDs, IEAVTRS3 issues a GETCELL to obtain storage for any required EEDs. If a cell cannot be obtained, then IEAVTRS3 places a 30 in register 0 so that RTM will bypass further processing.		NEWEEDS
C. IEAVTRS3 stores the registers and control registers three and four, in the first EED on the chain. When RTM1 is operating as a SLIH (second level interrupt handler), the RTM1WA field contains a pointer to the PSW at the time of the error. IEAVTRS3 also places this PSW in the EED.		
D. IEAVTRS3 copies the dump options and storage ranges pointed to by register 5 into an EED. The invoker of the ABEND, CALLRTM, or SETRP macro passed these dump options and storage ranges to IEAVTRS3. If an ERRORID exists and has not been placed in an EED, IEAVTRS3 places it in an EED. The ERRORID has three parts: a sequence number (SDWASEQ #), a logical processor ID (SDWACPU1) and a time stamp (SDWAERTM). The EEDTYPE field determines if the EEDs are DUMPOTYP or DUMPXTYP.		DUMPEED
E. IEAVTR1A copies the six bytes of instruction stream that precede and the six bytes that follow the instruction counter (IC) of the failing PSW into the EEDFAIN field of the EED. (See the M.O. diagram IEAVTR1A – RTM1 Failing Instruction Processor.)	IEAVTR1A	IEAVTR1B

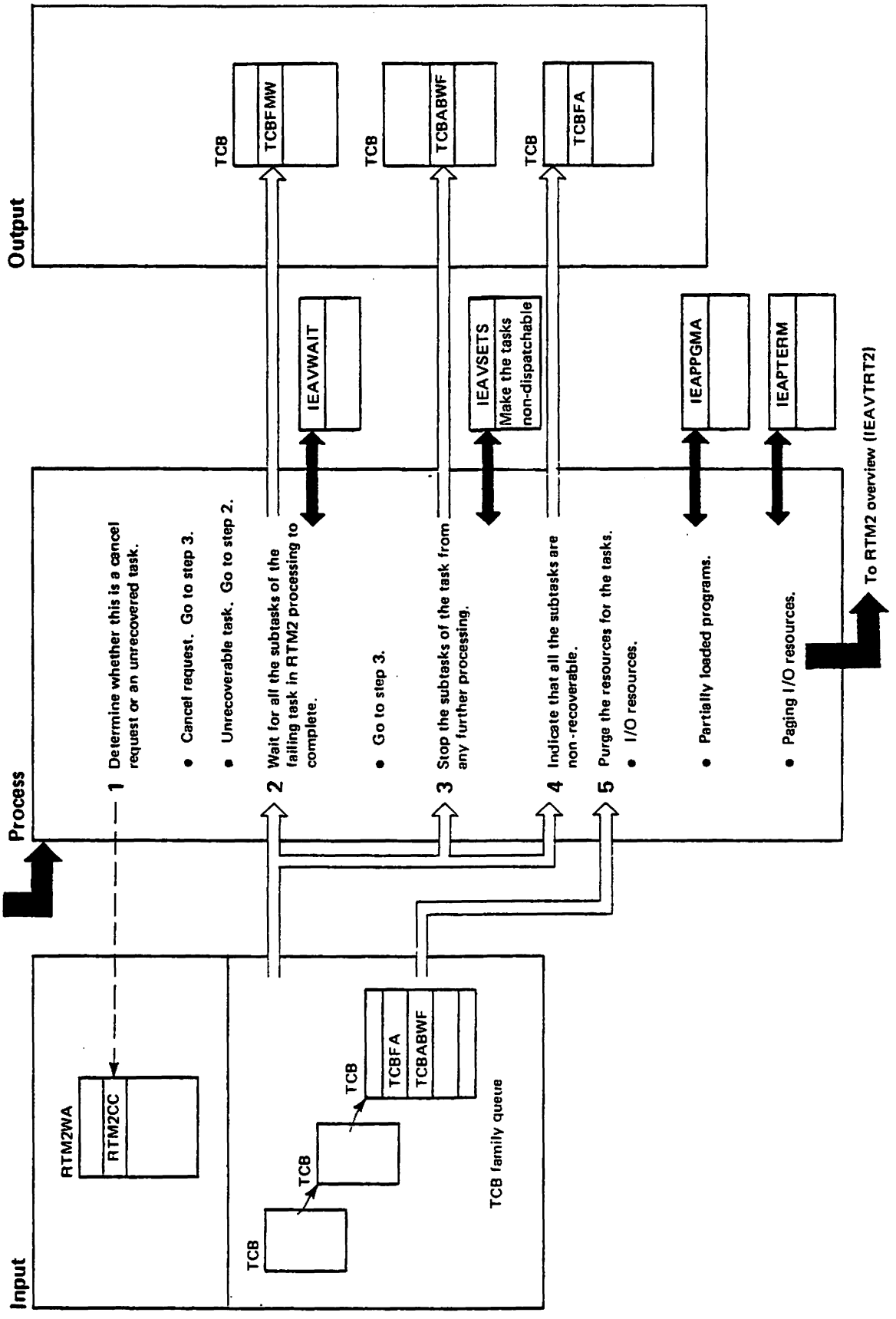
IEAVTRS0 – RTM1 Service Routines (Part 5 of 6)



IEAVTRS0 – RTM1 Service Routines (Part 6 of 6)

Extended Description	Module	Label
3 Two places in the SCHDRTM2 segment of IEAVTRTM call IEAVTRS0 at entry point IEAVTRS6. IEAVTRTM uses IEAVTRS6 to call STATUS when RTM wants to make a task dispatchable. The input to IEAVTRS6 is: Register 3 – A pointer to the TCB to be processed Register 4 – A pointer to an area containing the saved registers Register 12 – A pointer to the secondary tracking area used to save the registers		IEAVTRS6
Entry point IEAVTRS6 passes this input to STATUS, which makes the task dispatchable.	IEAVSETS	
4 The MEMTERM segment of IEAVTRTM calls IEAVTRS0 at entry point IEAVTRS7 if a DAT error has occured in an address space that may not be terminated (ASCBNOMT=1) and the ASCBNOMD flag was set to one. IEAVTRS7 calls IGFPTERM to terminate the system. The input IEAVTRS7 passes to IGFPTERM is: Register 1 – A pointer to a two-word area. The first word is a pointer to the WTO message "IEA802W – DAT ERROR IN SYSTEM ADDRESS SPACE". The second word points to a LOGREC buffer (LRB). IEAVTRS7 calls IGFPTERM to issue message IEA802W and puts the system in a X'A00' wait state.	IGFPTERM	IEAVTRS7

IEAVTRTC - SYNCHRONIZE FAILING TASKS



IEAVTRTC – Synchronize Failing Tasks (Part 1 of 2)

IEAVTRTC – Synchronize Failing Tasks (Part 2 of 2)

Extended Description	Module	Label
RTM2 synchronizes the termination of tasks in a TCB family queue to allow all the tasks to receive termination processing. RTM2 allows these subtasks to terminate and to have storage displays. This aids in debugging.		
RTM2 waits for all the tasks in RTM2 to complete processing before terminating them (except for CANCEL requests). RTM2 stops all the tasks in the failing task's TCB family queue from any processing, including asynchronous exit processing. This prevents any additional termination requests for this TCB family queue. Then, RTM2 gives control to special purging routines (*not* the resource managers described in M.O. diagram IEAVTMMT – Address Space Purge Processing to clean up task resources.		
1 RTM2 synchronizes failing tasks for one of two reasons: there has been a CANCEL request from the system or operator; or the task cannot be recovered (M.O. diagram IEAVTAS1 – Recover Task Processing shows recovery processing). RTM2 checks the completion code of the task, in RTM2CC, for a X'n22' value, with the n being any alphanumeric value, and with the last 2 characters being "22." This completion code indicates a CANCEL. For CANCEL requests, RTM2 performs steps 3, 4, and 5, in that order. For unrecovered tasks, RTM2 performs steps 2, 3, 4 and 5, in that order.	IEAVTRTC	RTCTLRCR
A cancel request must come through RTM1 using the CALLRTM macro.		
2 RTM2 allows subtasks undergoing RTM2 processing indicated by the TCBRTM2 field to complete. Note that for unrecoverable tasks, control will go to step 3, and the tasks will be set non-dispatchable.		RTCSTACK
3 RTM2 stops any further processing of the subtasks by giving control to the STATUS routine, with the request to make the subtasks non-dispatchable. The subtasks will be made dispatchable to finish RTM2 processing. Note that except for cancel requests, the subtasks will be allowed to finish RTM2 processing first.	IEAVSETS	RTCCSUB
4 RTM2 sets the TCBFA field in each TCB of the TCB family queue to indicate that these tasks cannot be recovered.	IEAVTRTC	
5 RTM2 now performs initial purging of some of the tasks' resources to prevent any contention for system resources. For example, a task set non-dispatchable while performing a FETCH request would not complete loading the requested program. No new FETCH requests would be honored. Also, no other tasks could use that requested program. Therefore, the RTM2 calls the partially loaded program purge routine to purge such resources. The same example would hold for I/O operations and paging I/O operations. For non-CANCEL requests, control goes to M.O. diagram IEAVTRT2 – RTM2 Overview.		RTCINPRG

IEAVTRTD - MODULE DESCRIPTION

DESCRIPTIVE NAME: RTM1 ASID Service Routine

FUNCTION:
This module verifies that an ASID (received as input to RTM1) is a valid address space to terminate. If the address space is valid, IEAVTRTD returns to IEAVTRTM to process the termination. If the address space is not valid, IEAVTRTD sets a return code indicating that an attempt to terminate an address space that cannot be terminated was made. In addition, if the CALLRTM TYPE=MEMTERM macro was issued, IEAVTRTD records a message in SYS1.LOGREC.

ENTRY POINT: IEAVTRS1

PURPOSE:
Verifies that an ASID, which was received as input to RTM1, represents a valid address space.

LINKAGE: Via BALR 14, 15

CALLERS: IEAVTRTM

INPUT:
Register 2 - Contains the ASID to be validated or the value zero representing the current address space.

OUTPUT:
Register 0 - Set to a return code for IEAVTRTM to determine if the validity check was a success or failure
Register 2 - Set to the ASID of the current address space if the value was zero on entry
Register 8 - Set to the address of the ASCB for the given ASID

Depending on the reason for failing the validity check, one of three possible error descriptions is recorded in SYS1.LOGREC along with a portion of the ASCB.

1.) MEMTERM for ASID XXXXXXXX rejected.
Damaged ASCB acronym missing at YYYYYYYY.
2.) (*)MEMTERM rejected.
ASID XXXXXXXX not assigned.
3.) (*)MEMTERM rejected.
ASID XXXXXXXX exceeds ASVTMAXU.

If a scan of the ASVT finds an assigned slot that points to an ASCB (where the ASCBASID matches XXXXXXXX), then a third part is added to the error descriptions for #2 and #3.
ASVT slot ZZZZ points to ASCB at YYYYYYYY
with matching ASID.

EXIT NORMAL: Returns to the caller, IEAVTRTM

EXIT ERROR:
Retries or reschedules the function from IEAVTRTR to a return point in IEAVTRTM through FRR recovery.

ENTRY POINT: IEAVTRS5

PURPOSE:
Records in SYS1.LOGREC that an attempt to terminate an address space, which may not be terminated (ASCBNOMT = '1'), was made.

LINKAGE: Via BALR 14, 15

CALLERS: IEAVTRTM

IEAVTRTD - MODULE DESCRIPTION (Continued)

INPUT:
- Register 1 - Contains the completion code associated with this MEMTERM request
- Register 2 - Contains the ASID of the address space that was requested to be terminated
- Register 4 - Points to the RESCHFRR 6 word parameter area (where the original registers 0-4 have been saved)
- Register 8 - ASCB address of the address space that was requested to be terminated
- Register 13 - Pointer to the MEMTERM requestor's register save area
- Register 14 - Return address of the caller

OUTPUT:
By invoking the software recording facility, IEAVTRTD records that a request to terminate an address space that may not be terminated was attempted.

EXIT NORMAL: Returns to the caller, IEAVTRTM

EXIT ERROR:
Reschedules the function from IEAVTRTR to a return point in IEAVTRTM through FRR recovery.

EXTERNAL REFERENCES:

ROUTINES: Branch enter recording facility

DATA AREAS: No data areas used.

CONTROL BLOCKS:

Common name	Macro ID	Usage	Function
ASCB	IHAASCB	read	Checks for a valid ASCB.
ASVT	IHAASVT	read	Obtains the entry of the ASID being validity checked to determine if the ASID is assigned.
CVT	CVT	read	Uses the referenced CVTPTR.
LCCA	IHALCCA	read	Obtains the state of the processor (spinning or not).
PSA	IHAPSA	read	Obtains current address space, control block address, FRR stacks, etc.
SDWA	IHASDWA	read and write	Sets up and writes to SYS1.LOGREC to describe erroneous requests.

TABLES: No tables used.

SERIALIZATION:

IEAVTRTD does not obtain any locks. However, the dispatcher lock is held on entry.

IEAVTRTD - MODULE OPERATION

IEAVTRTD receives control to verify that an ASID (received as input to RTM1) is a valid address space to terminate. If the address space is valid, IEAVTRTD returns to IEAVTRTM to process the termination. If the address space is not valid, IEAVTRTD sets a return code indicating that an attempt to terminate an address space that cannot be terminated was made. In addition, if the CALLRTM TYPE=MEMTERM macro was issued, IEAVTRTD records a message in SYS1.LOGREC.

Entry point IEAVTRS1 receives control from IEAVTRTM whenever the CALLRTM macro is issued for a cross memory ABTERM request or for a MEMTERM request.
IEAVTRS1 performs the following processing:

- Checks the validity of the ASID, received from IEAVTRTM, for the following conditions:
  - Is the ASID in the range of valid ASIDs?
  - Is the ASID currently assigned?
  - Does the ASVT entry point to a valid ASCB?

  If the responses to all the preceding questions are yes, IEAVTRTD has a valid address space to be terminated. IEAVTRTD returns control to IEAVTRTM to terminate the address space. If the ASID failed the validity check, the address space cannot be terminated. IEAVTRTD sets a return code (decimal 30) in register 0.

Entry point IEAVTRS5 receives control from IEAVTRTM whenever the CALLRTM TYPE=MEMTERM macro is issued and an attempt to to terminate an address space that cannot be terminated was made. (Field ASCBNOMT equals one.)
IEAVTRS5 performs the following processing:

- Records in SYS1.LOGREC a message indicating that an attempt was made to terminate an address space that cannot be terminated. The message contains the ASID, which was received as input, the ASCB address associated with the input ASID, and register 14, which has the return address of the requestor of the CALLRTM TYPE=MEMTERM macro.

RECOVERY OPERATION:

The logical phase recovery in IEAVTRTR protects IEAVTRTD. If the ASID fails to represent a valid address space, IEAVTRTD attempts a retry for a MEMTERM request. If the ASID represents a valid address space for a cross memory ABTERM or MEMTERM request, IEAVTRTD attempts to reschedule the function.

IEAVTRTD - DIAGNOSTIC AIDS

ENTRY POINT NAMES: IEAVTRS1
IEAVTRS5

MESSAGES: None

ABEND CODES: None

WAIT STATE CODES: None

RETURN CODES:

ENTRY POINT IEAVTRS1:

Register 0 - Contains decimal 30, if the ASID failed the validity check

ENTRY POINT IEAVTRS5: None

REGISTER CONTENTS ON ENTRY:

ENTRY POINT IEAVTRS1:

Register 0 - Function code for RTM1, equal to RT1WMEMT during MEMTERM processing
Register 1 - Completion codes and flags
Register 2 - Set to the ASID to be validity checked or the value 0 representing the current address space
Register 3 - Zero, as last used in IEAVTRTS
Register 4 - Pointer to a tracking area for check pointing volatile registers
Register 5 - Zero, as last used in IEAVTRTS
Register 6 - Contains one of the following:
. a pointer to a chain of EEDs (if any were previously acquired)
. zero (if there were no previous attempts to acquire EEDs)
. the EEDnull value (1) (if an attempt to acquire an EED cell failed
Register 7 - Zero, as last used in IEAVTRTS
Register 8 - Irrelevant
Register 9 - RTM's base register
Register 10 - Irrelevant
Register 11 - RTM's second base register
Register 12 - Contains a pointer to a secondary tracking area used to checkpoint recovery information
Register 13 - Contains a pointer to the registers at the time of the error
(If not previously placed into the EEDs)
Register 14 - Return address
Register 15 - Irrelevant

ENTRY POINT IEAVTRS5:

Register 0 - Function code for RTM1, equal to RT1WMEMT during MEMTERM
Register 1 - Completion codes and flags
Register 2 - Set to the ASID to be validity checked or the value 0 representing the current address space
Register 3 - Zero, as last used in IEAVTRTS
Register 4 - Pointer to a tracking area for check pointing volatile registers

IEAVTRTD - DIAGNOSTIC AIDS (Continued)

Register 5 - Zero, as last used in IEAVTRTS
Register 6 - Contains one of the following:
. a pointer to a chain of EEDs (if any were previously acquired)
. zero (if there were no previous attempts to acquire EEDs)
. the EEDnull value (1) (if an attempt to acquire an EED cell failed
Register 7 - Zero, as last used in IEAVTRTS
Register 8 - ASCB address
Register 9 - RTM's base register
Register 10 - Irrelevant
Register 11 - RTM's second base register
Register 12 - Contains a pointer to a secondary tracking area used to checkpoint recovery information
Register 13 - Contains a pointer to the registers at the time of the error (If not previously placed into the EEDs)
Register 14 - Return address
Register 15 - Irrelevant

REGISTER CONTENTS ON EXIT:

ENTRY POINT IEAVTRS1:

Register 0 - Set to a return code for IEAVTRTM to determine if the validity check was a success or failure
Registers 1-7 - Restored to the values on entry
Register 8 - Set to the address of the ASCB for the given ASID
Register 9 - RTM's base register
Register 10 - Irrelevant
Register 11 - RTM's second base register
Registers 12-13 - Restored to the values on entry
Register 14 - Return address
Register 15 - Irrelevant

ENTRY POINT IEAVTRS5:

Registers 0-4 - Restored to the values on entry
Registers 5-6 - Irrelevant
Register 7 - Zero, as last used in IEAVTRTS
Register 8 - Address of the ASCB for the given ASID
Register 9 - RTM's base register
Register 10 - Irrelevant
Register 11 - RTM's second base register
Register 12-13 - Restored to the values on entry
Register 14 - Return address
Register 15 - Irrelevant

IEAVTRTE - RECURSION PROCESSOR 2

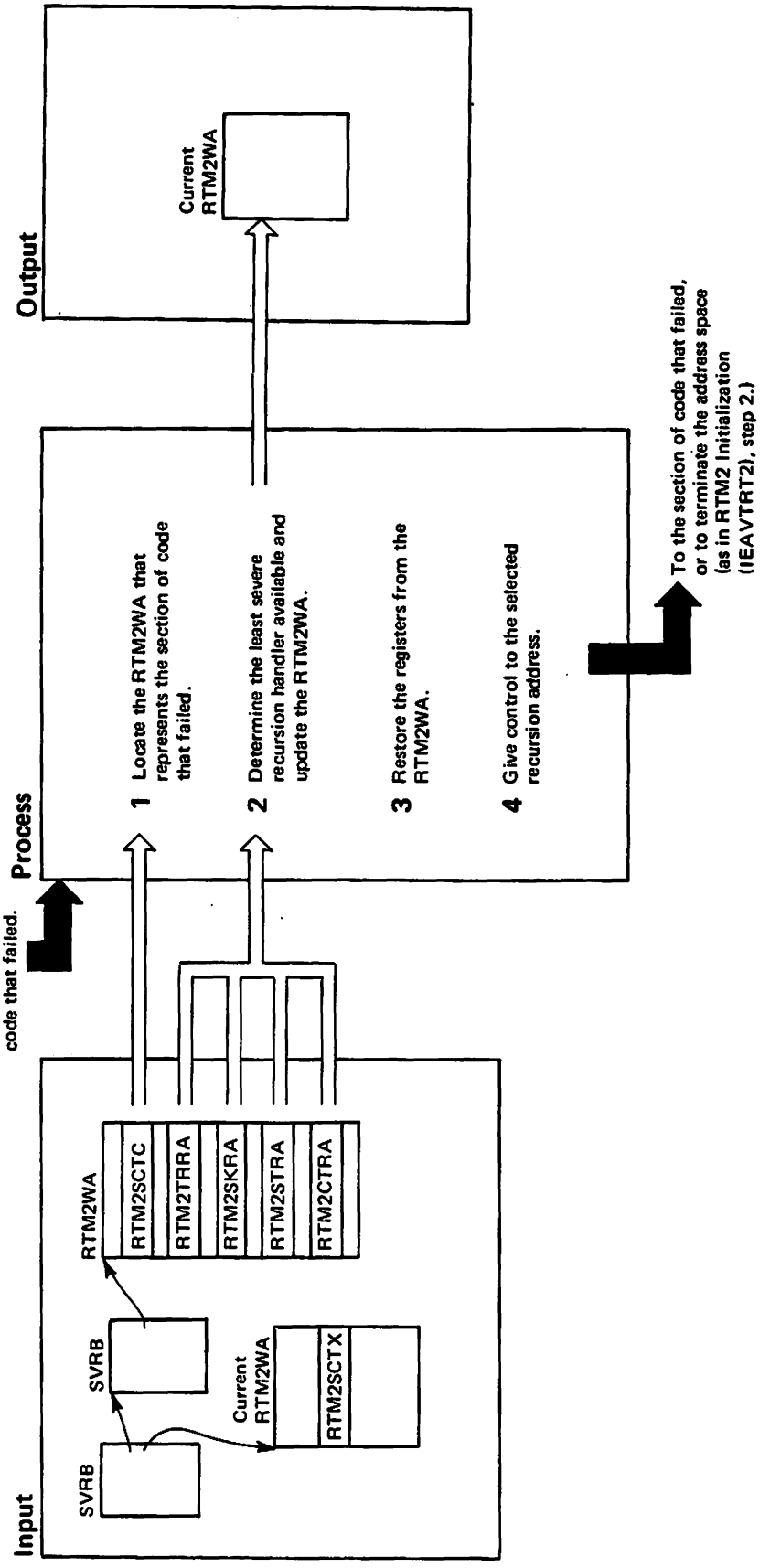
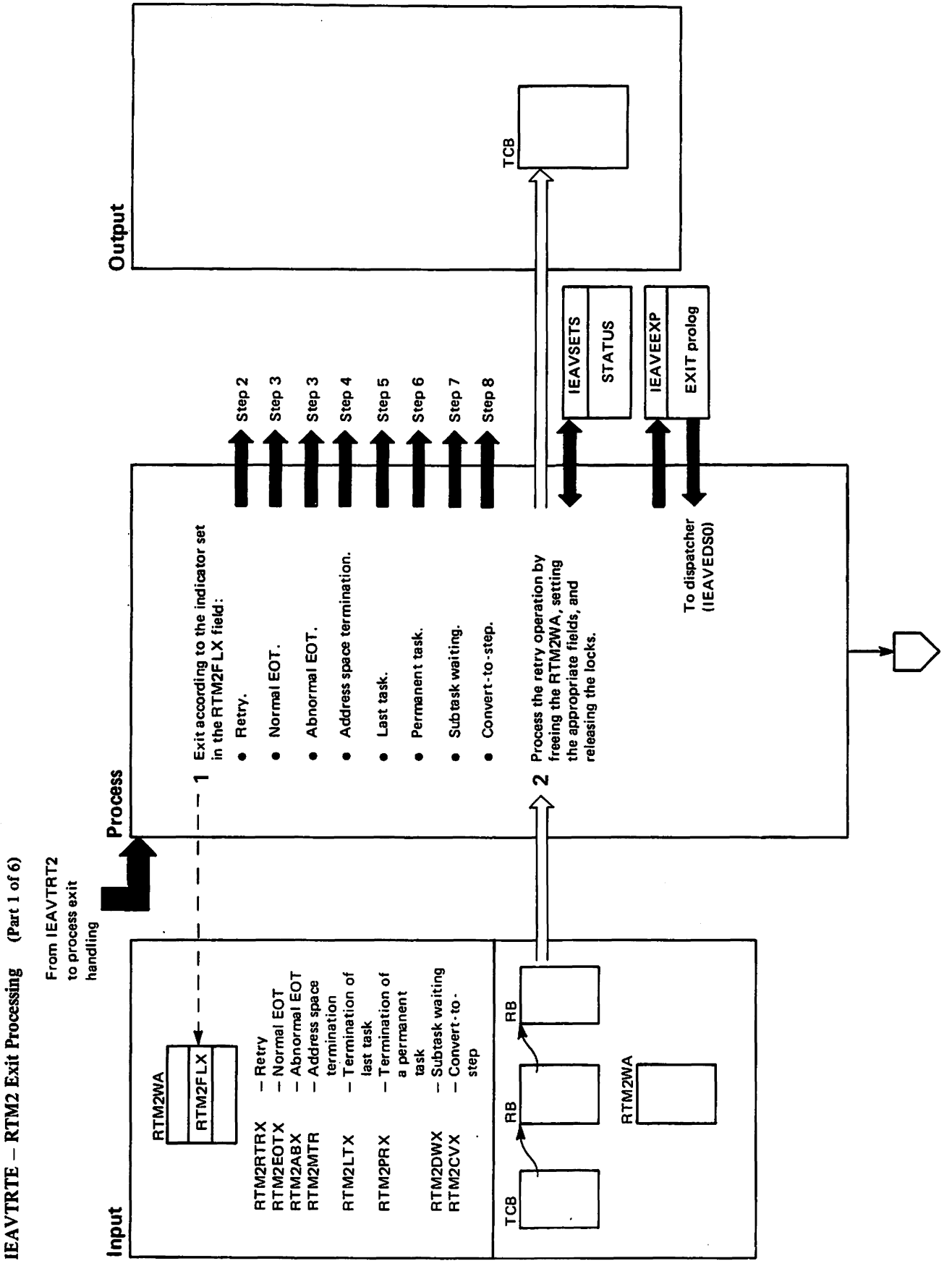


Diagram RTM-22. IEAVTRTE – Recursion Processor 2 (Part 1 of 2)

IEAVTRTE – Recursion Processor 2 (Part 2 of 2)

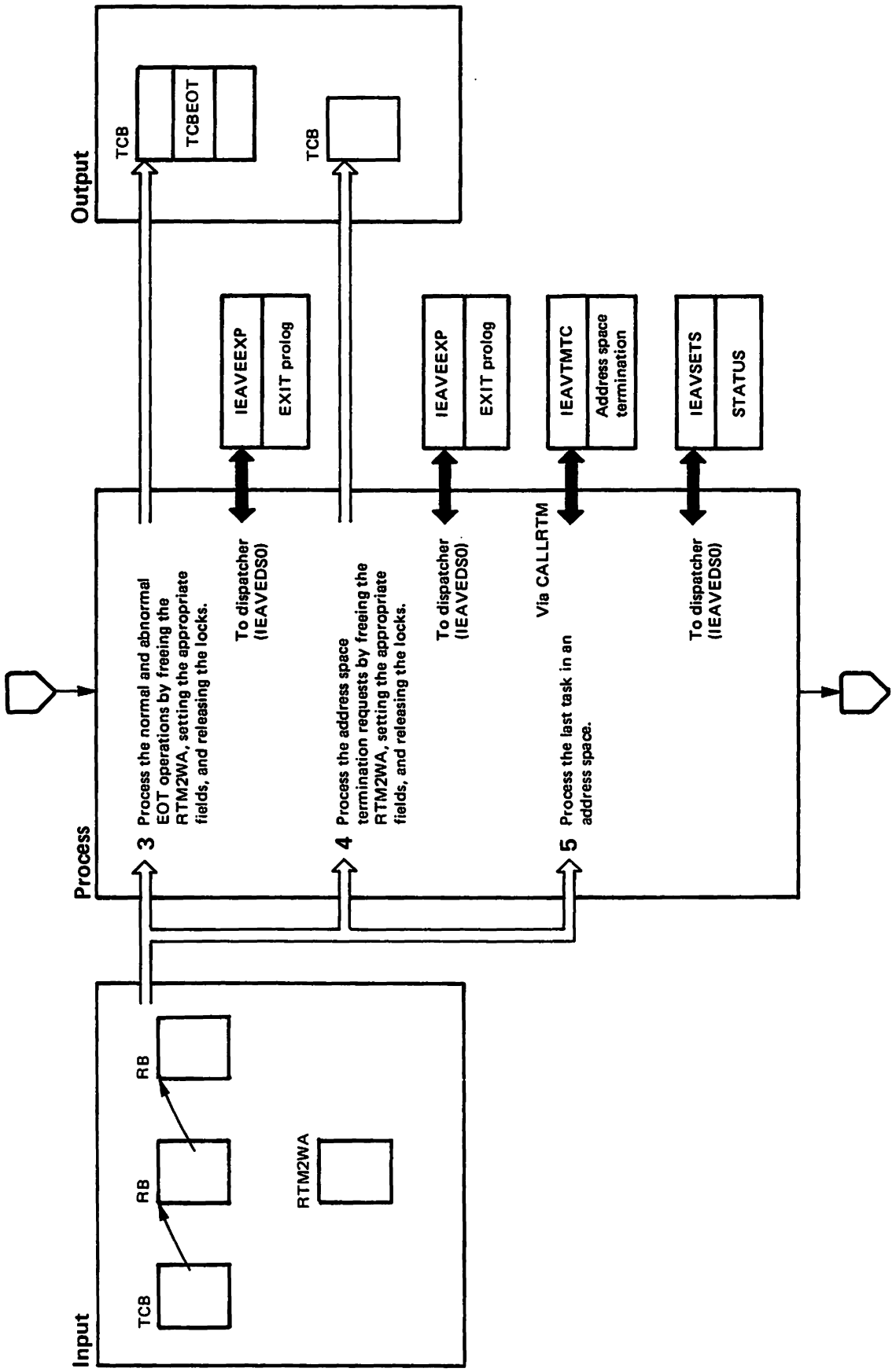
Extended Description	Module	Label
The recursion processor 2 function routes control to a recursion handler for the section of code that failed.		
1 The recursion processor 2 locates the RTM2WA for the failed section. It does this by matching the RTM2SCTX field passed as input with the RTM2SCTC field in the various RTM2WAs that represent the failed sections of the code.	IEAVTRTE	RTERCREX
2 The recursion processor 2 checks for a non-zero value, in order of increasing severity, in four fields in the RTM2WA: • RTM2TRRA – skip a small RTM2 function, such as a resource manager routine. • RTM2SKRA – skip a major RTM2 function, such as synchronizing failing tasks or task recovery. • RTM2STRA – terminate the job step. • RTM2CTRA – terminate the address space. to find the *least* severe recursion handler.		
For RTM2TRRA: The recursion processor 2 clears the section indicator in RTM2SCTR and allows the section to retry; it copies the address and registers that will skip the failing section from that RTM2WA to the current RTM2WA passed as input. This enables the recursion processor 2 to skip this section if it fails again. The fields set in the current RTM2WA are RTM2SKRA and RTM2SFSA. **For RTM2SKRA:** The recursion processor 2 *does not* clear the section indicator in RTM2SCTR; this section must be skipped every time it is reached and not be allowed to execute. **For RTM2STRA and RTM2CTRA:** The recursion processor 2 clears *no* fields. These fields contain the addresses of special recursion routines that handle serious errors.		RTESFRE
3 Prior to giving control to the section of code, the recursion processor 2 restores the registers from the RTM2SFSA field for RTM2SKRA processing, or from RTM2RREG for RTM2TRRA processing.		
4 Control goes to the appropriate section of code, using the address selected in step 2.		

IEAVTRTE - RTM2 EXIT PROCESSING



IEAVTRTE – RTM2 Exit Processing (Part 2 of 6)

Extended Description	Module	Label
RTM2 exits to either EXIT prolog (IEAVEEXP) or STATUS (IEAVSETS), depending on the settings of the RTM2FLX field of the RTM2WA, after task termination or address space termination.		
1 Exit processing determines the type of exit.	IEAVTRTE	
2 IEAVTRTE frees the copied trace table pointed to by RTM2WA (RTM2TRTB) and the current RTM2WA, clears the TCB flags if no RTM2 SVRBs will remain on the RB queue after retry, and reloads the registers that will not be altered by exit (15, 0, 1) from the SVRB. IEAVTRTE then passes control to EXIT prolog.		RTECMEX RTEFREWA

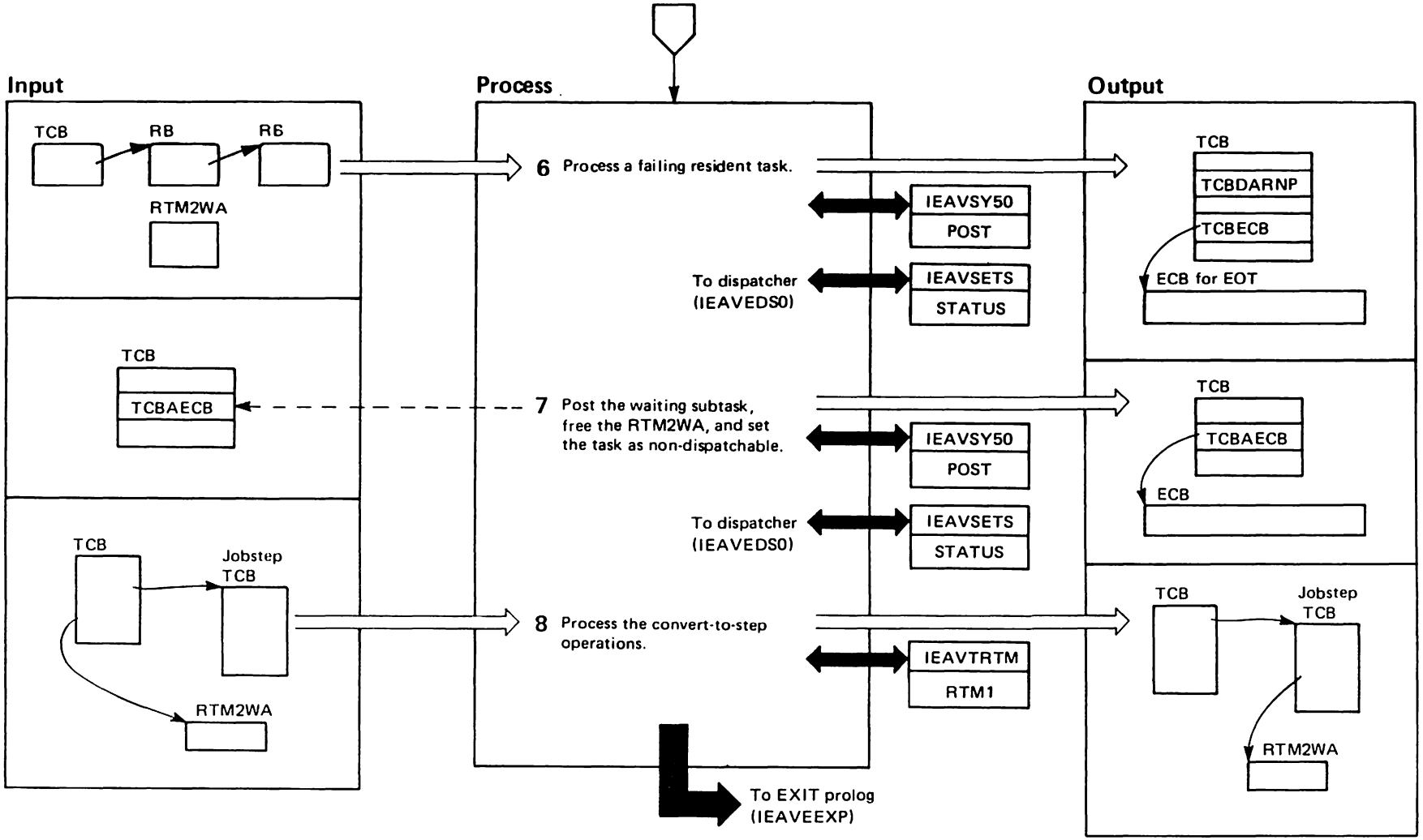


IEAVTRTE – RTM2 Exit Processing (Part 3 of 6)

IEAVTRTE – RTM2 Exit Processing (Part 4 of 6)

Extended Description	Module	Label
3 IEAVTRTE sets the TCBEOT flag to indicate all RTM2 processing is complete for this task. RTM2 frees the copied trace table and all RTM2 work areas. IEAVTRTE passes control to EXIT prolog.		
4 RTM frees the copied trace table and the RTM2WA and passes control to EXIT prolog.		
5 IEAVTRTE terminates the address space using CALLRTM TYPE=MEMTERM. The current task is set non-dispatchable to await completion of memory termination.		RTELTEX

IEAVTRTE – RTM2 Exit Processing (Part 5 of 6)



IEAVTRTE – RTM2 Exit Processing (Part 6 of 6)

Extended Description	Module	Label
6 When a resident task ends, normal processing (which includes freeing the TCB) is impossible. IEAVTRTE posts the end-of-task ECB, to indicate completion, and sets the task permanently non-dispatchable using TCBDARPN.		
7 IEAVTRTE posts the ECB that the subtask is waiting for (located by TCBAECB). The jobstep task sets itself non-dispatchable to await ABTERM. RTM2 is entered from the top for the STEP ABEND. This is not regarded as a recursive entry.		RTESWEX
8 IEAVTRTE queues the current RTM2WA to the jobstep TCB. Then the jobstep task is abnormally terminated with a 20D completion code. The subtask terminates by branching to EXIT prolog. If the jobstep TCB is already in RTM2 processing, it may be necessary to wait for it to complete critical processing before terminating it.		RTECONV RTECNVEX

IEAVTRTF - MODULE DESCRIPTION

DESCRIPTIVE NAME: RTM1 Super FRR Retry Routine

FUNCTION:
This module is the retry routine scheduled by the super FRR, IEAVESPR, when the current stack at the time of error is the RTM stack. IEAVTRTF issues a CALLRTM TYPE=MEMTERM macro to terminate the current address space and cleans up the error environment.

ENTRY POINT: IEAVTRTF

PURPOSE: See function

LINKAGE: BALR from RTM

CALLERS: RTM1 FRR processing

INPUT: Register 15 is the entry point address

OUTPUT: None

EXIT NORMAL: Exit to the dispatcher

EXIT ERROR: System abend code X'0DC'

EXTERNAL REFERENCES:

ROUTINES: None

DATA AREAS: No data areas used

CONTROL BLOCKS:

Common name	Macro ID	Usage	Function
ASCB	IHAASCB	write	Resets the type one SVC flag.
CVT	CVT	read	Obtains RTM and dispatcher routines entry addresses.
LCCA	IHALCCA	read	Obtains processor state - (SRB or TASK).
PSA	IHAPSA	read	Obtains branch entry service routine addresses, lock indicators, control block addresses, etc.
FRRS	IHAFRRS	write	Cleans up FRR stacks.
RT1W	IHART1W	write	Obtains and resets RTM1 information.

TABLES: No tables used.

SERIALIZATION:
IEAVTRTF does not obtain any locks. IEAVTRTF runs disabled to serialize the RTM stacks. IEAVTRTF obtains the RTM super bit to retain disablement during RTM MEMTERM processing.

IEAVTRTF - MODULE OPERATION

IEAVTRTF receives control when the RTM1 stack is the current stack at the time of the error. IEAVESPR, the super FRR, schedules IEAVTRTF as IEAVESPR's retry routine. IEAVTRTF issues a CALLRTM TYPE=MEMTERM macro to terminate the current address space and performs clean up processing of the error environment.

Entry point IEAVTRTF performs the following processing:

- Sets an indicator in the tracking area of RTM1 stack's RTM1 work area so that any errors in the remainder of IEAVTRTF's processing will be protected by RTM1's abort recovery module (IEAVTRTR).
- Issues a CALLRTM TYPE=MEMTERM macro to terminate the current address space.
- Issues a SETFRR macro for each super stack to purge any recovery routines that may have been esatablished. IEAVTRTF sets the logical phase number (LPN) value of each super stack to zero to allow normal error processing to occur in RTM the next time the stack is used. The RTM super stack is not processed at this time to ensure abort recovery protection is not lost.
- Purges all FRRs from the normal stack.
- Determines whether any locks are held.
  - If the LOCAL lock is held, IEAVTRTF releases the LOCAL intersect.
  - If the DISPATCHER lock is held, IEAVTRTF releases the GLOBAL intersect.
- Frees all locks held by the current unit of work.
- Sets an indicator in the tracking area of RTM1 stack's RTM1 work area to ensure that normal recovery can be performed the next time the stack is used.
- Makes the normal stack the current stack.
- Determines whether IEAVTRTF is in SRB or TASK mode and then exits to the appropriate dispatcher entry point.

RECOVERY OPERATION:

This module's entire operation is that of recovery. If any errors occur during IEAVTRTF's processing, IEAVTRTR performs abort processing.

IEAVTRTF - DIAGNOSTIC AIDS

ENTRY POINT NAME: IEAVTRTF

MESSAGES: None

ABEND CODES:

X'0DC' - The current address space is considered unrecoverable and is therefore terminated by issuing a CALLRTM TYPE=MEMTERM macro with a reason code of X'04'.

WAIT STATE CODES: None

RETURN CODES: None

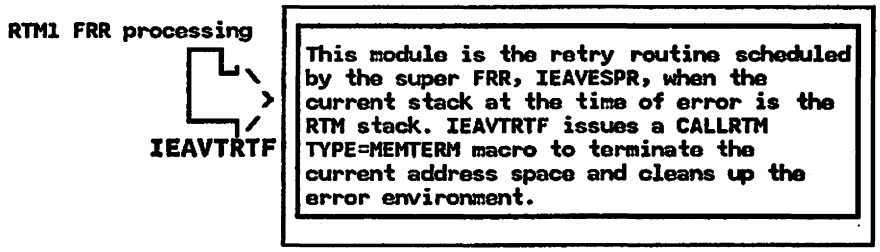
REGISTER CONTENTS ON ENTRY:

Registers 0 - 15 - Irrelevant

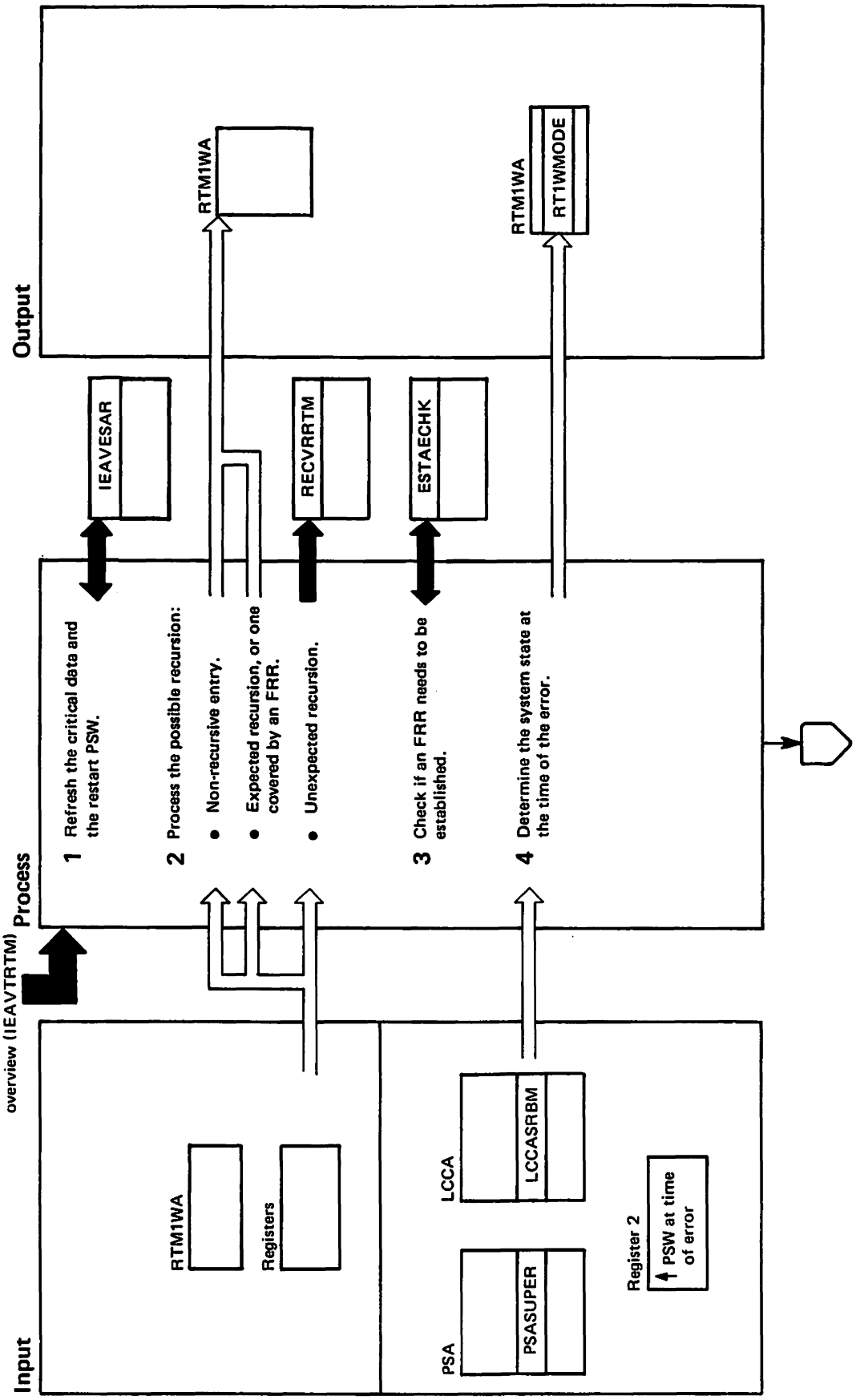
REGISTER CONTENTS ON EXIT:

Registers 0 - 15 - Irrelevant

IEAVTRTF - RTM1 Super FRR Retry Routine



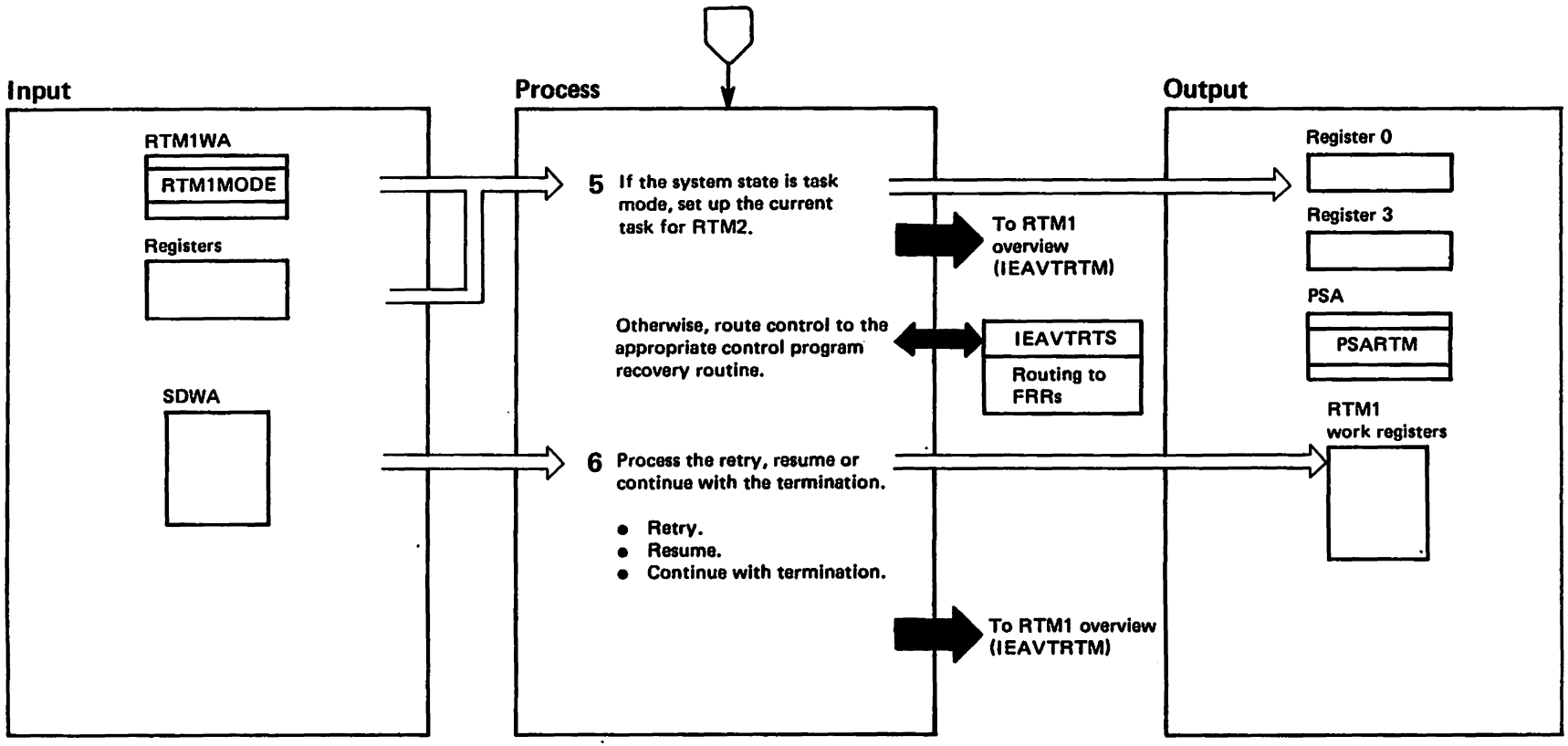
IEAVTRTM - PROCESSING SLIH REQUESTS



IEAVTRTM – Processing SLIH Requests (Part 1 of 4)

IEAVTRTM – Processing SLIH Requests (Part 2 of 4)

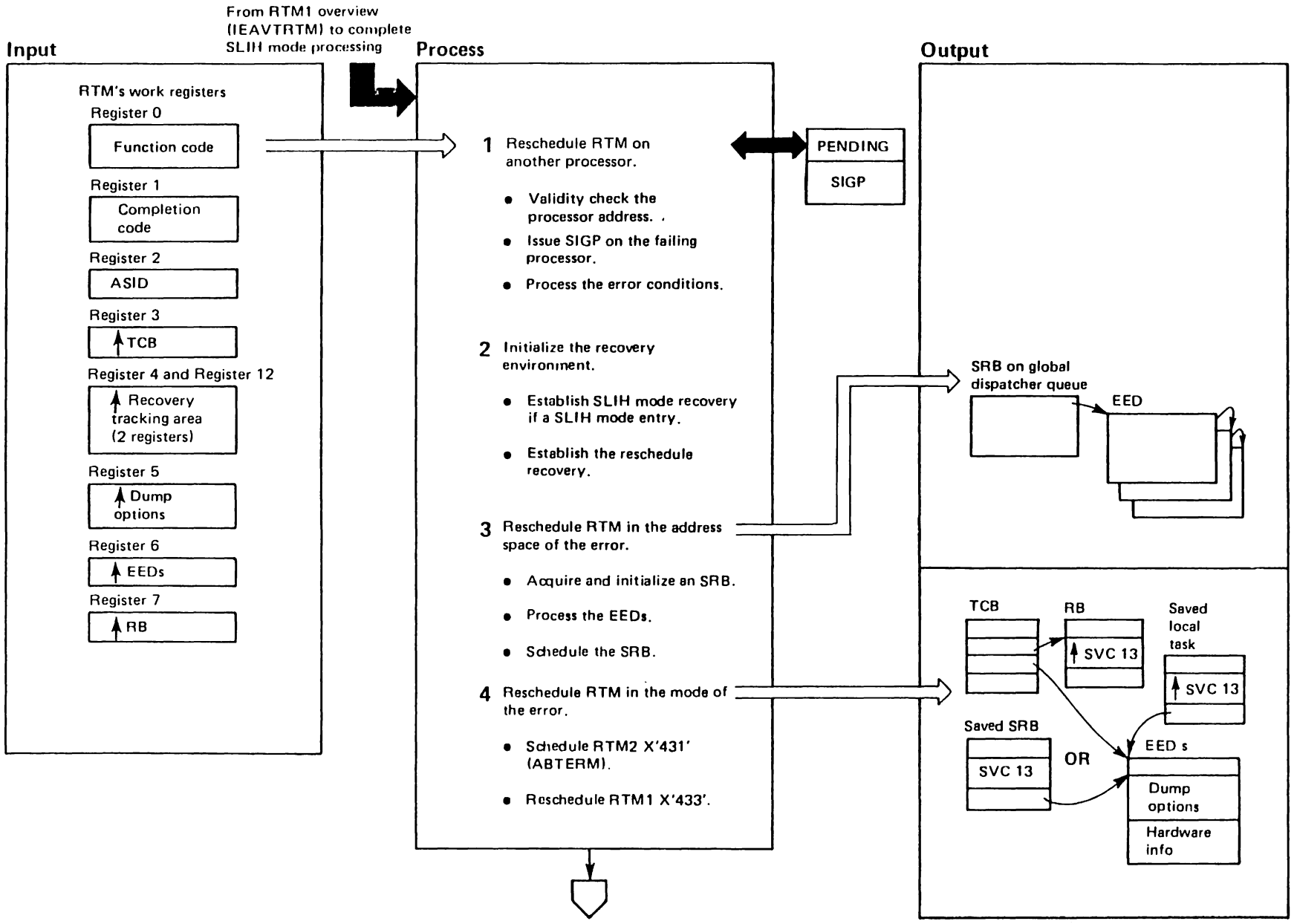
Extended Description	Module	Label
This diagram illustrates the flow of control during RTM1's SLIH processing.		
1 Whenever RTM1 performs SLIH processing, RTM1 first attempts to refresh critical common fixed constants. RTM1 invokes IEAVESAR (the supervisor router routine) to aid in the recovery of the failing processor by refreshing critical system control block pointers and fields (such as PSALCCAV and PSASCWA) that might have been overlaid. RTM1 also attempts, on its own, to refresh the restart new PSW and the CVT restart resource lock word (CVTRSTWD). No recovery of the original error is possible if a second error occurs during the processing of this segment. If a second error occurs, all information from the original error is lost.	IEAVTRTM IEAVESAR	REFRESH
2 RTM1 continues SLIH processing for a non-recursive entry into RTM1, for an anticipated recursive entry, or for a recursion covered by one of RTM1's FRRs. Otherwise, RTM1 processes an unanticipated recursive entry by routing control to a recovery routine (RECVRRTM in module IEAVTRTR) that determines whether any recovery of this recursive error can be performed. *Note:* If RTM1 calls RECVRRTM, control is not returned to IEAVTRTM.	IEAVTRTM	RECURSE
3 RTM1 calls the ESTAECHK routine to see if an FRR needs to be established. If an error occurred in the module IEAVSTA0, ESTAECHK establishes an FRR. For errors in all other modules, no FRR is established.		
4 RTM1 determines the system state at the time of the error by examining indicators in the PSA, LCCA and the PSW at the time of the error. The succeeding flow of control during SLIH mode processing depends on the system state (system mode or task mode). The system state is one of the following: supervisor control mode, physically disabled mode, global spin lock mode, global suspend lock mode, locally locked mode, type 1 SVC mode, SRB mode, EUT (enabled unlocked task with an established FRR) mode, or task mode.		SYSTATE



IEAVTRTM – Processing SLIH Requests (Part 4 of 4)

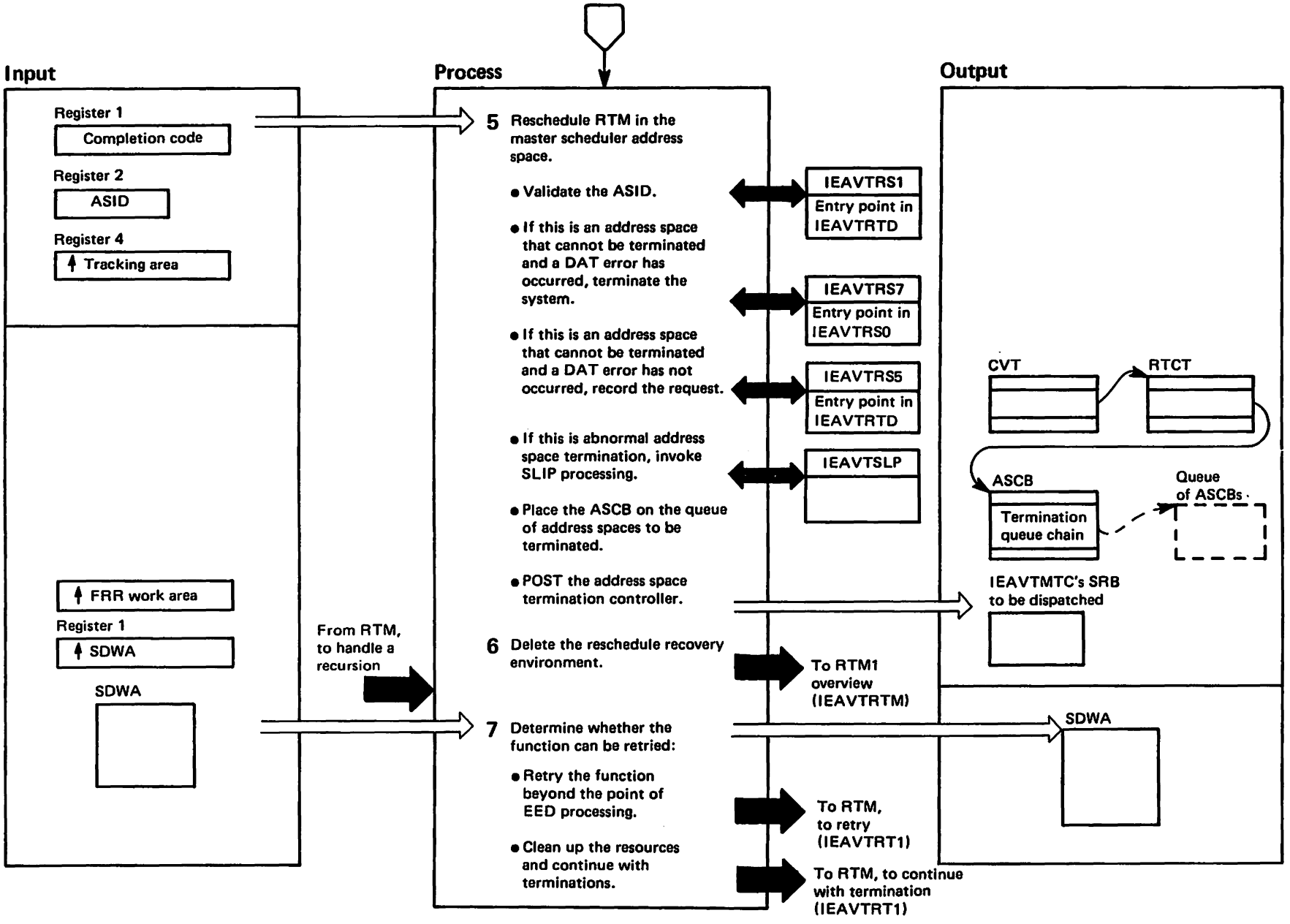
Extended Description	Module	Label
For errors in task mode when the interrupt occurred, RTM1 schedules RTM2 to terminate the current task. RTM1 sets register 0 to indicate task termination and register 3 to point to the current TCB. This allows the reschedule section of IEAVTRTM to schedule the current task for termination. RTM1 sets the PSARTM to supervisor mode and gives control to the reschedule section.		SETUPABT
5 For errors in all other system states (see step 4), control program recovery must be performed. To effect this recovery, the system recovery module (IEAVTRTS) receives control and routes control to any appropriate recovery routine (FRR) associated with the failing routine.		SYSRCVR
6 RTM1 analyzes the output from routing to FRRs in the SDWA to determine the next appropriate action in the handling of the error. For retry requests, control goes to RTM's cleanup and exit processing. For valid resume requests, RTM1 establishes an interface to the reschedule processor function. Otherwise RTM1 continues with termination, setting its work registers to establish the correct interface to the reschedule function.		SYSRCVR
For DATERR entries to RTM1, RTM1 establishes the address space termination interface. RTM1 adjusts the data areas used by the dispatcher to prevent the dispatcher from attempting to reference any private areas in the failing address space.		DATPERC
When the system is in SRB mode, RTM1 determines if the SRB has a dependent task. If a dependent task exists and is in the current address space, RTM1 establishes an ABTERM interface. If the dependent task is not in the current address space, RTM1 establishes an XABTERM interface to cause RTM1 to reenter the task in the task's own address space. If RTM1 determines that there are no dependent tasks, RTM1 passes control to the dispatcher.		SRBPERC

IEAVTRTM – Reschedule RTM1 (Part 1 of 4)



Extended Description	Module	Label
RTM1 performs a reschedule service either when entered in service routine mode, or to complete SLIH mode processing. The basic input to the reschedule function consists of RTM1's work registers, which contain the necessary values to perform the requested service.		
1 RTM1 attempts to process on another processor if a restart interruption caused the entry to RTM1 and the FRR on the current processor validly requested resume. This indicates that the interrupted program on the current processor was waiting for a resource held by another processor.	IEAVTRTM	RESCPU
• If the FRR returned a valid processor address, RTM1 issues a SIGP instruction to another processor. As a result of the SIGP restart interruption, RTM1 begins processing on the signalled processor.		
• If the FRR returned an invalid processor address or if the restart could not be performed, RTM issues an ABEND causing the FRR of the interrupted program to receive control once again to clean up its resources.		
2 If RTM1 received control to perform a service, then some recovery has already been provided by an FRR established in IEAVTRT1 (RT1FRR). If, however, RTM1 had been entered in SLIH mode, no FRR has been established.		RESCHED
• For SLIH mode entries, RTM1 places an FRR (RTMSMFRR) on the FRR stack.		
• Whether RTM1 received control in SLIH mode or in service routine mode, RTM1 places the reschedule FRR (RTMRSFRR) on the stack. This protects the reschedule function by two FRRs. The parameter areas of both FRRs are used to save registers and other information necessary for RTM1's recovery.		
3 RTM1 attempts to reschedule itself in another address space under two conditions: when an ABTERM function has been requested and a nonzero ASID has been provided (cross memory ABTERM), or when the system is in SRB mode and the associated task being terminated cannot be suspended (it cannot obtain the local lock).		
• RTM1 acquires and initializes an SRB. A resource manager (the FREESRBS entry point in IEAVTRTR) is established for the SRB. This resource manager frees the SRB and EEDs if an SRB's related task or address space is terminated before the SRB is dispatched. (See the description of the FREESRBS entry point in IEAVTRTR.)		GETANSRB
• RTM1 obtains EEDs (extended error descriptors) to contain the error registers, PSW, dump options, and error ID. If applicable; a pointer to these EEDs is placed in the SRB.		
• RTM1 schedules the SRB to the specified address space to cause reentry to RTM1 in SRB mode (reentry point IEAVTRTX in IEAVTRT1). Operating as an SRB, RTM1 causes RTM2 to be invoked in the specified address space.		
4 RTM1 performs the reschedule mode function in three cases: for an ABTERM of a task in the current address space (ASID=0); for a STERM paging service; or for post-SLIH mode processing requesting the termination of a task in the current address space.		RESMODE
• RTM1 reschedules page fault errors in either a locally-locked or an SRB routine for reentry into RTM1.		SCHDRTM1
• In all other cases, RTM1 schedules RTM2 to be dispatched from the failing routine. RTM1 places a pointer to an SVC 13 (ABEND) instruction in the resume PSW; this instruction will be the first one executed when the routine in error regains control.		SCHDRTM2

IEAVTRTM – Reschedule RTM1 (Part 3 of 4)

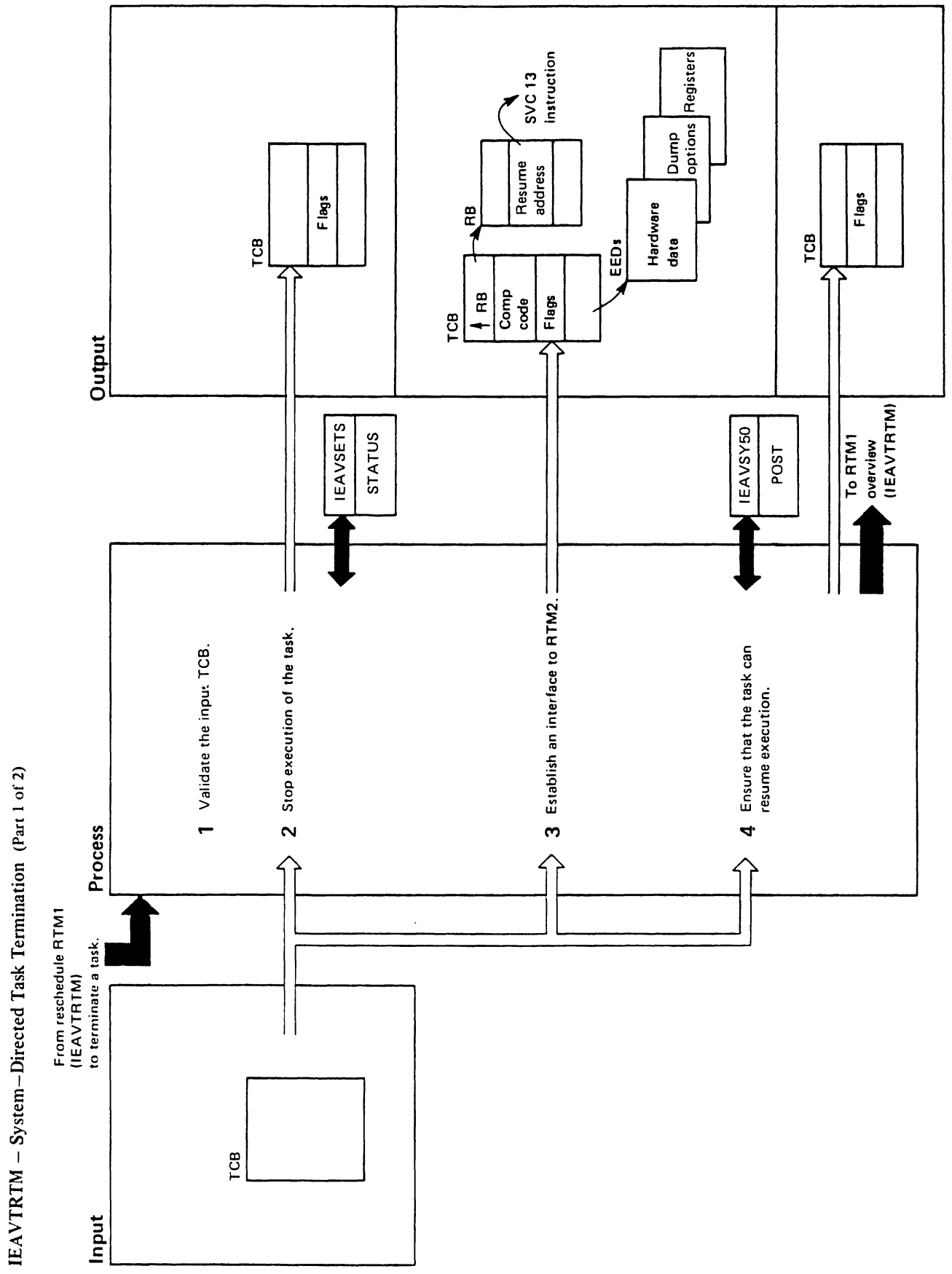


IEAVTRTM – Reschedule RTM1 (Part 4 of 4)

Extended Description	Module	Label
5 If the address space termination function has been requested, RTM1 attempts to schedule the address space termination controller part of RTM (IEAVTMTC), which resides in the master scheduler address space.		
• RTM1 verifies that the input ASID represents a valid address space.	IEAVTRTD	IEAVTRS1
• RTM1 calls IEAVTRS7 when an attempt has been made to terminate an address space because of a DAT error in an address space the system can not allow to be terminated. IEAVTRS7 calls IGFPTERM to terminate the system by issuing message IEA802W and putting the system in an A00 wait state.	IEAVTRS0	IEAVTRS7
• RTM1 calls IEAVTRS5 when a request has been made to terminate an address space that may not be terminated and a DAT error has not occurred. IEAVTRS5 records the fact that an attempt has been made to terminate an address space that may not be terminated by invoking the software recording facility.	IEAVTRTD	IEAVTRS5
• If an address space is being terminated abnormally, RTM1 calls the SLIP processor (IEAVTSLP). For a description of the SLIP processor, see the diagram SLIP Action Processor – Part 1 (IEAVSLP).	IEAVTSLP	
• For a valid ASID, RTM1 places the corresponding ASCB on the address space termination queue. Invalid ASIDs are ignored.		
• RTM1 schedules the address space termination controller's SRB to process the address space termination queue. If the request for address space termination was honored, RTM1 passes a return code of zero to the caller. Otherwise, RTM1 returns a return code of 4.	IEAVTRTM	WAKEMTC
6 RTM1 deletes the SLIH mode FRR, if applicable, and the reschedule FRR. If RTM1 had been entered in SLIH mode, recovery now reverts to the scheme of logical phase recovery routines. (See M.O. diagram IEAVTRTR – RTM1 Recursion Processing for a description of logical phases.)		RESCHED

Extended Description	Module	Label
7 The reschedule RTM1 function protects itself with an FRR (functional recovery routine). The FRR determines whether the reschedule function can be retried past the portion of code where the error occurred, or whether to continue with termination. The FRR requests retry only for errors that occur during processing non-essential to RTM1's handling of the original error. One such example of non-essential processing is EED processing. If the FRR must continue with termination, the FRR cleans up the resources used during the reschedule function.	IEAVTRTR	RTMRSFRR
This provides an additional parameter area used by the reschedule RTM1 FRR (RTMRSFRR). This FRR passes a continue with termination request, when entered.		RTMSMFRR

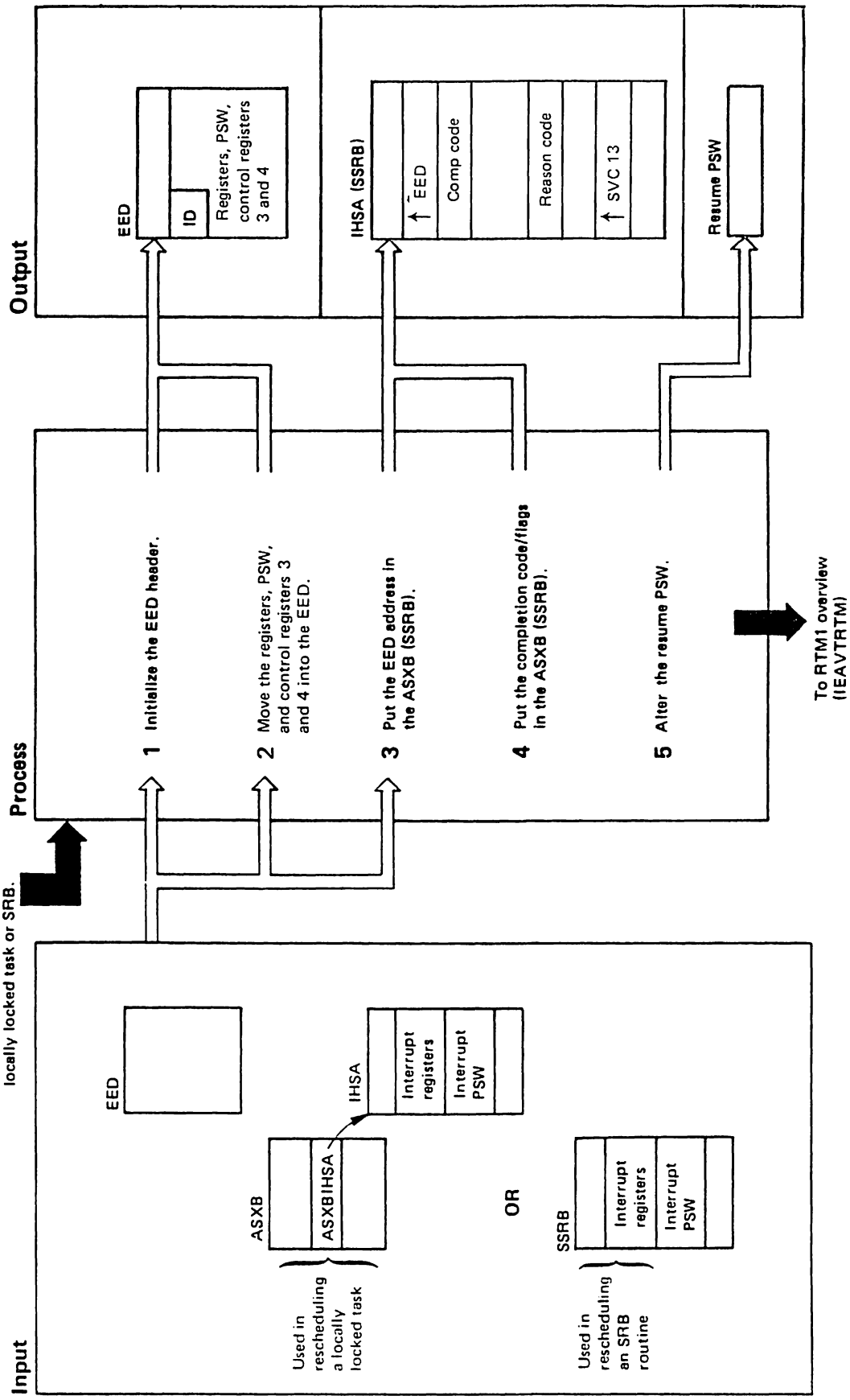
IEAVTRTM - SYSTEM DIRECTED TASK TERMINATION



IEAVTRTM – System–Directed Task Termination (Part 1 of 2)

Extended Description	Module	Label
Since the task recovery and termination process (RTM2) must operate under the TCB being serviced, RTM1 must modify the task control block structure (TCB/RB) so that RTM2 receives control (via SVC 13) as an RB on the effected TCB. RTM1 performs validity checking to prevent erroneous modification of key 0 storage and unnecessary ABEND processing. The task must be stopped because in a multiprocessing environment, the task might be operating on another processor. Resetting the task's non-dispatchability indicators and wait indicators prevents deadlock situations.		
1 RTM1 ensures that the task passed as input exists on the TCB priority queue of the address space. (RTM1 does not check the priority queue if the current task is being terminated.) RTM1 also checks whether or not the task had previously been passed to ABTERM but has not yet executed the SVC 13 (ABEND) instruction. If the TCB is invalid or the ABTERM is already in progress, RTM1 bypasses scheduling the ABTERM function.	IEAVTRTM	VALIDCK
2 RTM1 calls the STATUS routine to stop the execution of the task on another processor in a multiprocessing environment. The task will not be redispatched while RTM1 holds the local lock. For current tasks, no call is necessary since the task has already stopped execution.	IEAVSETS	CKNONCUR
3 RTM1 alters the resume address of the task to that when the task subsequently receives control it will execute an SVC 13 (ABEND) instruction to enter RTM2. The information concerning the error resides in the TCB/RB and EED(s) for use by RTM2.	IEAVTRTM	TCBRB
4 These considerations affect the dispatchability of the TCB/RB being terminated: 1) The wait count in the RB. 2) The non-dispatchability flags in the TCB.		SCHDRTM2
IEAVTRTM enters POST via a branch to reduce the wait count. IEAVTRTM reissues the POST until it takes the RB out of a wait condition (when the wait count becomes 0). STATUS sets the task forced-dispatchable by resetting all non-dispatchability flags. This function breaks any deadlock situations caused by routines that set tasks non-dispatchable and neglect to reset them.	IEAVSY50	

IEAVTRTM - RESCHEDULE LOCALLY LOCKED TASK OR SRB

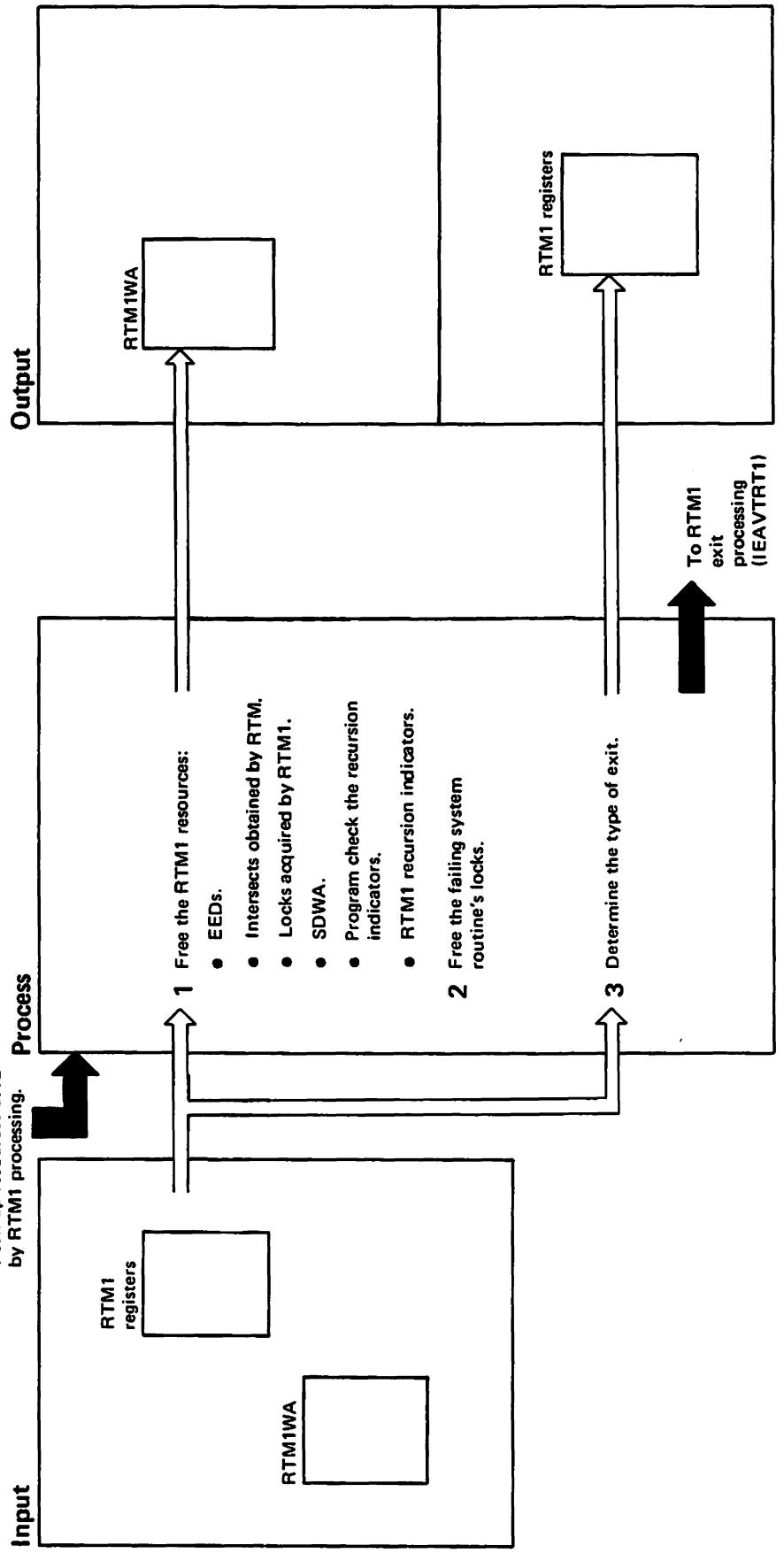


IEAVTRTM – Reschedule Locally Locked Task or SRB (Part 1 of 2)

IEAVTRTM – Reschedule Locally Locked Task or SRB (Part 2 of 2)

Extended Description	Module	Label
When an error occurs during page fault processing for a locally locked task or an SRB routine, RTM1 sets the task to be redispatched from the IHSA (or the SSRB) with an SVC 13 (ABEND) instruction as the first instruction to be executed. When the SVC executes, the SVC IH gets control. It issues a CALLRTM TYPE=SVCERR macro since it appears that an ineligible routine (that is a locked task or SRB routine) has issued an SVC.		
1 RTM1 clears the EED and sets the ID field to indicate a register type.	IEAVTRTM	SCHDRTM1
2 The registers, PSW, and control registers 3 and 4 from the time of the page fault are stored in the EED. For a task, these values come from the IHSA (IHSAGPRS and IHSACPSW) and XSB (XSBXMCRS). For an SRB, these values come from the SSRB (SSRBGPRS and SSRBCPSW) and XSB (XSBXMCRS). For the locally locked task, if the suspended task holds a CML lock, IEAVTRTM issues a CMSET SET macro to establish addressability to the CML-locked address space in order to access the IHSA fields. After accessing the IHSA fields, IEAVTRTM issues a CMSET RESET macro to reestablish the cross memory status at the time of the entry to RTM.		SCHDRTM1
3 RTM1 alters register 0 in the IHSAGPRS field (for a task) or SSRBGPRS field (for an SRB routine) to point to the EED. This becomes input to RTM1 upon reentry.		SCHDRTM1
4 RTM1 places the completion code and options flags in the register 1 slot in the IHSAGPRS or SSRBGPRS field.		SCHDRTM1
5 RTM1 alters the PSW (the IHSACPSW or SSRBCPSW field) to point to an SVC 13 instruction within the RTM module. (This allows RTM1 to uniquely identify the reentry as a reschedule function rather than a regular ABEND request issued by another routine.)		SCHDRTM1

IEAVTRTM - RTM1 CLEAN-UP PROCESSING

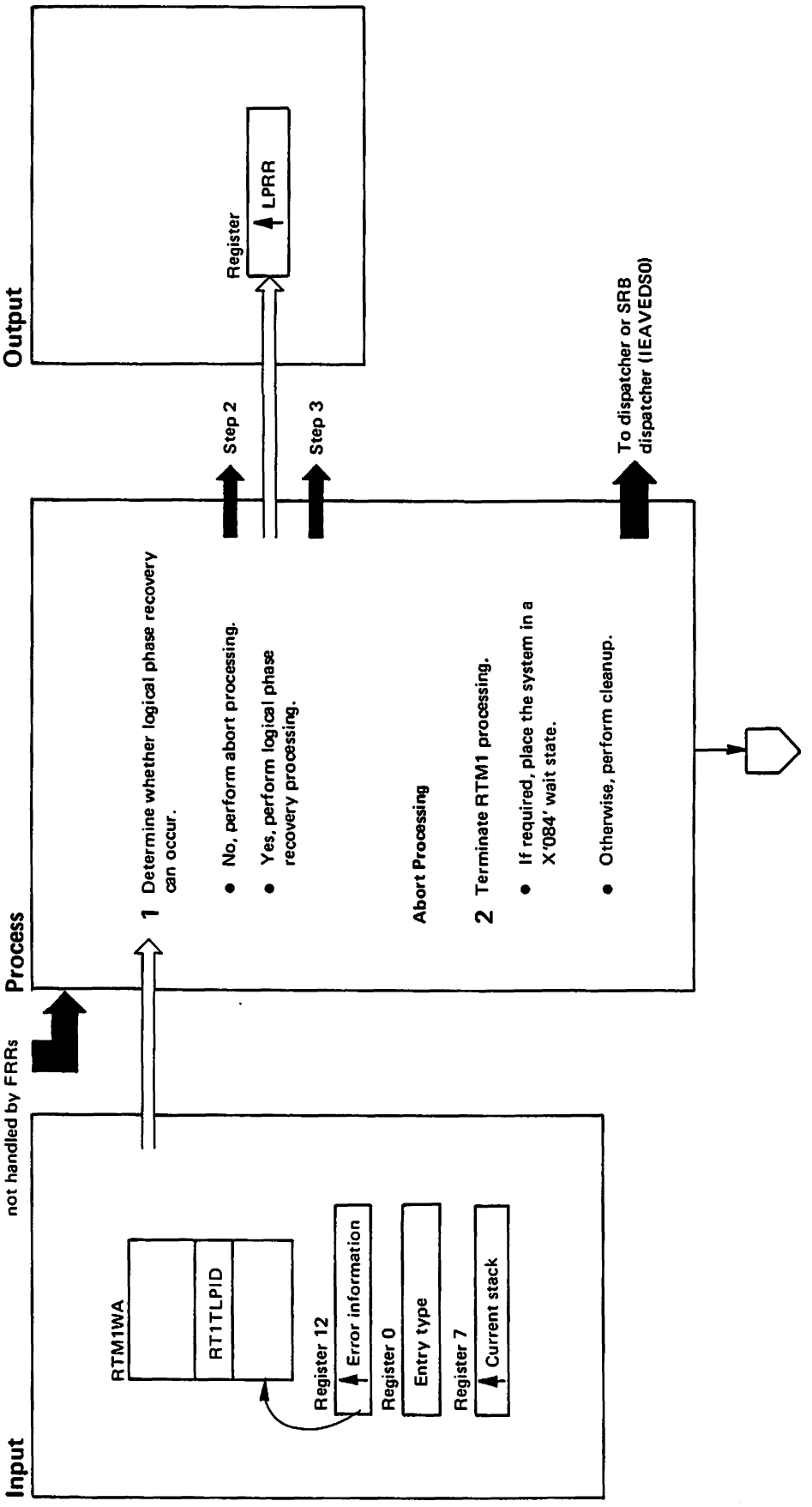


IEAVTRTM – RTM1 Clean-up Processing (Part 1 of 2)

IEAVTRTM – RTM1 Clean-up Processing (Part 2 of 2)

Extended Description	Module	Label
This diagram illustrates the functions performed by RTM1 during cleanup processing.		
1 The cleanup processing frees any locks, EEDs, intersects, or an SDWA acquired during the RTM1 processing that are no longer needed.	IEAVTRTM	SYSCLEAN
2 Cleanup frees all intersects and locks currently held by the failing routine. FREELOCK (EP Name= IEAVFRLK) performs this function.		
3 Cleanup deletes any recursion indicators in the RTM1WA or the current FRR. Cleanup then returns to the entry point/exit point processor with an indication of the type of exit to process.		EXIT

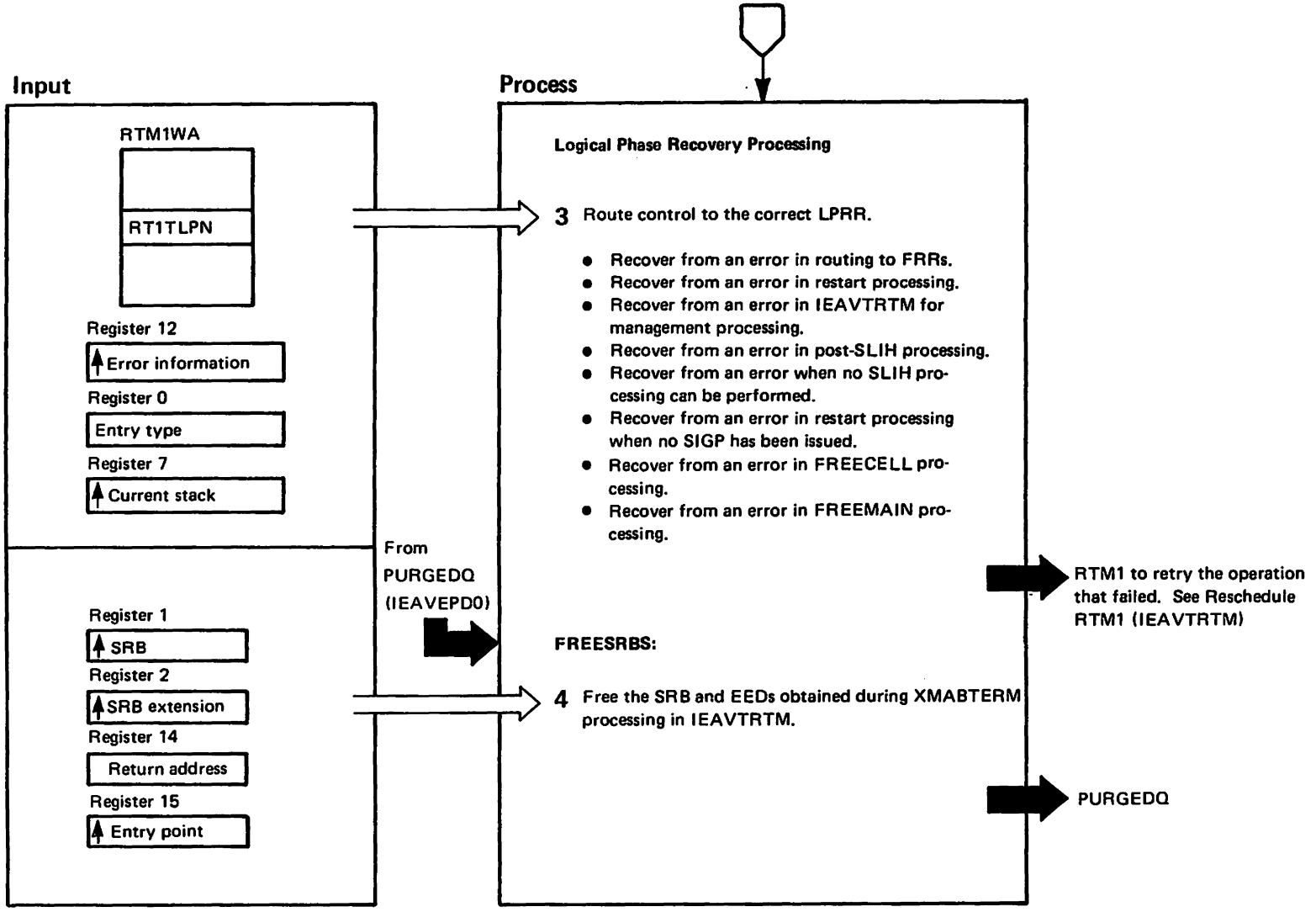
IEAVTRTR - RTM1 RECURSION PROCESSING



IEAVTRTR – RTM1 Recursion Processing (Part 1 of 4)

IEAVTRTR – RTM1 Recursion Processing (Part 2 of 4)

Extended Description	Module	Label
In certain paths through RTM1 processing, recursions cannot be processed by FRRs (functional recovery routines). For example, the phase of the module that actually routes control to FRRs (module IEAVTRTS) cannot be protected by an FRR (that is, if this phase does not work, it cannot route to an FRR to protect itself). To handle these situations where certain phases cannot be protected with an FRR, RTM1 uses LPRRs (logical phase recovery routines). To use LPRRs, RTM1 tracks its own processing. The tracking information consists of two items: • An LPID – A logical phase ID that identifies the LPRR that can process the recursion. • An LPN – A logical phase number that identifies the phase of RTM1's processing in control at the time of the error. Recursion processing routes control to the LPRR identified by the LPID.		
1 When IEAVTRTM discovers a recursive condition in RTM1, it passes control to the recursion processing subroutine. Recursion processing first determines whether a logical phase identifier exists by checking the RT1TLPID field of the RTM1WA. Any time an RTM1 logical phase uses an LPRR for recovery, it sets the RT1TLPID to a non-zero number. If the recursion processing routine finds a non-zero number in that field, it gives control to the correct LPRR. This processing is described in step 3. If it finds a zero, no specific LPRR exists and the abort LPRR receives control.	IEAVTRTR	RECVRRTM
2 The abort processing subroutine initially determines: • If this is a recursive entry (an error encountered in abort processing). or • If an error occurred in IEAVTRTF (RTM super FRR retry routine). If one of the previous conditions is true, the system is placed in a X'084' disabled wait state with an appropriate reason code (8 or C). If one of the previous conditions is *not* true, abort processing does the following: • Issues a CALLRTM TYPE=MEMTERM macro to terminate the current address space. • Resets intersects. • Releases any locks. • Purges all recovery routines from each of the super stacks. • Purges all recovery routines from the normal stack. • Makes the normal stack the current stack. • Sets all super bits to zero. Abort processing then gives control to either the dispatcher or the SRB dispatcher, depending on the mode at the time of the error.		ABORT



IEAVTRTR – RTM1 Recursion Processing (Part 4 of 4)

Extended Description	Module	Label
3 When the RT1TLPID field is non-zero, an LPRR exists. The recursion processing routine routes control to the appropriate LPRR according to the type of recovery desired. (The RT1TLPN field of the RTM1WA indicates the logical phase in control.) RTM1 LPRRs recover from the following:		LPRECOV1
• Errors in routing to FRRs.	IEAVTRTR	SRMDRCOV
• Errors in restart processing.		RVRSTRT
• Errors in mainline SLIH post-processing after routing to FRRs.		RVPOSTSR
• Errors in the mainline SLIH when no routing to FRR processing has been performed.		RVNORTS
• Errors in restart processing when no SIGP (signal processor) macro instruction was issued.		RVNORST
• Errors in FREECELL processing.		RVEEDFRE
• Errors in FREEMAIN processing.		RVFREEMN
• Errors in the management and control routing of RTM1 (IEAVTRT1).		
If the LPRR can recover from the recursive error, it gives control to either IEAVTRTS or IEAVTRTM to resume processing the original error. Otherwise, the LPRR returns to RTM1 main processing to continue processing the new error.		
4 The PURGEDQ routine (IEAVEPD0) calls IEAVTRTR (at entry point FREESRBS, a task and address space termination resource manager) to free an SRB and the EEDs that IEAVTRTM obtained and scheduled during XMABTERM processing. When an SRB's related task or address space was terminated before the SRB was dispatched, FREESRBS returns the SRB to the supervisor SRB pool and the EEDs to RTM1's EED pool.	IEAVTRTR	FREESRBS

IEAVTRTS - MODULE DESCRIPTION

DESCRIPTIVE NAME: RTM FRR Processing Module

FUNCTION:
IEAVTRTS is the main control module for FRR processing. It controls the processing required in routing to FRRs.

ENTRY POINT: IEAVTRTS

PURPOSE: Provides the main functions of this module.

LINKAGE: BALR

CALLERS: IEAVTR10

INPUT: The RTM1 work area

OUTPUT: The RTM1 work area, FRR stack and SDWA

EXIT NORMAL: Returns to IEAVTR10.

EXIT ERROR: Returns to IEAVTR10.

ENTRY POINT: FRRETRN

PURPOSE:
Resumes FRR routing processing after the FRR returns to RTM. The caller, IEAVTRG1 (an RTM1 glue module), is the return point for all FRRs.

LINKAGE: BALR

CALLERS: IEAVTRG1

INPUT: None

OUTPUT: None

EXIT NORMAL:
The exit is from module IEAVTRTS. This entry is a continuation of IEAVTRTS.

EXIT ERROR: There are no exit error conditions.

ENTRY POINT: EUTRTM1A

PURPOSE:
Acquires a new dynamic area for IEAVTRTS through the consecutive CALL and ENTRY statements. IEAVTRTS freed the original dynamic area when it had to enable itself to obtain the local lock for EUT FRR processing.

LINKAGE: BALR

CALLERS: IEAVTRTS

INPUT: None

OUTPUT: None

EXIT NORMAL: The exit is from module IEAVTRTS.

EXIT ERROR: There are no exit error conditions.

ENTRY POINT: EUTRTM1B

PURPOSE:
Acquires a new dynamic area for IEAVTRTS through the consecutive CALL and ENTRY statements. IEAVTRTS freed the

IEAVTRTS - MODULE DESCRIPTION (Continued)

original dynamic area when it had to enable itself to obtain the local lock for EUT FRR processing.

LINKAGE: BALR

CALLERS: IEAVTRTS

INPUT: None

OUTPUT: None

EXIT NORMAL: The exit is from module IEAVTRTS.

EXIT ERROR: There are no exit error conditions.

EXTERNAL REFERENCES:

ROUTINES:

- IEAVTR1C - SDWA Selection and Initialization Module via CALL
- IEAVTR1D - Retry/percolation Processing, Lock freeing and mode switching Module (IEAVTR1D is an entry point in module IEAVTR1C) via CALL
- IEAVTR1F - RTM1 FRR Routing Pre-Processor via CALL
- IEAVTR1G - RTM1 GTF Processing Module via CALL
- IEAVTR1R - RTM1 RECORD Interface Module via CALL
- IEAVTR1X - RTM1 CMSET Interface Module via CALL
- IEAVTSLP - SLIP Action Processor via CALL
- IEAVTSSX - SLIP Space Switch Handler via CALL

CONTROL BLOCKS:

Common name	Macro ID	Usage	Function
CVT	CVT	read and write	Sets the restart resource word and obtains the SLIP control block pointer.
ASCB	IHAASCB	read	Sets information via the INTSECT macro.
FRRS	IHAFRRS	read and write	Obtains FRR entry information and set RTM control information in FRR entries
LCCA	IHALCCA	read	Obtains spin indicators.
PSA	IHAPSA	read	Obtains super bit, lock status, current ASCB and TCB addresses, and FRR stack information.
RT1W	IHART1W	read and write	Obtains and sets RTM1 control information.
SDWA	IHASDWA	read and write	Obtains and sets FRR input and output information.
SHDR	IHASHDR	read and write	Obtains and sets SLIP control information.
SVT	IHASVT	write	Set by the INTSECT macro.
TCB	IHATCB	write	Sets fields to cause RTM2 to serialize with RTM1.

IEAVTRTS - MODULE OPERATION

IEAVTRTS is comprised of several entry points. The secondary entry points into IEAVTRTS are a result of a structural requirement more than a functional one. There is only one primary function performed by this module, the overall control of the FRR routing process.

The main processing of IEAVTRTS is as follows:

1) Acquires and initializes a system diagnostic work area (SDWA) for use by FRRs. IEAVTRTS calls IEAVTR1C to perform this function. IEAVTR1C handles the different processing that is required of initial entry verses recursive entry.

2) Invokes the SLIP processing module, IEAVTSLP, to check abnormal termination conditions, such as completion code and reason code, against the currently active SLIP traps, and to determine if any special SLIP action is required for this particular error occurrence.

3) Obtains and saves the current lock status. Many RTM decisions are based on the type of locks held.

4) Determines if there are any FRRs available or eligible to receive control. IEAVTRTS calls IEAVTR1F to examine the current state of the system and the FRR stack. IEAVTR1F then determines if there are any FRRs that should be routed to.

5) Serializes processing with RTM2 for EUT FRRs and prevents EUT FRRs from retrying if the error is a cancel type abnormal termination.

   To accomplish the RTM2 synchronization, IEAVTRTS must enable itself to obtain the local lock. Before enabling, IEAVTRTS must free the dynamic area storage stack, thus losing its dynamic area. (The dynamic areas in RTM1 are useable only while disabled.) After obtaining the local lock, IEAVTRTS disables again and acquires a new dynamic area by calling entry point EUTRTM1A. The only use of this entry point is to acquire a new dynamic area.

6) Prepares the FRR cross memory addressing environment. (See module IEAVTR1X.)

7) Gives control to the most recently established FRR via a LPSW instruction. The return address established for the FRR is within module IEAVTRG1. IEAVTRG1 is a glue module responsible for reestablishing RTM's addressing environment and calling IEAVTRTS at entry point FRRETRN to continue further FRR processing.

8) If RTM2 is waiting for this task to complete recovery processing, prevents retry from an EUT FRR.

   To accomplish the RTM2 synchronization, IEAVTRTS must enable itself to obtain the local lock. Before enabling, IEAVTRTS must free the dynamic area storage stack, thus losing its dynamic area. After obtaining the local lock, IEAVTRTS disables again and acquires a new dynamic area by calling entry point EUTRTM1B. The only use of this entry point is to acquire a new dynamic area.

9) Ensures that the SDWA describing each new software error is written to the SYS1.LOGREC data set.

10) Performs the action of percolation, retry or resume as

IEAVTRTS - MODULE OPERATION (Continued)

determined by module IEAVTR1D.

11) Repeats steps 3 through 8 until all FRRs have been routed to or until an FRR retries or resumes.

12) Returns to IEAVTR1O to process the final steps in an FRR retry or resume request, or to percolate the error to RTM2.

IEAVTRTS - DIAGNOSTIC AIDS

ENTRY POINT NAMES: IEAVTRTS
FRRETRN
EUTRTM1A
EUTRTM1B

MESSAGES: None

ABEND CODES: None

WAIT STATE CODES: None

RETURN CODES: None

REGISTER CONTENTS ON ENTRY:

ENTRY POINT IEAVTRTS:

Registers 0 - 14 - Irrelevant
Register 15 - Entry point address

ENTRY POINT FRRETRN:

Registers 0 - 14 - Irrelevant
Register 15 - Entry point address

ENTRY POINT EUTRTM1A:

Registers 0 - 14 - Irrelevant
Register 15 - Entry point address

ENTRY POINT EUTRTM1B:

Registers 0 - 14 - Irrelevant
Register 15 - Entry point address

REGISTER CONTENTS ON EXIT:

ENTRY POINT IEAVTRTS:

Registers 0 - Unpredictable
Registers 1 - Abnormal termination completion code
if attempt to obtain an SDWA failed
Otherwise, unpredictable
Registers 2 - 13 - Unpredictable
Register 14 - Return point address
Register 15 - Abnormal termination completion code

ENTRY POINT FRRETRN:

Registers 0 - 15 - Irrelevant

ENTRY POINT EUTRTM1A:

Registers 0 - 15 - Irrelevant

ENTRY POINT EUTRTM1B:

Registers 0 - 15 - Irrelevant

IEAVTRTV - MODULE DESCRIPTION

DESCRIPTIVE NAME: RTM PSACSTK Verification Module

FUNCTION:
This module checks the value in the PSACSTK to determine if it is a valid FRR (functional recovery routine) stack address when RTM1 is entered in SLIH (second-level interruption handler) mode.

ENTRY POINT: IEAVTRTV

PURPOSE: See function

LINKAGE: BALR

CALLERS: IEAVTRT1

INPUT:
Registers in use by IEAVTRT1
Register 14 - Return address

OUTPUT: None

EXIT NORMAL: Returns to IEAVTRT1

ENTRY POINT: IEAVTEXS

PURPOSE: SEE ENTRY POINT IEAVTEXS FOR DESCRIPTION

LINKAGE: None

CALLERS: None

INPUT: None

OUTPUT: None

EXIT NORMAL: Returns to IEAVTRT1

EXIT ERROR: The system enters a X'084' wait state.

EXTERNAL REFERENCES:

ROUTINES:
IGFPTERM - Terminates the system with a X'084' wait state.

DATA AREAS: None referenced

CONTROL BLOCKS:

Common name	Macro ID	Usage	Function
FRRS	IHAFRRS	read	Obtains the length of each super FRR stack.
LCCA	IHALCCA	read	Maps the WSAVTC
PCCA	IHAPCCA	read	Contains excessive spin length factor
PSA	IHAPSA	read	Obtains the address of the current FRR stack (PSACSTK) and the normal FRR stack (PSANSTK).
RTIW	IHARTIW	read	Obtains the stack mapping.
SDWA	IHASDWA	read	Obtains the length of an SDWA in a super FRR stack.
YSTAK	IHAYSTAK	read	Obtains the number of entries in each super FRR stack.
WSAVTC	IHAWSAVT	read	Obtains the addresses of the frr stack save areas.

SERIALIZATION: IEAVTRTV does not obtain any locks.

IEAVTRTV - MODULE OPERATION

IEAVTRTV receives control to check whether the PSACSTK field has a valid FRR stack address.

Entry point IEAVTRTV receives control from IEAVTRT1 and checks through these two sources:

1. The normal FRR stack
2. The super FRR stacks

IEAVTRTV performs the following three processing steps until a valid FRR stack address has been found:

1. Checks the normal FRR stack.

   IEAVTRTV compares the contents of the PSACSTK field (which contains the address of the currently used FRR stack) with the address of the PSASTAK field (which contains the normal FRR stack). If they are equal, RTM1 has access to the normal FRR stack. IEAVTRTV returns to IEAVTRT1.

2. Checks the super FRR stacks.

   IEAVTRTV compares the contents of the PSACSTK field with the address of each super FRR stack. If a match is found, RTM1 has access to a super FRR stack. IEAVTRTV returns to IEAVTRT1.

If no valid FRR stack address is found, RTM1 cannot continue its initialization processing. IEAVTRTV invokes IGFPTERM to put the system in a X'084' non-restartable wait state (reason code=4) and to issue message IEA797W.

RECOVERY OPERATION:

There is no recovery processing for IEAVTRTV. If a valid FRR stack address cannot be found, IEAVTRTV invokes IGFPTERM to put the system in a X'084' non-restartable wait state (reason code=4) and to issue message IEA797W.

IEAVTRTV - DIAGNOSTIC AIDS

ENTRY POINT NAMES: IEAVTRTV
IEAVTEXS

MESSAGES:

IEA797W - The pointer to the current FRR stack is not valid.

ABEND CODES: None

WAIT STATE CODES:

X'084' RTM (recovery termination management) has encountered an uncorrectable error while trying to provide recovery or termination to some unit of work in the system. The wait state PSW (bits 40-47) contains a reason code of 4 which indicates that the PSACSTK does not have a valid FRR stack address.

RETURN CODES: None

REGISTER CONTENTS ON ENTRY:

ENTRY POINT IEAVTRTV:

Registers 0-13 - Irrelevant
Register 14 - Return address to IEAVTRT1
Register 15 - Irrelevant

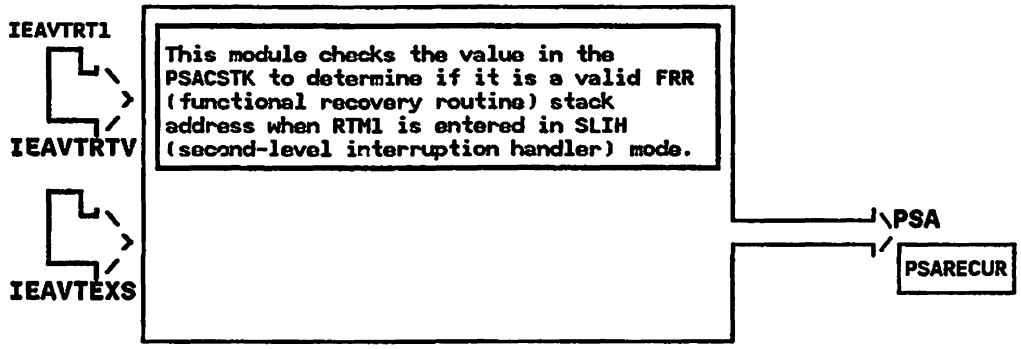
ENTRY POINT IEAVTEXS: Irrelevant

REGISTER CONTENTS ON EXIT:

EXIT NORMAL:

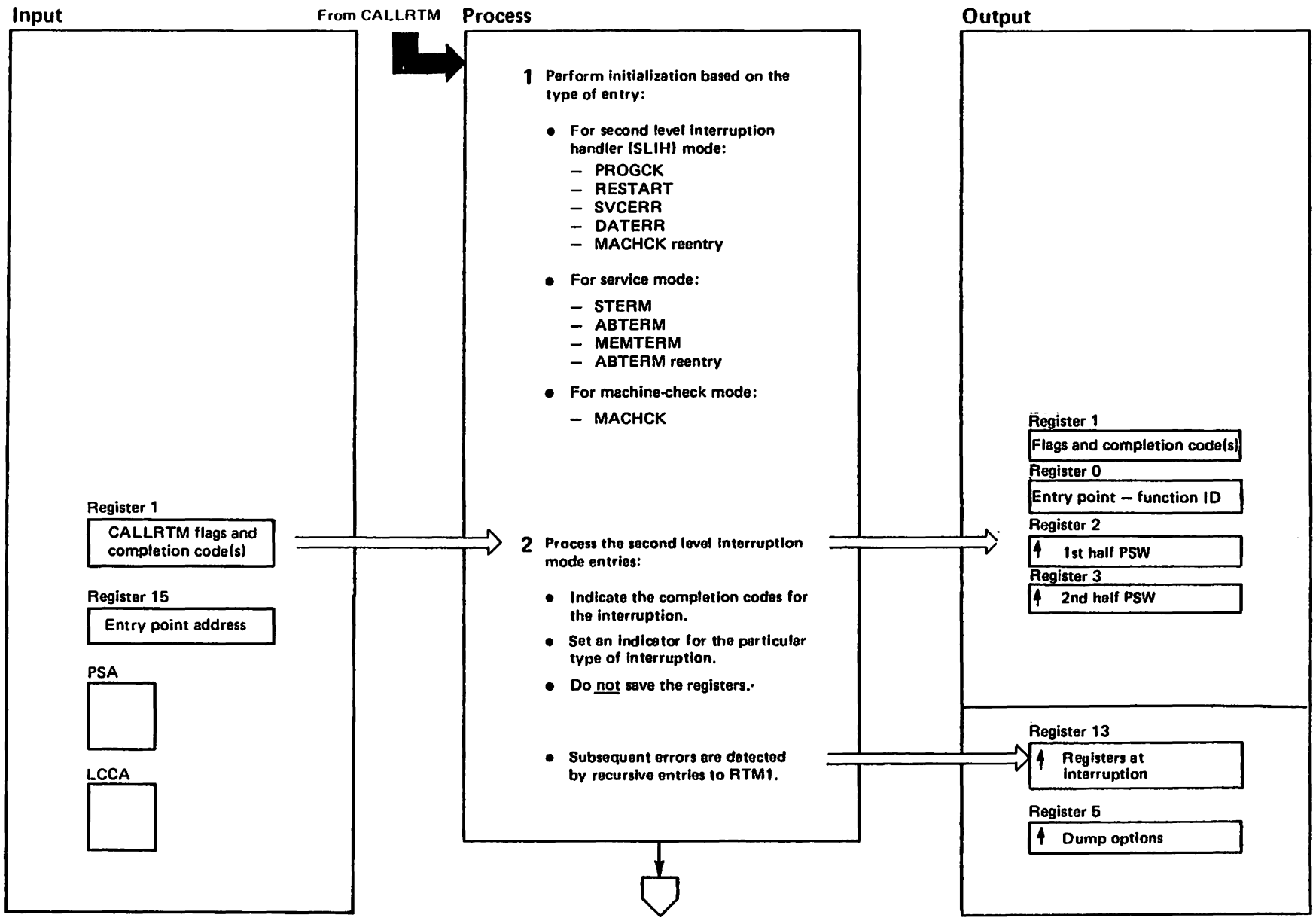
Registers 0-15 - Irrelevant

IEAVTRTV - RTM PSACSTK Verification Module



This page left blank intentionally.

IEAVTRT1 – RTM1 Initialization (Part 1 of 6)



IEAVTRT1 – RTM1 Initialization (Part 2 of 6)

Extended Description	Module	Label
RTM1 processing receives control via the CALLRTM macro. The expansion of the macro locates the correct entry point address into RTM1 from the RTM1 branch table (pointed to by the CVTBTERM field of the CVT). RTM1 initialization combines the various entry point data to create a common interface for RTM1 processing.		
1 RTM1 initialization consists of saving the registers, saving the cross memory information (control registers 3 and 4), indicating the completion codes, and, for a service mode entry, establishing a recovery environment based upon the type of entry. IEAVTRTV accomplishes the latter for all SLIH mode entry points by verifying that the address in the PSACSTK points to a valid FRR stack. If it does, RTM1's initialization processing continues. If it does not, then IEAVTRTV invokes IGFPTERM to put the system in a X'084' wait state and to issue message IEA797W. (See the M.O. diagram IEAVTRTV – RTM PSACSTK Verification Module). RTM1 performs three types of initialization; one based on requests made by the interruption handlers (IH), another based on a service request for an RTM1 service; and the last for machine check interruptions..	IEAVTRT1 IEAVTRTV	
2 RTM1 initializations prepares the following entry points for SLIH mode:		
• **Program check entry point** – Used by the program check IH when an invalid page fault or program check occurs. When the program check IH passes RTM1 a completion code, the registers and PSW have been saved by the program check IH in the primary save areas of the PSA and LCCA (logical configuration communications area). When RTM1 does not receive a completion code, initialization processing builds one from the interruption code and the error information in the secondary save area in the LCCA. (See the Supervisor Control section of the *System Logic Library* for a description of the program check IH and the different save areas used.)		PROGCK
• **Restart entry point** – Used by the restart IH after the operator has requested RTM processing. The subsequent handling of a restart request in RTM is tailored to loop-breaking, that is, a looping program is not allowed to retry, and a validly spinning program is allowed to request RTM to interrupt the program that owns the resource being waited upon. The restart IH has saved the registers in the LCCA and the resume PSW in the PSA.		RESTART
• **SVC IH entry point** – Used whenever an SVC is issued by a routine that is locked, disabled, in SRB mode, in EUT mode (enabled unlocked task with an established FRR), or is under supervisor control (non-dispatchable supervisory functions). Upon entry, IEAVTRT1 issues a CMSET SET to the home address space. If the SVC is not an SVC 13 (an ABEND SVC), RTM1 interprets it to be an error. The SVC IH has saved the registers, the cross memory information (control registers 3 and 4), and the PSW. (See the SVC IH (IEAVESVC) diagram for a complete description of the SVC IH.) If the SVC was an SVC 13, and it was not issued by RTM, IEAVTRT1 interprets the entry point as an explicit request for ABEND processing. If the SVC 13 was issued by RTM, error information (registers, cross memory information, and the PSW) was put in an EED (extended error descriptor) by the RTM1 Subroutines module (IEAVTRS0). IEAVTRT1 sets up the interface to RTM1 mainline by using the information in the EED. If an SRB is percolating to its related task's FRR, IEAVTRT1 obtains the completion code and the EED address from the TCB.		SVCERR

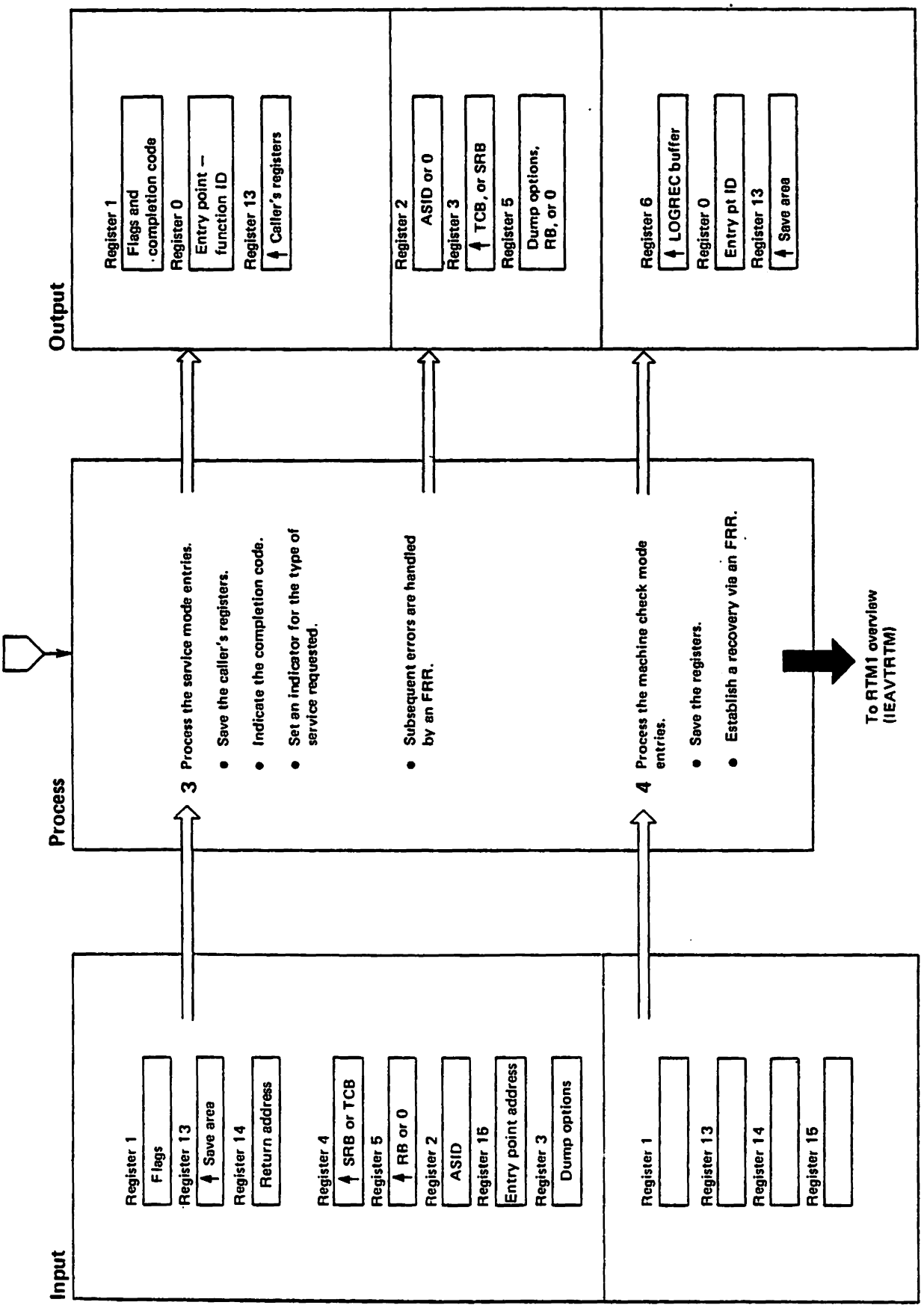
IEAVTRT1 - RTM1 Initialization (Part 3 of 6)

No diagram

Extended Description continued on next page.

Extended Description	Module	Label
2 (continued)		
• **DATERR entry point** – Used by the program check IH when a recursive translation exception occurs during either the program check IH's processing, or RTM1's FRR processing. Before calling RTM1, the program check IH attempted to circumvent any further translation failures by altering control register 1 (the segment table origin register, which contains the master's STOR). Both the primary and secondary ASIDs equal the master's ASID. The PSW S-bit is turned off. The current stack is that of the PC FLIH and IEAVTRT1 gives any FRRs on that stack control in the same environment in which RTM was entered. In this case, IEAVTRT1 does not change the PSAAOLD from the ASCB of the home address space at the time of the DAT error. IEAVTRT1 makes a check to see if the DAT error occurred in the home address space or in another address space. If the DAT error occurred in the home address space, IEAVTRT1 passes control to mainline RTM1 to process the DAT error. If errors occur again, the program check IH places the system in a disabled wait state. RTM1 does not allow normal recovery processing to occur during DATERR processing since the private areas of the failing address space are no longer addressable. If a supervisor control routine was in control when the original error occurred, then IEAVTRT1 gives its FRR control with a special indication to warn it that private areas are no longer addressable (the ASID of the failing address space is specified on the CALLRTM macro. This ASID is placed in the SDWA (SDWAFMID) and passed to the FRRs.) The super FRR may recover the address space or terminate it (via MEMTERM). If a super FRR is not available, RTM1 bypasses all recovery, records the incident and terminates the address space. If the DAT error occurred in an address space other than the home address space, IEAVTRT1 conditionally obtains an SRB,		DATERR
If the SRB is obtained, IEAVTRT1 initializes it to run in the master's address space with the entry point DATMEM in IEAVTRT1 and then schedules it (DATMEM will issue the MEMTERM macro for the address space suffering the DAT error). Whether the SRB is scheduled or not, IEAVTRT1 loads register 1 with a completion code of X'0FC' and gives the PROGCK entry point in IEAVTRT2 control.		
• **MACHCK reentry** – Used when RTM1 sets up MCH (machine check handler) or ACR (alternate CPU recovery) for re-entry into RTM1 after RTM1 was initially entered for a machine check. RTM1 uses this entry to attempt software recovery processing if a machine check caused software damage. RTM1 is entered in the home mode. On entry, register 4 points to two words that contain the contents of control registers 3 and 4 at the time of the error. The passed PSW indicates the value of the PSW S-bit at the time of the error.		IEAVTRTN

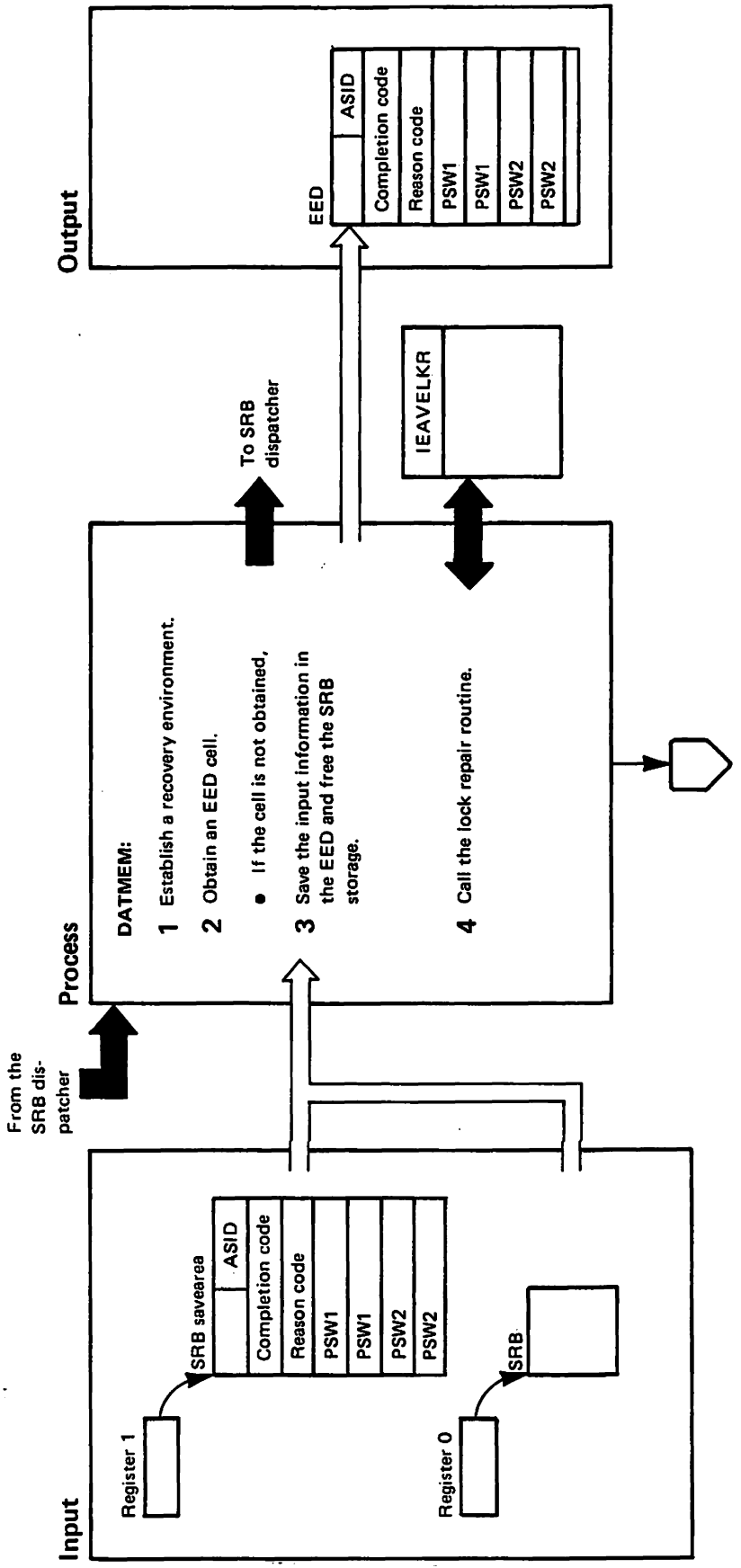
IEAVTRT1 - RTM1 Initialization (Part 5 of 6)



IEAVTRT1 – RTM1 Initialization (Part 6 of 6)

Extended Description	Module	Label
3 RTM1 initialization prepares the following entry points for service mode entries:		
• **STERM entry point** – Used by the reset subroutine of real storage management when an error occurs while processing a page fault. RTM1 is entered in home mode. The TCB must be addressable on entry to RTM1. Initialization processing only saves register 14 (the return address). The routine that suffered the paging error is forced to issue an ABEND instruction (SVC 13) to cause the linkage to RTM for recovery and termination services. Initialization processing for this entry point passes the address of the TCB or SRB that suffered the error. If a task suffered the error, the address of the RB is also passed. RTM retrieves the cross memory information of the error from the XSB associated with the SSRB, the RB, or the IHSA of the locked address space.		STERM
• **ABTERM entry points** – Used by key 0, supervisor state routines to set a task up for entry to RTM2 for ABEND. There are two types of ABTERM entry: ABTERM with the ASID option; and ABTERM without the ASID.		
ABTERM with the ASID option is a request to terminate a task in an address space other than the current one. RTM1 can get control in any cross memory mode. RTM1 executes this service mode request in the caller's environment and returns to the caller in the same environment. RTM1 schedules itself as an SRB into the specified address space to perform the ABTERM request. RTM1 saves the caller's registers in a caller-supplied save area.		XABTERM
ABTERM without the ASID option is a request to terminate a task in the current address space. RTM1 saves the caller's registers and PSW, and performs the ABTERM request. RTM1 is invoked in home mode.		CABTERM
• **MEMTERM entry point** – Used to request scheduling of an address space termination. Since there are no specific lock requirements, the caller must provide a register save area. RTM1 performs a MEMTERM asynchronously with dependencies on locks and the dispatcher. Therefore, control may or may not return to the caller, depending on the lock status when the caller issued the request.		MEMTERM
• **ABTERM reentry** – Used when RTM1 scheduled itself as an SRB during a previous entry when the caller requested ABTERM with the ASID option. When entered at this entry point, RTM1 is operating as an SRB in the specified address sapce.		IEAVTRTX
4 MCH (machine check handler) and ACR (alternate CPU recovery) use this entry point when requesting hardware recording and hardware damage repair. The caller passes the address of a LOGREC buffer which contains all the information about the error. If RTM1 subsequently determines that software recovery is warranted, it will establish the appropriate software interface.		IEAVTRTN

IEAVTRT1 - ADDRESS TERMINATION ON A DAT ERROR



IEAVTRT1 – Address Space Termination on a DAT Error (Part 1 of 4)

IEAVTRT1 – Address Space Termination on a DAT Error (Part 2 of 4)

Extended Description	Module	Label
DATMEM is an SRB routine scheduled by the DATERR entry point in IEAVTRT1. DATMEM is scheduled when a DATERR occurs in an address space other than the home address space. DATMEM will terminate this address space.	IEAVTRT1	DATMEM

1 DATMEM establishes its own FRR (DATMEFRR) for error recovery.

2 DATMEM attempts to obtain an EED (extended error descriptor) to use as a savearea. If an EED cell is not obtained, DATMEM increases the RTCTEEDC counter by one. (RTCTEEDC contains a count of the number of times an EED GETCELL failed in RTM1. RTCTEEDC was set to zero during IPL.) DATMEM frees the SRB storage, deletes the FRR, and returns control to the SRB dispatcher.

3 If an EED cell is obtained, DATMEM copies the ASID, completion code, reason code, PSW1, and PSW2 from the SRB save area into the EED. DATMEM then issues a FREESRB command to free the storage for the SRB.

4 DATMEM establishes serialization by obtaining the DISP lock and the global intersect. DATMEM then calls the lock repair routine (IEAVELKR).

The interface to the lock repair routine is as follows:

- Register 0: The function code for the DAT error.
- Register 1: The system completion code in bits 8-19.
- Register 13: The address of a two-word parameter list:
  - 1st word – Contains zeroes
  - 2nd word – Contains the address of a four-fullword parameter list (WSACTRT1)

Extended Description	Module	Label

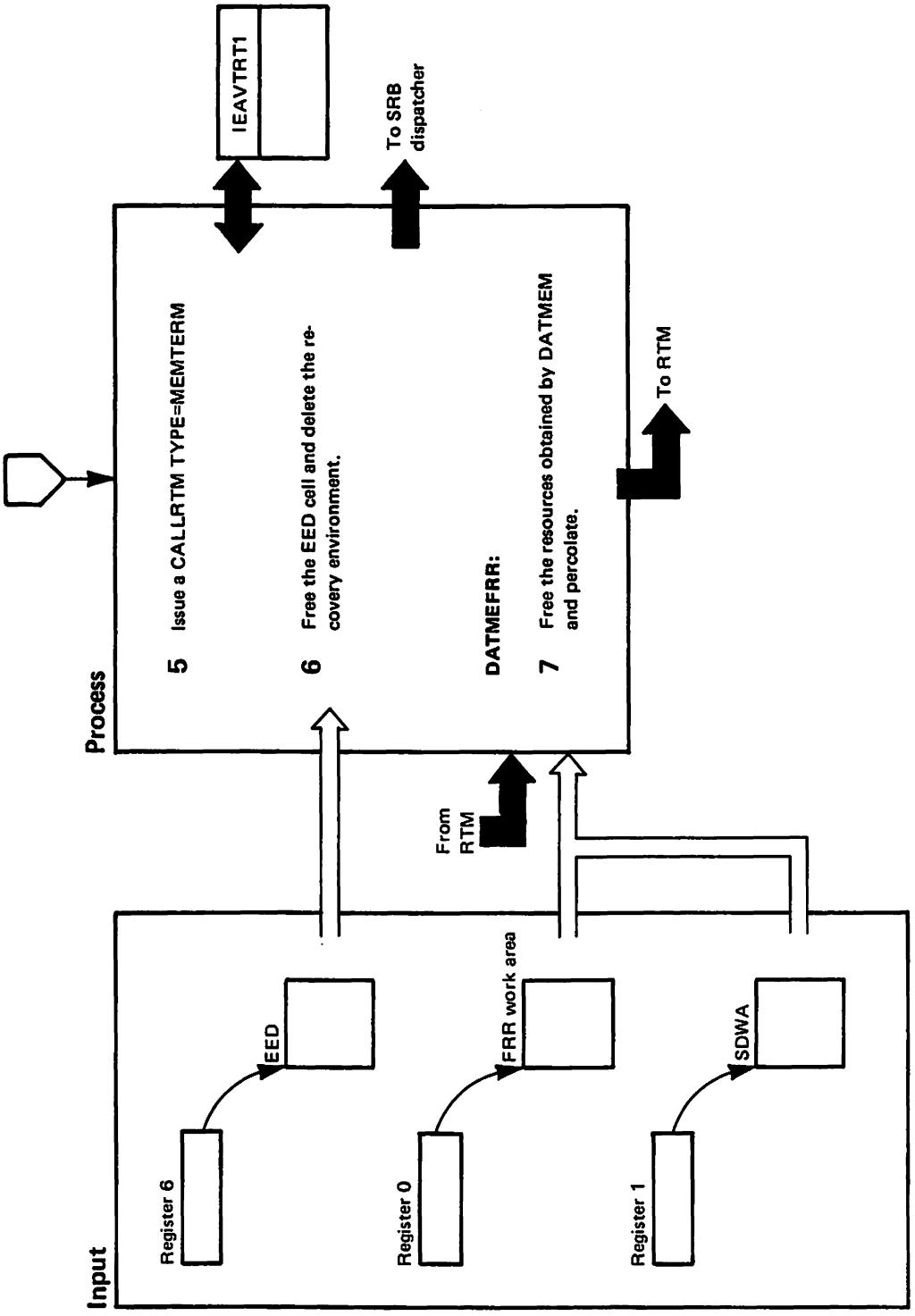
4 (continued)

The WSACTRT1 parameter list consists of:

- 1st word – The address of two contiguous words containing the completion code and reason code.
- 2nd word – The address of the failing PSW.
- 3rd word – The address of two contiguous halfwords containing the interrupt length code and interrupt code.
- 4th word – The ASID of the failing address space, right justified.

Register 14: DATMEM's return address.
Register 15: The entry point address for IEAVELKR.

When IEAVELKR returns control to DATMEM, DATMEM frees the DISP lock and the global intersect.

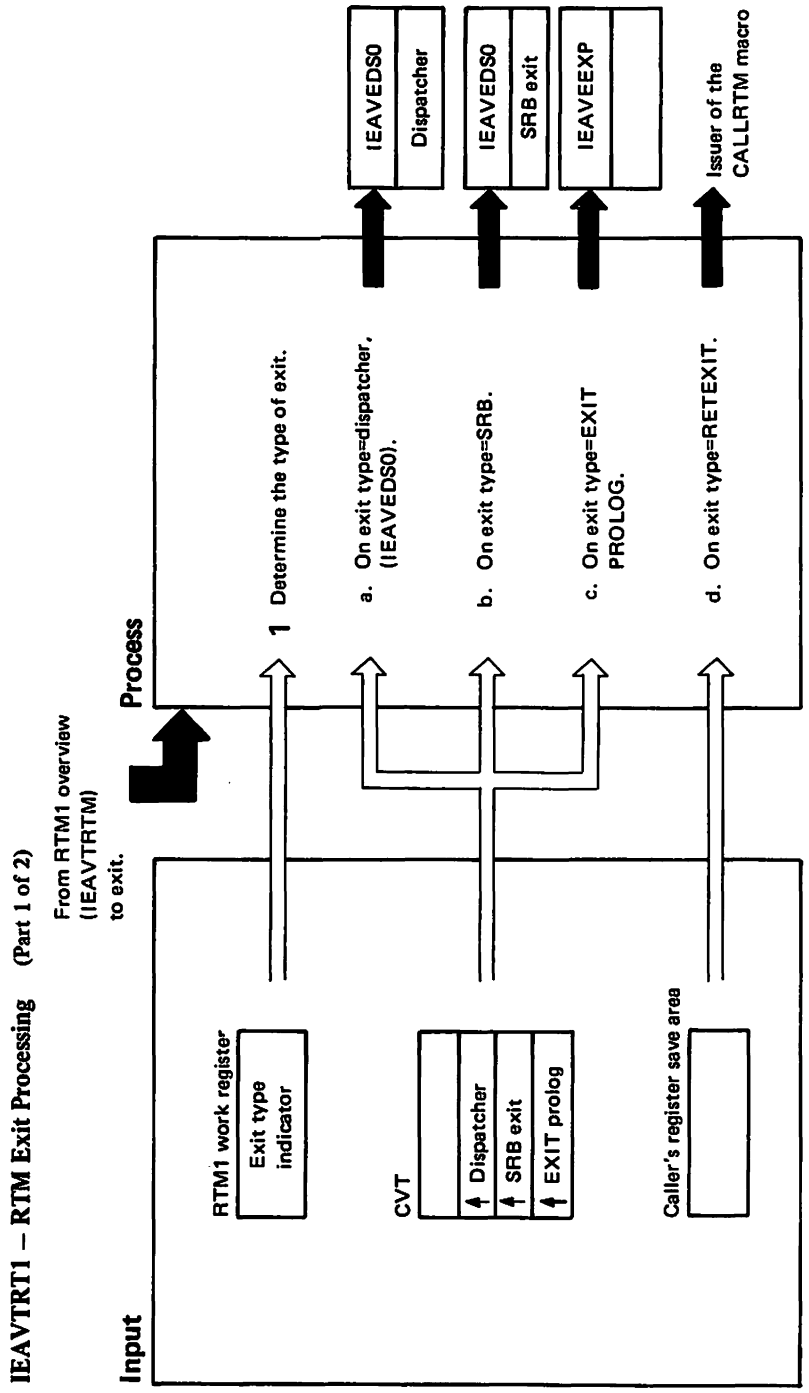


IEAVTRT1 — Address Space Termination on a DAT Error (Part 3 of 4)

IEAVTRT1 -- Address Space Termination on a DAT Error (Part 4 of 4)

Extended Description	Module	Label
5 IEAVTRT1 issues CALLRTM TYPE=MEMTERM to terminate the address space. The completion code is X'0D2'.		
6 DATMEM issues a FREECELL to free the EED cell and deletes the recovery environment by issuing a SETFRR. IEAVTRT1 returns control to the SRB dispatcher.		
7 In error cases, IEAVTRT1 passes control to DATMEFRR for recovery. DATMEFRR places diagnostic information in the SDWA. If DATMEM obtained an EED cell or the global intersect, it is freed. If DATMEM obtained the DISP lock, DATMEM requests that RTM free this lock. DATMEM issues a SETRP to request recording of the SDWA and percolation of the error. DATMEM returns control to RTM.	IEAVTRT1	DATMEFRR

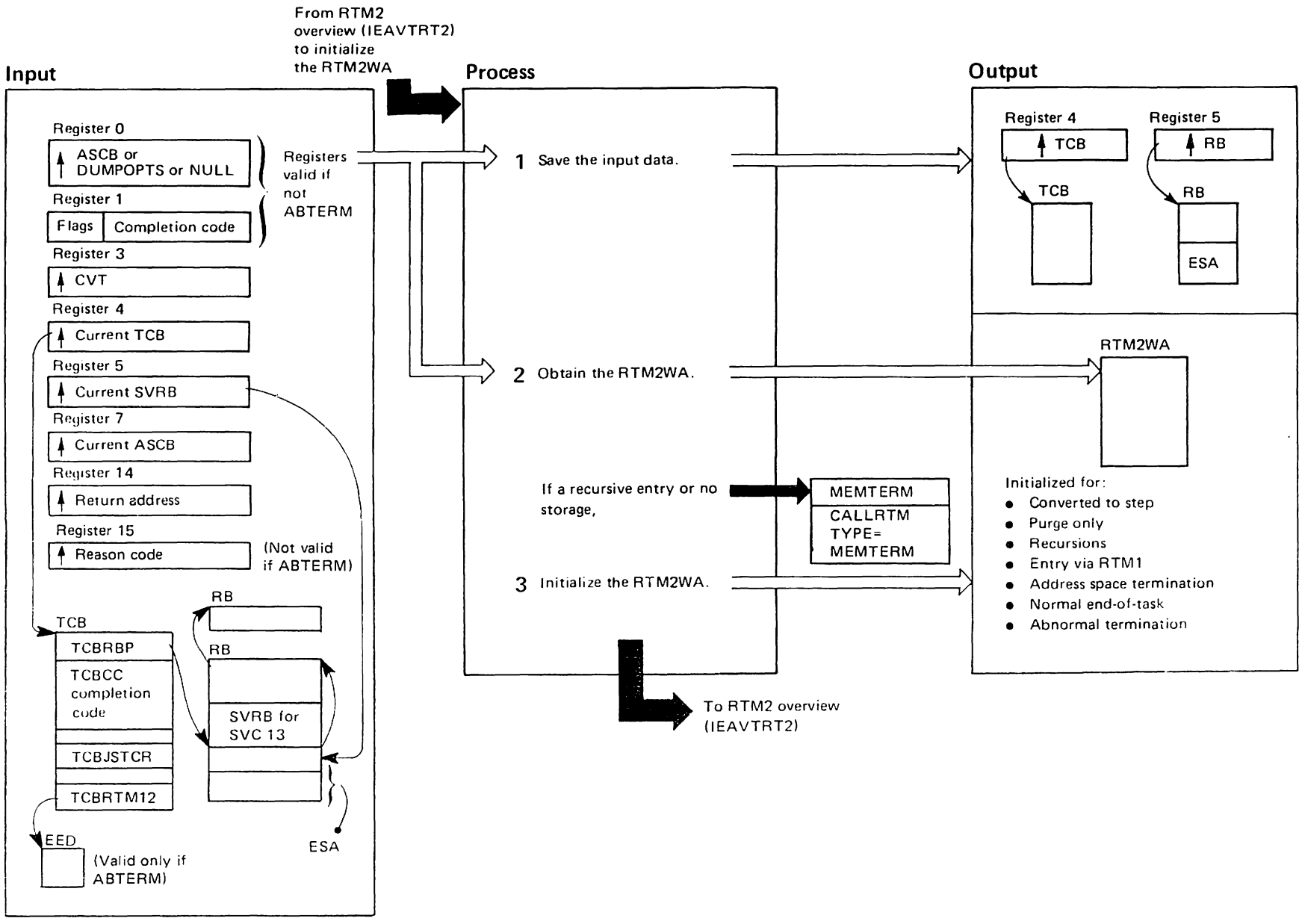
IEAVTRT1 - RTM1 EXIT PROCESSING



IEAVTRT1 – RTM1 Exit Processing (Part 2 of 2)

Extended Description	Module	Label
RTM1 routines exit from a common exit routine within module IEAVTRT1.		
1 RTM1 exit processing uses the exit type determined by the module IEAVTRTM to perform the appropriate exit procedure, as follows:	IEAVTRT1	IEAVTRTZ
a. When exiting to the dispatcher, IEAVTRT1 places the dispatcher's exit point (the CVTODS field of the CVT) in register 15. If the current stack is the normal stack, IEAVTRT1 turns off the EUT mode indicator (PSANSS). It then branches to the address in register 15.		RT1EXIT2
b. When an SRB is to get control, IEAVTRT1 puts the SRB exit point (CVTSRBRT field of the CVT) in register 15. If the current stack is the normal stack, IEAVTRT1 turns off the EUT mode indicator (PSANSS). It then branches to the address in register 15.		RT1EXIT4
c. When EXIT prolog is to get control, IEAVTRT1 places EXIT prolog's exit point (CVTEXPRO field of the CVT) in register 15. If the current stack is the normal stack, IEAVTRT1 turns off the EUT mode indicator (PSANSS). It then branches to the address in register 15.		RT1EXIT6
d. IEAVTRT1 reloads registers 0-15 from the register save area and executes a branch on register 14. For STERM, only register 14 (the return address) is restored.		RT1EXITE

IEAVTRT2 – RTM2 Initialization (Part 1 of 2)



IEAVTRT2 – RTM2 Initialization (Part 2 of 2)

Extended Description	Module	Label
RTM2 communicates between its various routines via the RTM2WA. RTM2 initialization processing creates and initializes the RTM2WA for subsequent use by the RTM2 routines. The RTM2WA contains the following types of information: • Address of the TCB, RB, CVT, ASCB, and SDWA • Registers and PSW at the time of the error, and flags indicating the system state for ABTERM and ABEND requests • Machine check information • DUMP options if any were passed • Address of any previous work area and indicators for recursive entries • Error ID Control goes from initialization to the RTM2 controller (represented by the M.O. diagram IEAVTRT2 – RTM2 Overview) to continue processing. Register 0 contains either the address of the ASCB representing the address space to be terminated if address space termination is requested or the address of the dump options if dump options were supplied and entry is not via the RTM1 ABTERM function. Register 1 contains the completion code and flags indicating the type of request and options if the entry is not via the RTM1 ABTERM function. If entry is via the RTM1 ABTERM function, the dump options, completion code, and type of request, are passed via the TCB fields.		
1 Initialization processing saves the input registers and TCB flags in the ESA. Those TCB fields set by RTM1 are cleared to prevent confusion in case of recursion. The TCB fields necessary for recursion tracking are set. IEAVTRT2 blocks asynchronous exits. If this is a recursive entry, IEAVTRT2 copies the recursion flags from the previous ESA.	IEAVTRT2	RT2INESA
2 If the ESACTS flag is on, this ABEND is on a jobstep task: RTM2 converted an ABEND to the step level. If so, the workarea required for the initial ABEND has been queued to this TCB and no new workarea should be acquired. If the flag is off, storage is acquired for an RTM2WA and RTM's copy of the SDWA.		RT2GETWA
If it is not possible to obtain storage (RC=4, from GETMAIN), initialization processing passes control to the critical error routine, which attempts to take an SVC dump and terminate this address space. This is done since no storage remains in the LSQA.		RT2CRERR RT2TMRY
3 Initialization processing places the critical error routine address in the RTM2WA (RTM2CTRA) and sets an initialization phase recursion indicator (ESAINREC) in the ESA. If this is not a purge-only entry, or an entry on a jobstep TCB, initialization processing also places the step conversion recursion handler address in the RTM2WA (RTM2STRA). The initialization processing routine initializes the RTM2WA, using data found originally in the input registers, the TCB, the RB queue, and, if the entry is from RTM1, the extended error descriptors (EEDS). Further initialization of RTM2WA occurs when IEAVTRT2 calls IEAVTR2A. IEAVTR2A obtains the six bytes of instruction stream that precede and the six bytes that follow the instruction counter (IC) of the failing PSW and copies them in the RTM2FAIN field. (See M.O. diagram IEAVTR2A – RTM2 Failing Instruction Processor.)		RT2INWA RT2INCNV RT2INCM RT2INEOT RT2INABD RT2INRT1 RT2INMT RT2CYEED RT2MODE RT2INPG RT2INRCR IEAVTR2A
Initialization processing terminates the address space if this is a recursive entry and if the ESAINREC flag is on. If control returns normally from initialization, initialization processing resets the ESAINREC flag. Initialization processing issues the SNAPTRC macro to take a snapshot of the system trace table.		

IEAVTRT2 – Recursion Processor 1 (Part 1 of 2)

Extended Description

- An intermediate level of recursion handling is established which causes a recursion on a non-jobstep TCB to abnormally terminate the jobstep and reinitiate RTM2 processing at that level. This is preferable to the critical recursion handling because it may permit a larger number of TERM exits and resource managers to get control. If the error persists, the critical recursion handler will get control. However, if the error was due to an asynchronous event that does not recur, RTM2 processing should complete normally at the jobstep level.
- For critical RTM2 processing and for situations for which no recovery is possible, a fourth recursion routine exists which will request an address space termination. This routine is also used when all other recursion routines have been exhausted. During the time that no RTM2WA exists in initialization and exit processing, the recursion control is managed using the ESA, and the critical recursion routine is always invoked on an error.

On recursion entries, these recursion handling routines make no attempt to determine the cause of the error.

On recursive entries, a purge back of SVRBs and RTM2WAs is not done, except for recursion during task recovery pre-exit processing. This permits full information to appear in a dump and also provides some loop control as a routine must specifically establish a recursion routine on this error for it to be applicable on the next. An RB purge is done for task recovery to avoid passing error data for errors suffered by routines used by task recovery to the recovery exits.

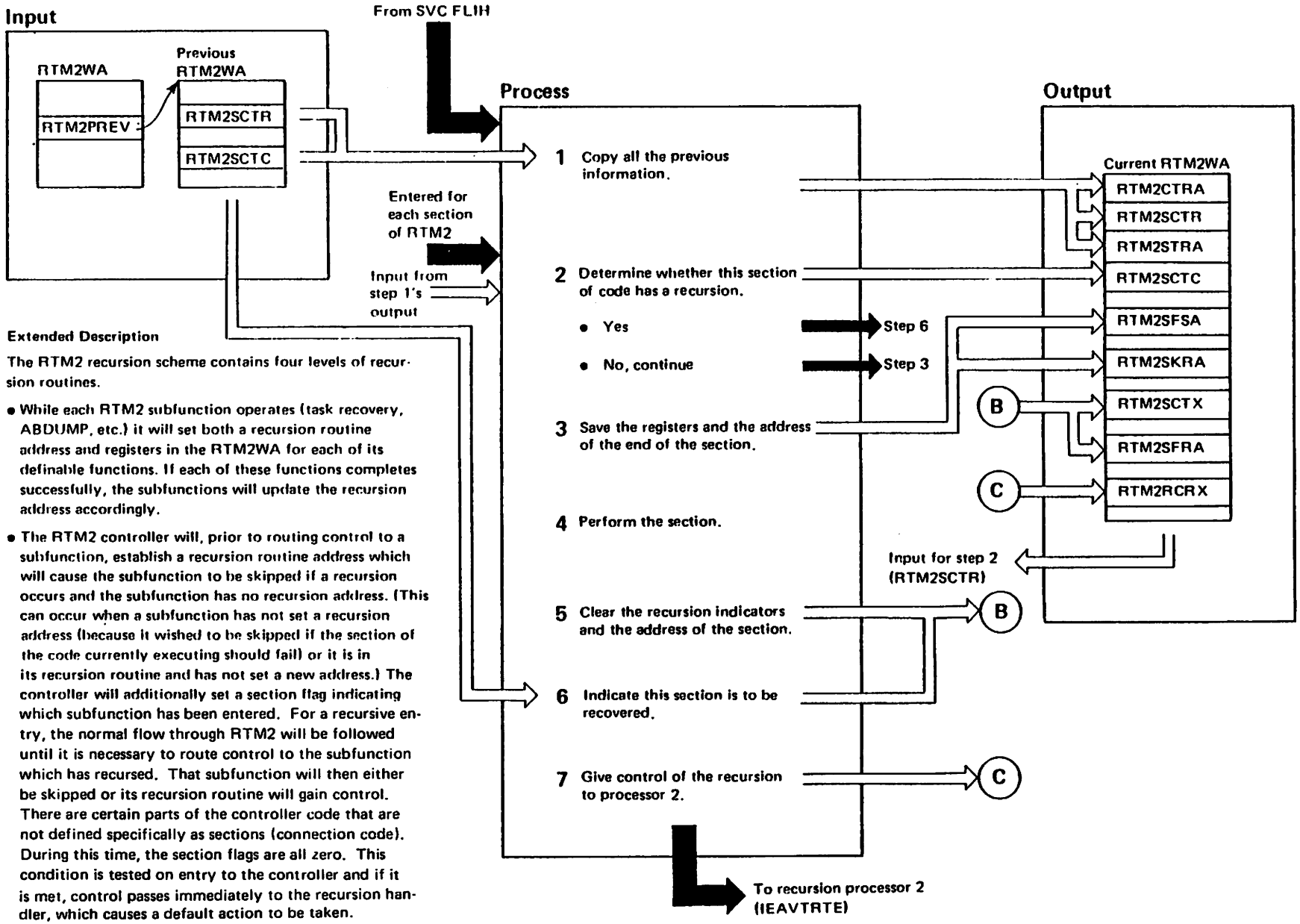
RTM2 uses three sets of flags to maintain control during recursion. RTM2 sets the RTM2SCTC flags as it enters each section and sets them to zero when the section is complete. When one of these flags is set, there is generally a skip address which will cause the section to be bypassed if it does suffer an error.

The RTM2SCTR flags contain the history of all the sections that have suffered a recursion which has not yet been recovered. The controller tests this flag prior to setting the RTM2SCTC flag for a given section and if it is on the recursion exit is taken to give the recursion address control. These flags are necessary as RTM2 processing follows a different order of paths based on the type of error encountered.

The RTM2SCTX flags indicate to the recursion exit handler the section whose recursion address must be given control. When the controller finds the RTM2SCTR flag on for the section it is about to execute, it sets the corresponding RTM2SCTX flag and passes control to the exit handler. The exit handler will then use the RTM2SCTX flag to locate the appropriate RTM2WA and recursion address for this section.

Extended Description	Module	Label
1 The recursion processor 1 first copies any previous status information that applies for all failures, and combines the recursion information from the most recent failing section of code with all previous failed sections of code. (See the M.O. diagram IEAVTRT2 – RTM2 Initialization for a description of the recursion indicators set for critical error routine address and step conversion recursion handler address.) This provides a complete set of recovery information.	IEAVTRT2	RT2INRCR
2 Each section of code performs the operation described in steps 2-7. The section checks the RTM2SCTR field of the RTM2WA for a recursion indication. If this indicator shows that this section of code failed and has not been recovered, it cannot be reentered. Control goes to step 6. Otherwise, control continues to step 3.	IEAVTRTC	
3 The section of code sets an indicator, in the RTM2SCTC field in the RTM2WA, that shows which section has control. If a recursion should occur, the position of this indicator (a bit) in the field (1 word) will locate the section of code that failed.	IEAVTRTC IEAVTRTE	
The section of code saves the registers, in the RTM2SFSA, that will be needed if the section fails, and saves the address of the code following the section in the RTM2SKRA. Using this information, the section of code can be skipped if necessary.		
4 Each section of code can be further divided into subsections, by using flags unique to the section. If a section can handle certain recursions on its own, it sets another recursion address in the RTM2TRRA field and saves the registers in the RTM2RREG field. This permits, for example, a failing caller ESTAE exit to be skipped, without causing all of task recovery to be skipped.		
5 The section clears the section indicator in the RTM2SCTC field, and the address in the RTM2SKRA field.		
6 After determining that this section has failed (in step 2), the recursion processor 1 sets an indicator in the RTM2SCTX field that indicates the section of code that failed.		
7 The recursion processor 1 sets the RTM2RCRX field. When this field is set, the recursion processor 2 will receive control to process the recursion.		

IEAVTRT2 – Recursion Processor 1 (Part 2 of 2)



Extended Description

The RTM2 recursion scheme contains four levels of recursion routines.

- While each RTM2 subfunction operates (task recovery, ABDUMP, etc.) it will set both a recursion routine address and registers in the RTM2WA for each of its definable functions. If each of these functions completes successfully, the subfunctions will update the recursion address accordingly.
- The RTM2 controller will, prior to routing control to a subfunction, establish a recursion routine address which will cause the subfunction to be skipped if a recursion occurs and the subfunction has no recursion address. (This can occur when a subfunction has not set a recursion address (because it wished to be skipped if the section of the code currently executing should fail) or it is in its recursion routine and has not set a new address.) The controller will additionally set a section flag indicating which subfunction has been entered. For a recursive entry, the normal flow through RTM2 will be followed until it is necessary to route control to the subfunction which has recursed. That subfunction will then either be skipped or its recursion routine will gain control. There are certain parts of the controller code that are not defined specifically as sections (connection code). During this time, the section flags are all zero. This condition is tested on entry to the controller and if it is met, control passes immediately to the recursion handler, which causes a default action to be taken.

IEAVTR1A - MODULE DESCRIPTION

DESCRIPTIVE NAME: RTM1 Failing Instruction Processor

FUNCTION:
This module obtains the six bytes of storage that precede and the six bytes that follow the instruction counter (IC) in the failing program status word (PSW) and copies the failing instruction stream into either the SDWAFAIN field or the EEDFAIN field depending on the caller.

ENTRY POINT: IEAVTR1A

PURPOSE:
Puts the failing instruction stream into the SDWAFAIN field of the system diagnostic work area (SDWA).

LINKAGE: BALR

CALLERS: IEAVTR1I

INPUT: The FAINPL

OUTPUT:
The failing instruction stream is in the SDWAFAIN in the interrupted/error SDWA.

EXIT NORMAL: Returns to IEAVTR1I

EXIT ERROR: Terminates

ENTRY POINT: IEAVTR1B

PURPOSE:
Puts the failing instruction stream in the EEDFAIN field of the extended error descriptor block (EED).

LINKAGE: BALR

CALLERS: IEAVTRS0

INPUT:
Registers in use by IEAVTRS0.
Register 1 - Address of the FAINPL
Register 4 - Address of the RT1TRACK
The FAINPSW contains the failing PSW.

OUTPUT: The failing instruction stream is in the EEDFAIN.

EXIT NORMAL: Returns to IEAVTRS0

EXIT ERROR: Terminates

ENTRY POINT: R1AFRR

PURPOSE:
Recovers from errors encountered during IEAVTR1A's processing.

LINKAGE: LPSW

CALLERS: RTM1

INPUT:
Register 0 - Address of a 200 byte FRR work area
Register 1 - Address of the SDWA associated with RTM's FRR stack.

Register 14 - Return address to RTM1
Register 15 - R1AFRR entry point address

OUTPUT:
The SDWARCDE field of the SDWA contains a return code indicating either retry or continue with termination.

EXIT NORMAL: Returns to RTM1

EXIT ERROR: No exit error conditions

EXTERNAL REFERENCES:

ROUTINES:
IEAVTRV3 - Converts the real address in the PSWIC to a virtual address.

DATA AREAS:
- FAINPL-the failing instruction parameter list structure follows:

Field	Decimal Offset	Length	Description
FAINPSW	0	8	Failing PSW
FAINTADR	8	4	The target address for the failing instruction stream
FAINID	12	1	Identity of the caller
FAINLPN	13	1	Logical phase number
FAINLPID	14	1	Logical phase identification
FAINAUDT	15	1	Switch to current FRR stack
FAINISTK	16	4	Address of error stack
FAINRETY	20	4	IEAVTR1A's processing variables

CONTROL BLOCKS:

Common name	Macro ID	Usage	Function
CVT	CVT	read	Obtains IEAVTRV3's address to convert addresses from real to virtual.
EED	IHART1W	read and write	Gets and frees one cell from the EED pool that is used as a work area to contain the FAINPL. Adds the failing instruction stream to the EED, which is passed as a parameter to IEAVTR1B.
FRRS	IHAFRRS	read and write	Adds and deletes one FRR from RTM's FRR stack.
PSA	IHAPSA	read and write	Obtains and changes the address of the current FRR stack.
RT1W	IHART1W	read and write	Sets and resets the LPN and LPID fields.
SDWA	IHASDWA	read and write	Adds the failing instruction stream to the SDWA, which is passed as a parameter to IEAVTR1A.

IEAVTR1A - MODULE DESCRIPTION (Continued)

TABLES: No tables used.

SERIALIZATION: IEAVTR1A does not obtain any locks.

IEAVTR1A - MODULE OPERATION

IEAVTR1A receives control to obtain the failing instruction stream around the instruction counter (IC) of the failing PSW and to put the stream into either the SDWAFAIN or the EEDFAIN field depending on the caller.

Entry point IEAVTR1A receives control when IEAVTR1I is the caller to:

- Verify that the SDWA already contains the failing instruction stream.
- Determine the availability of the PSW at the time of the error.
- Initialize the FAINPL.

Entry point IEAVTR1B receives control when IEAVTRS0 is the caller to save the LPN (logical phase number), and LPID (logical phase identification) in the FAINPL. IEAVTRS0 builds the FAINPL prior to calling IEAVTR1A.

This module does the following common processing:

- Establishes the recovery environment via an FRR (functional recovery routine).
- Verifies that the PSWIC has a virtual address.
- Reads the failing instruction stream in the user's PSW key and copies it to the SDWA or EED, normally in one move. If the instruction stream overlaps a page boundary, copies it in two moves.
  - If the failing instruction stream is not accessible, RTM1 requests a retry via its FRR.
- Performs RTM1's failing instruction cleanup processing.
- Returns to its caller.

RECOVERY OPERATION:

The FRR requests a retry:

- At location RETRY1 - if an error occurs while IEAVTR1A is copying in two moves the first segment of the failing instruction stream. The SDWAFAIN field or the EEDFAIN field might be partially filled-in.
- At location RETRY2 - if an error occurs while IEAVTR1A is copying in two moves the second segment of the failing instruction stream or while copying in one move the complete failing instruction stream. The SDWAFAIN field or the EEDFAIN field contains hexadecimal zeross.
  - if an error occurs while IEAVTR1A is executing (other than copying the failing instruction stream).

If a retry is not allowed (the SDWACLUP bit is on), the FRR requests percolation.

Note: If an error occurs during recovery processing, RTM abort processing will get control.

IEAVTR1A - MODULE OPERATION (Continued)

The following chart summarizes IEAVTR1A's recovery processing:

Condition	Retry	Location	Error Recorded on SYS1.LOGREC
Failure to copy the first segment of the failing instruction stream.	Yes	RETRY1	No
Failure to copy the second segment of the failing instruction stream.	Yes	RETRY2	No
IEAVTR1A execution failure.	Yes	RETRY2	Yes
Failure to copy the failing instruction stream in one move.	Yes	RETRY2	No
SDWACLUP bit is on.	(Percolation occurs)	N/A	Yes

IEAVTR1A - DIAGNOSTIC AIDS

ENTRY POINT NAMES: IEAVTR1A
IEAVTR1B
R1AFRR

MESSAGES: None

ABEND CODES: None

WAIT STATE CODES: None

RETURN CODES:

ENTRY POINT IEAVTR1A: None

ENTRY POINT IEAVTR1B: None

ENTRY POINT R1AFRR:

EXIT NORMAL:

0 - in the SDWARCDE field indicates termination
4 - in the SDWARCDE field indicates retry

REGISTER CONTENTS ON ENTRY:

ENTRY POINT IEAVTR1A:

Register 1	- FAINPL address
Register 13	- Register save area address
Register 14	- Return address
Register 15	- Entry point address

ENTRY POINT IEAVTR1B:

Register 1	- FAINPL address
Register 4	- RT1TRACK address
Register 13	- Register save area address
Register 14	- Return address
Register 15	- Entry point address

ENTRY POINT R1AFRR:

Register 0	- 200 byte FRR work area address
Register 1	- SDWA address
Registers 2-13	- Irrelevant
Register 14	- Return address
Register 15	- Entry point address

REGISTER CONTENTS ON EXIT:

ENTRY POINT IEAVTR1A:

Registers 0-15 - Same as on entry

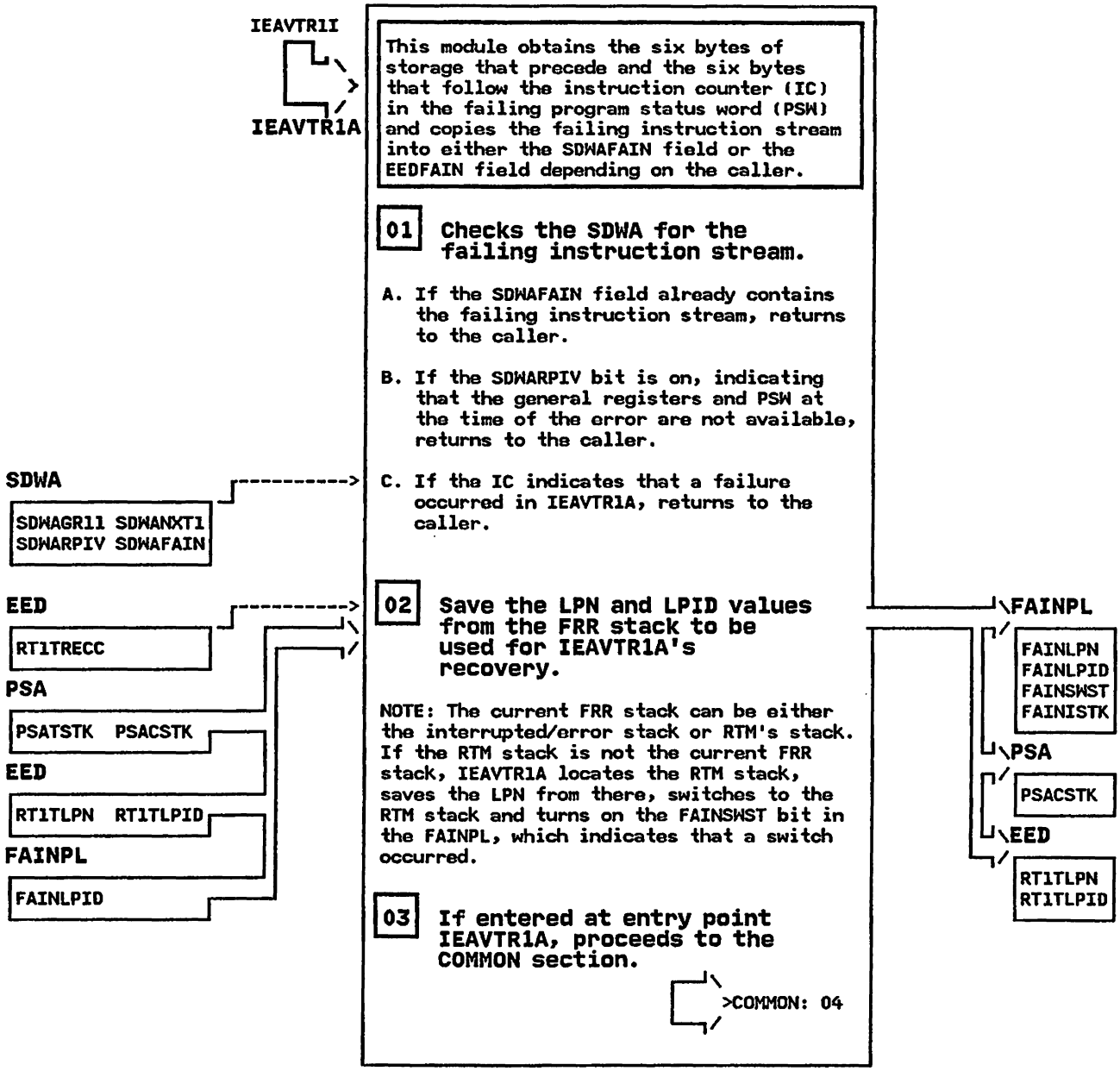
ENTRY POINT IEAVTR1B:

Registers 0-15 - Same as on entry

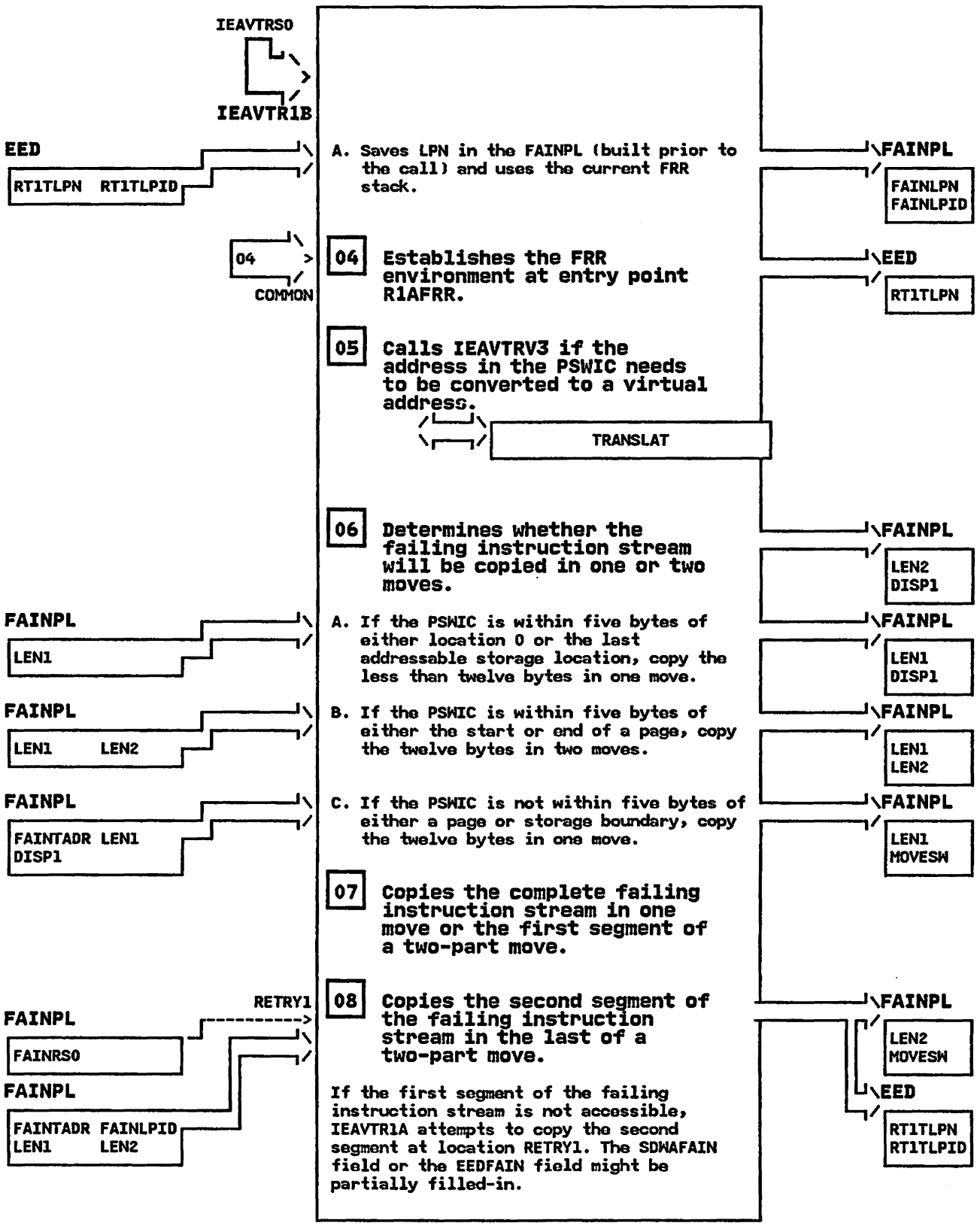
IEAVTR1A - DIAGNOSTIC AIDS (Continued)

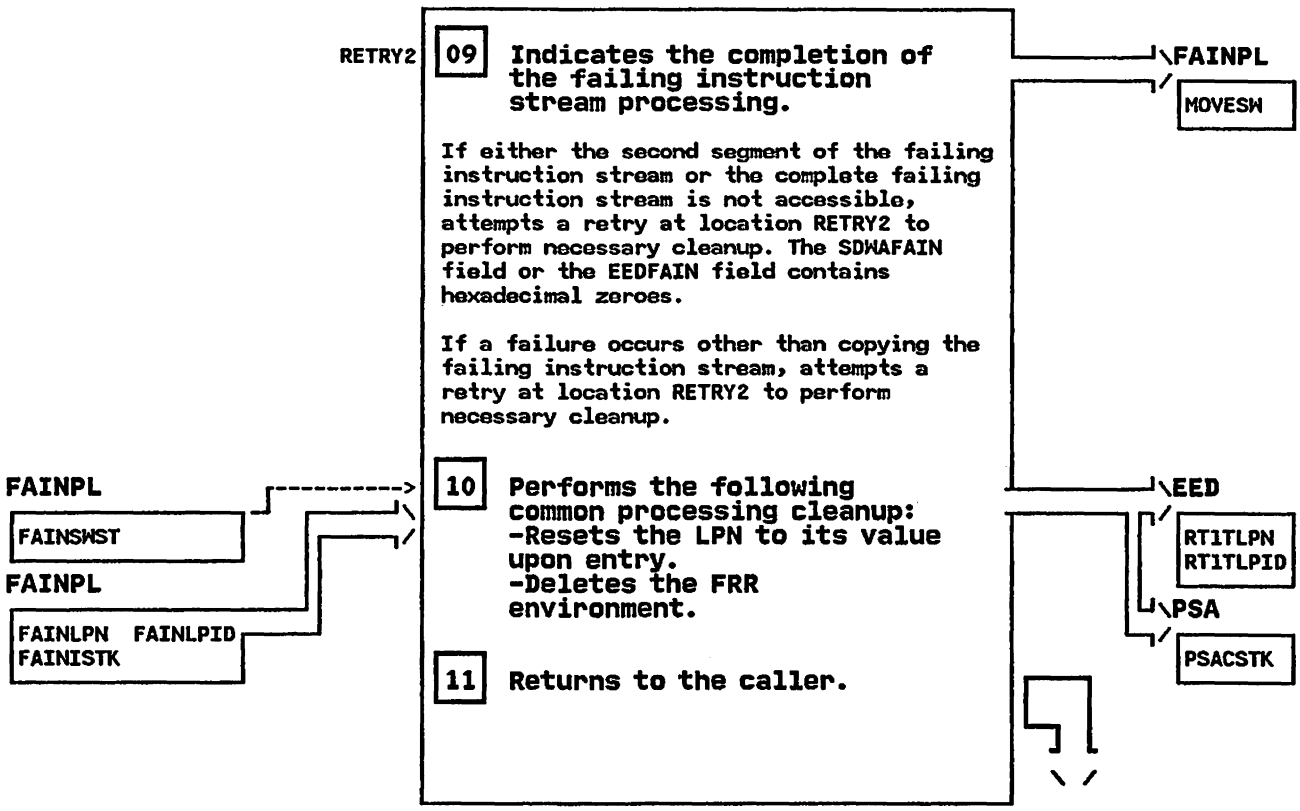
ENTRY POINT R1AFRR:

Registers 0-13 - Unpredictable
Register 14 - Return address to RTM1
Register 15 - Unpredictable



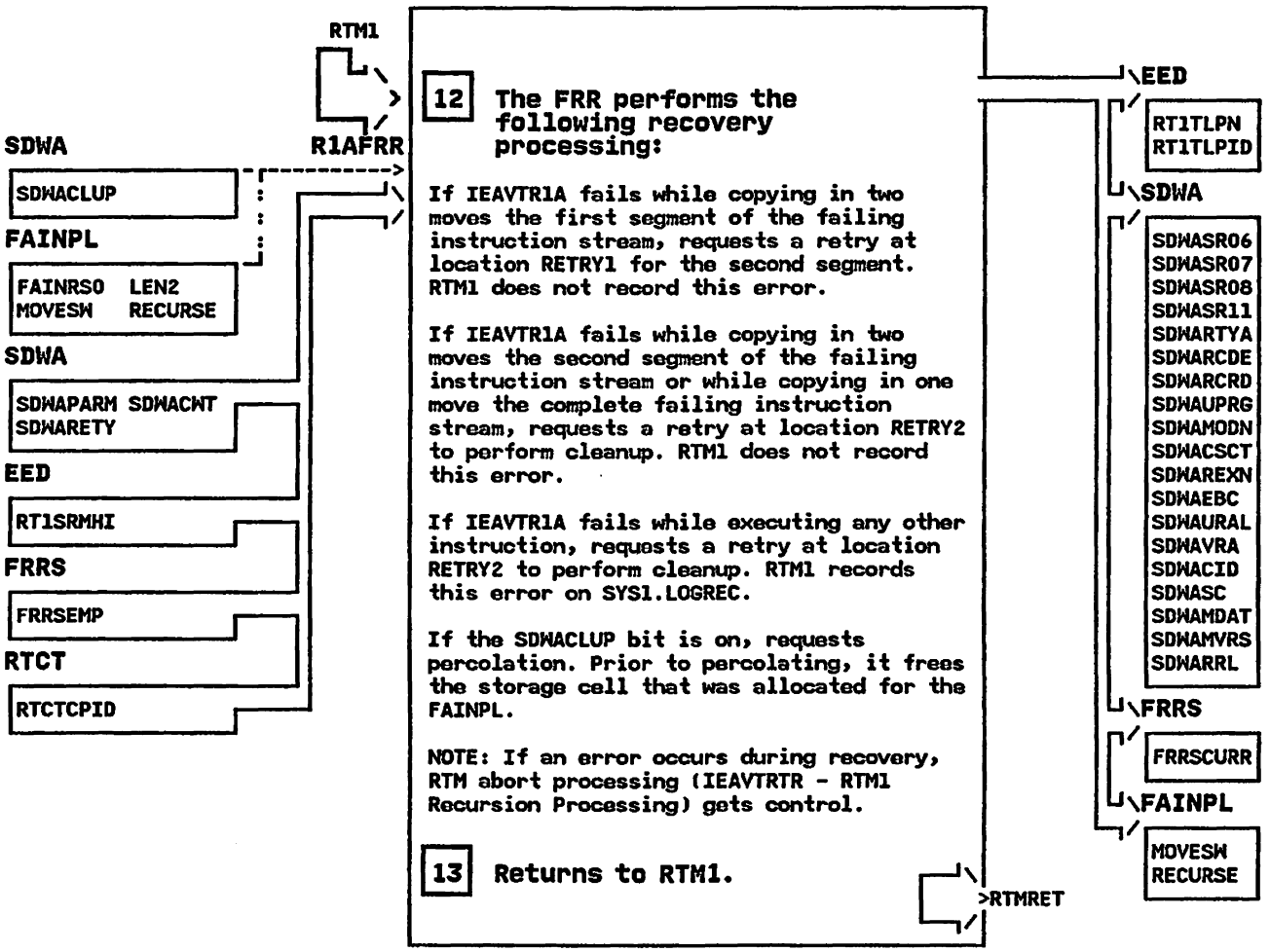
IEAVTR1A - RTM1 Failing Instruction Processor STEP 03A





IEAVTR1A - RTM1 Failing Instruction Processor

STEP 12



IEAVTR1C - MODULE DESCRIPTION

DESCRIPTIVE NAME: Service Module for IEAVTRTS

FUNCTION:

This module performs two main functions.

Entry point IEAVTR1C performs the first function, which is performed once at the beginning of each error entry into RTM. IEAVTR1C obtains an SDWA based on the current system state, initializes that SDWA with error related information, and prepares the FRR stack and RTM work area for the processing of this error. The error might be in the mainline, resulting in initial entry into RTM or it might be an error in an FRR, causing recursive entry into RTM.

Entry point IEAVTR1D performs the second function, which is performed whenever an FRR has returned from its processing. IEAVTR1D prepares the SDWA, the FRR stack and RTM1 work area for percolating to the next FRR, for retry or for resume processing.

ENTRY POINT: IEAVTR1C

PURPOSE:
Obtains and initializes an SDWA, prepares the FRR stack and initializes the RTM environment whenever an error has occurred involving system routines. This might be an error in the mainline, causing initial entry into RTM or it might be an error in an FRR, causing recursive entry into RTM.

LINKAGE: BALR

CALLERS: IEAVTRTS

INPUT: The RTM1 work area

OUTPUT: An initialized SDWA and FRR stack

EXIT NORMAL: Returns to IEAVTRTS.

EXIT ERROR: There are no exit error conditions.

ENTRY POINT: IEAVTR1D

PURPOSE:
Processes the return of an FRR, copies FRR information from the SDWA to the RTM1WA, determines the correct RTM processing to take depending on the environmental conditions, and performs that processing: retry, resume, or percolation.

LINKAGE: BALR

CALLERS: IEAVTRTS and IEAVTR1F

INPUT: The RTM1 work area

OUTPUT: An updated FRR stack, SDWA and RTM1 work area

EXIT NORMAL: Returns to IEAVTRTS or IEAVTR1F.

EXIT ERROR: There are no exit error conditions.

EXTERNAL REFERENCES:

IEAVTR1C - MODULE DESCRIPTION (Continued)

ROUTINES:
- IEAVTR1I - RTM1 SDWA Initialization Module via CALL
- IEAVTR1R - RTM1 RECORD Interface Module via CALL
- IEAVTR1S - RTM1 SDWA Allocation Module via CALL
- IEAVTSSX - SLIP Space Switch Handler via CALL

CONTROL BLOCKS:

Common name	Macro ID	Usage	Function
ASCB	IHAASCB	read	Obtains the ASID of the locally locked address space.
CVT	CVT	write	Clears the CVT restart word.
FRRS	IHAFRRS	read and writes	Obtains various FRR status information and used by the SETFRR expansion.
LCCA	IHALCCA	read	Obtains address of CPU work save area.
PSA	IHAPSA	read	Obtains addresses of various FRR stacks, the ASCB and LCCA. Also used by SETFRR expansion.
RT1W	IHART1W	read and write	Obtains and sets RTM control information.
SDWA	IHASDWA	read and write	Obtains control information and sets FRR status indicators.
SVT	IHASVT	read and write	Obtains information for the INTSECT macro.

TABLES: No tables used.

IEAVTR1C - MODULE OPERATION

Entry point IEAVTR1C receives control once for each entry into RTM1 to process FRRs.

For an error encountered in mainline processing, IEAVTR1C:

- Prepares the normal FRR stack for processing. The RTM1 control information in each entry must be cleared in preparation for error processing.
- Calls IEAVTR1S to obtain an SDWA.
- Calls IEAVTR1I to initialize the SDWA with error related information.

For an error encountered in an FRR's processing, IEAVTR1C:

- If there are nested FRRs to be processed, copies the current error information (SDWA, selected RTM1 work area information and 200 byte FRR work area) into a checkpoint area. Nested FRRs might be allowed to retry or resume if the information can be copied to the checkpoint area. If the information can not be copied, the nested FRRs will not be allowed to retry or resume. IEAVTR1C writes the SDWA, which represents the previous error, to SYS1.LOGREC and releases the locks as required. In either case, the SDWA is initialized with information about the new error.
- If there are no nested FRRs to be processed, writes the SDWA representing the previous error to SYS1.LOGREC, releases the locks as required, and initializes the SDWA with information about the new error.

Entry point IEAVTR1D receives control to perform post FRR processing whenever an FRR completes its processing and returns to RTM or whenever there is an FRR on the stack that must be removed in preparation for routing to the next FRR. IEAVTR1D performs the following processing:

- Updates the RTM1 work area lock information to reflect locks that have been added by the FRR, locks that have been released by the FRR, and locks that have been requested to be released by the FRR via the SDWA. This information is important in making correct decisions during IEAVTR1D's subsequent processing.
- Examines the SDWA, the locks currently held, the super bits currently set and other system information to determine the type of action to be performed -- percolate, retry or resume. An indication of the type of action to be performed is saved in the RTM1 work area for later use.
- Does much of the actual processing required to process the percolation, retry or resume request. IEAVTR1O and IEAVTRTM will complete this processing.

RECOVERY OPERATION:

RTM1 default recovery processing protects IEAVTR1C's processing against errors. IEAVTRTR performs default recovery processing.

IEAVTR1C - DIAGNOSTIC AIDS

ENTRY POINT NAMES: IEAVTR1C
IEAVTR1D

MESSAGES: None

ABEND CODES: None

WAIT STATE CODES: None

RETURN CODES: None

REGISTER CONTENTS ON ENTRY:

ENTRY POINT IEAVTR1C:

Registers 0 - 12 - Irrelevant
Register 13 - Address of a standard register save area
Register 14 - The return address
Register 15 - The entry point address

ENTRY POINT IEAVTR1D:

Registers 0 - 12 - Irrelevant
Register 13 - Address of a standard register save area
Register 14 - The return address
Register 15 - The entry point address

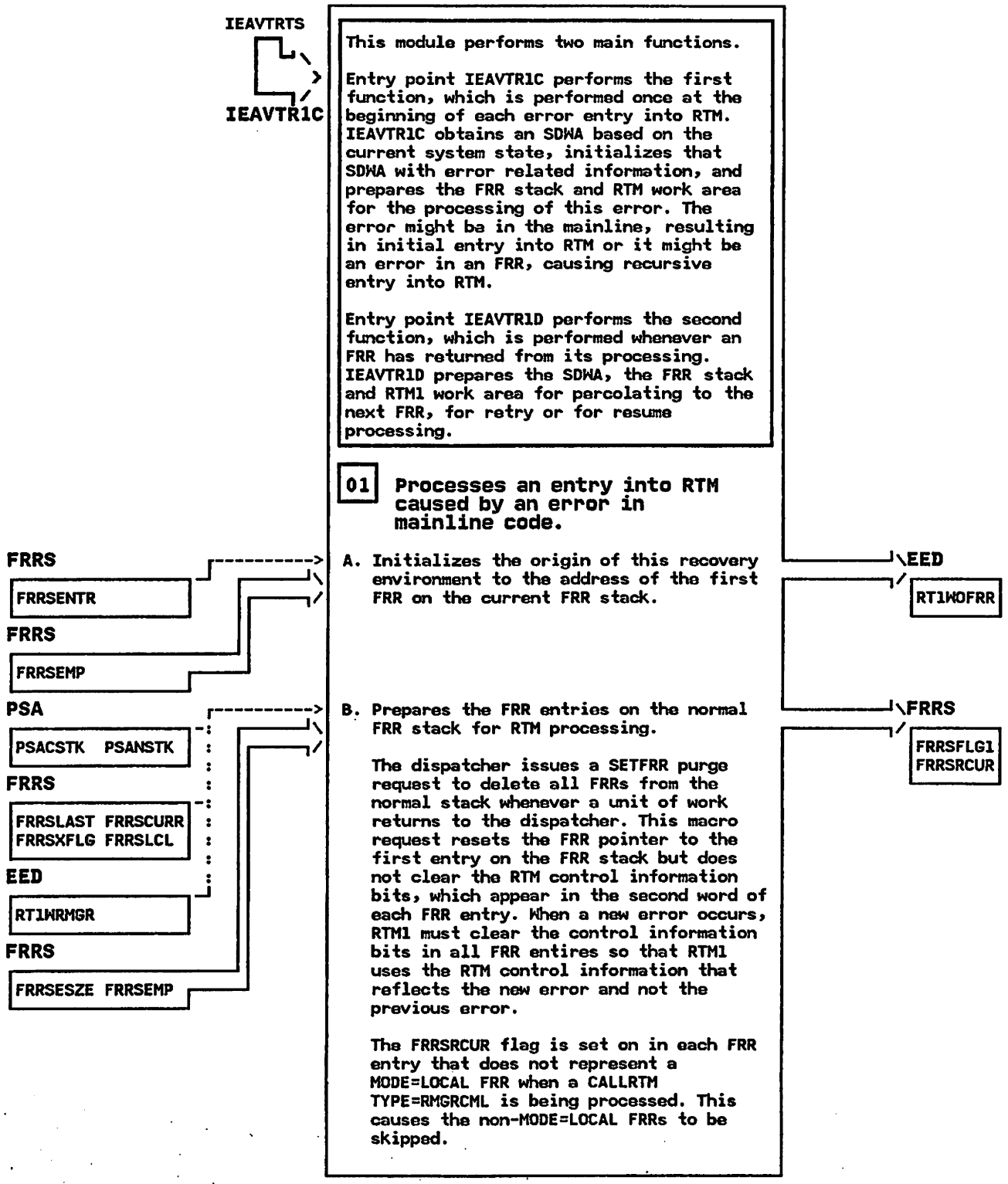
REGISTER CONTENTS ON EXIT:

ENTRY POINT IEAVTR1C:

Registers 0 - 15 - Same as on entry

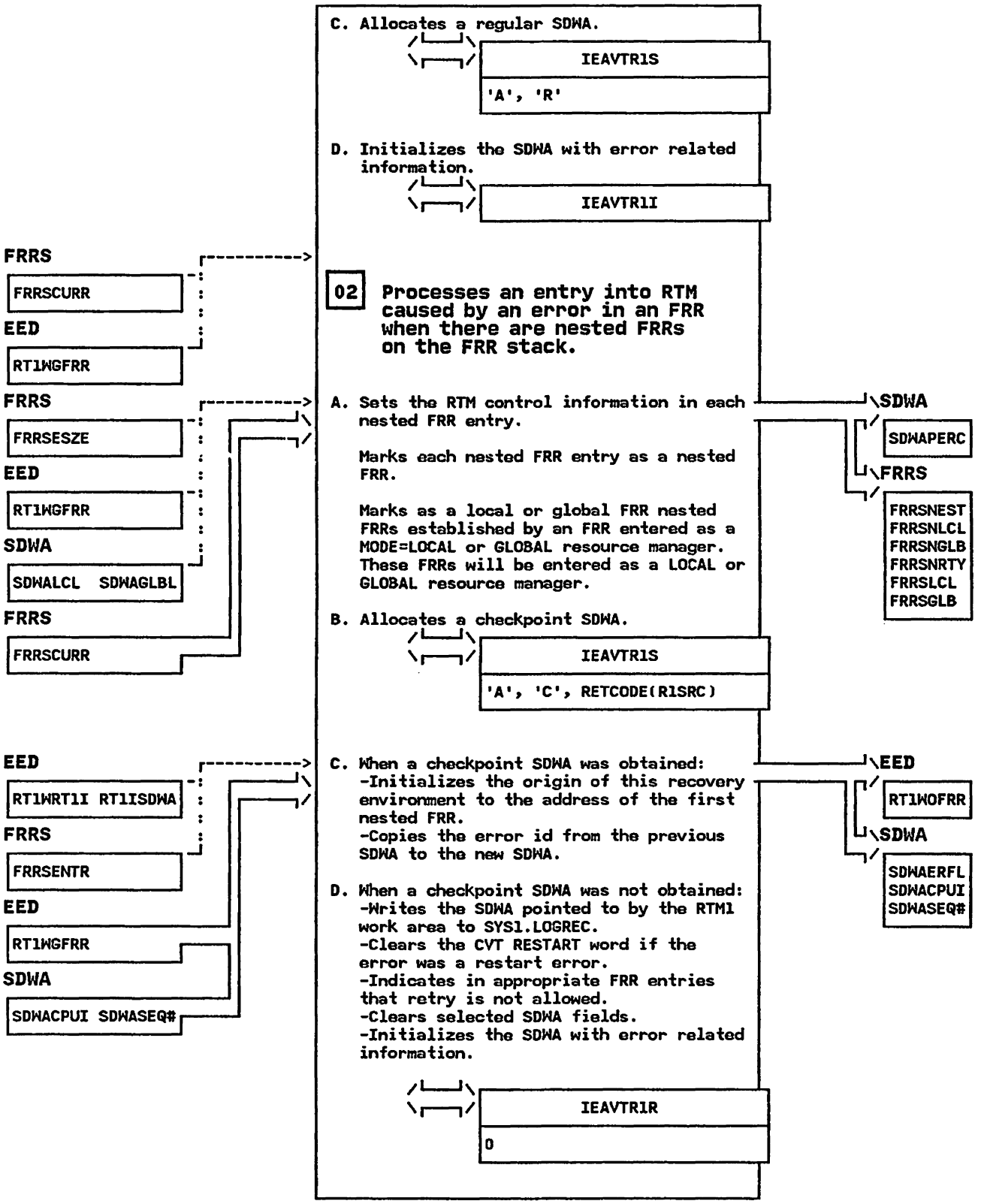
ENTRY POINT IEAVTR1D:

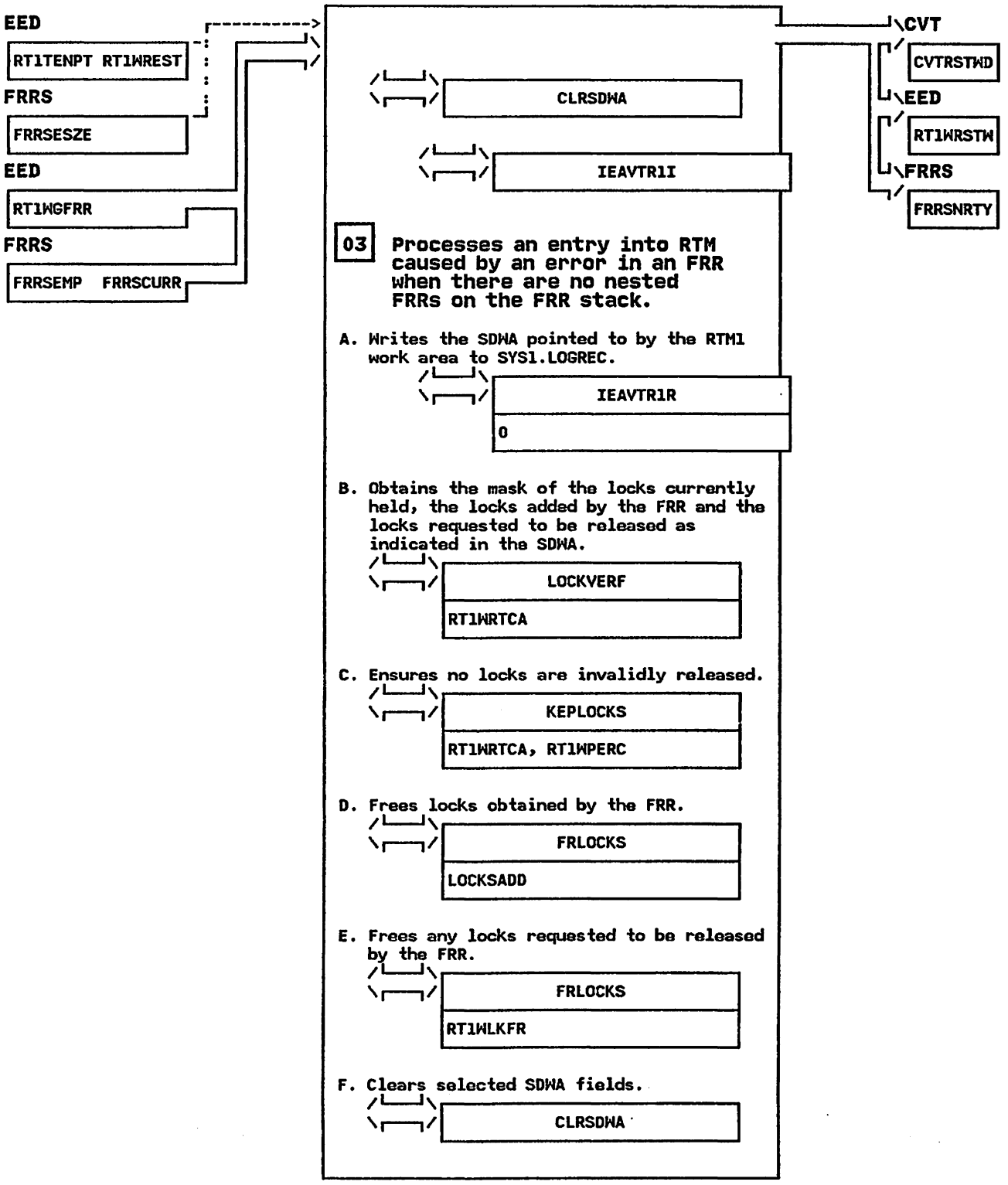
Registers 0 - 15 - Same as on entry



IEAVTR1C - Service Module for IEAVTRTS

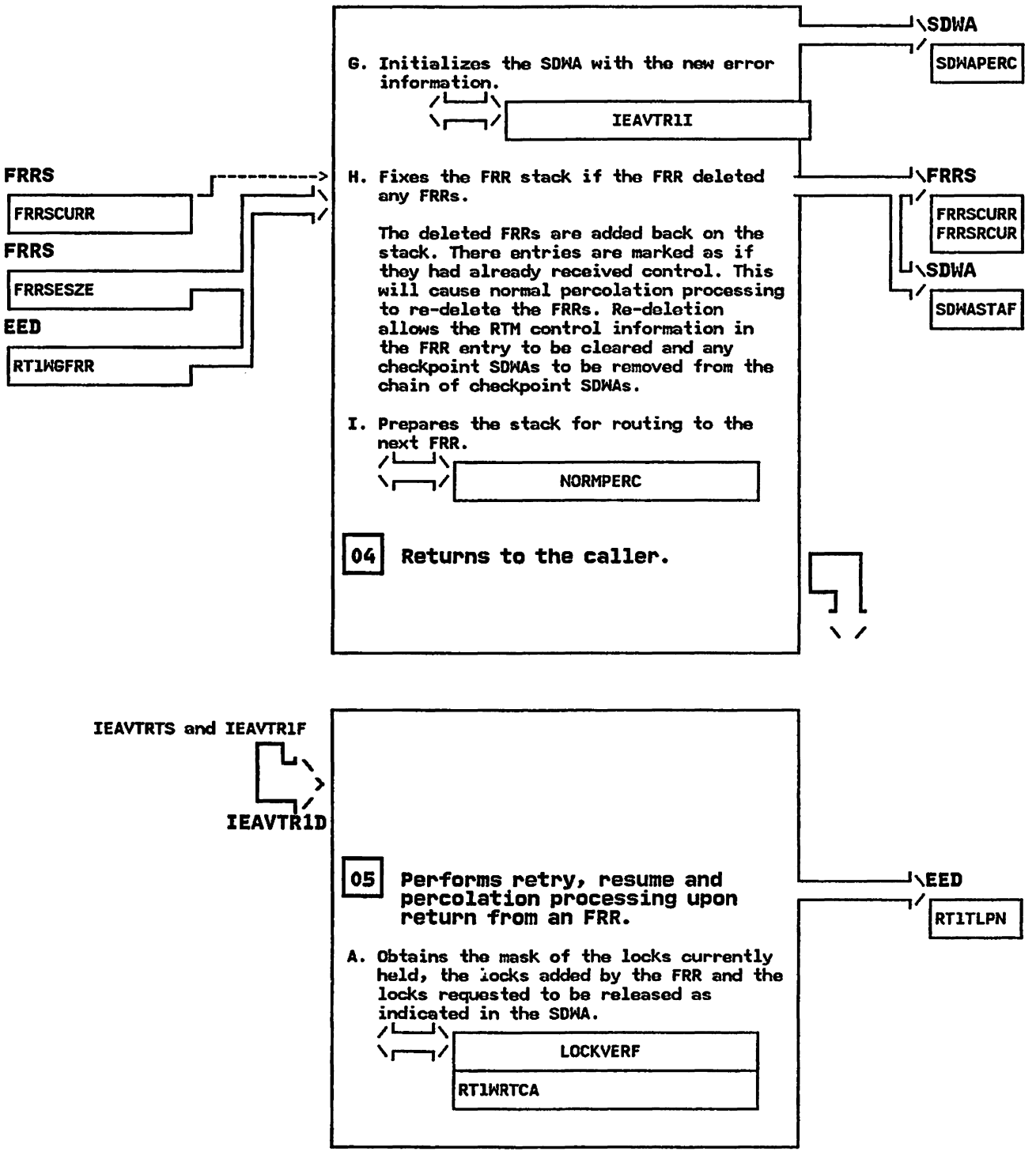
STEP 01C





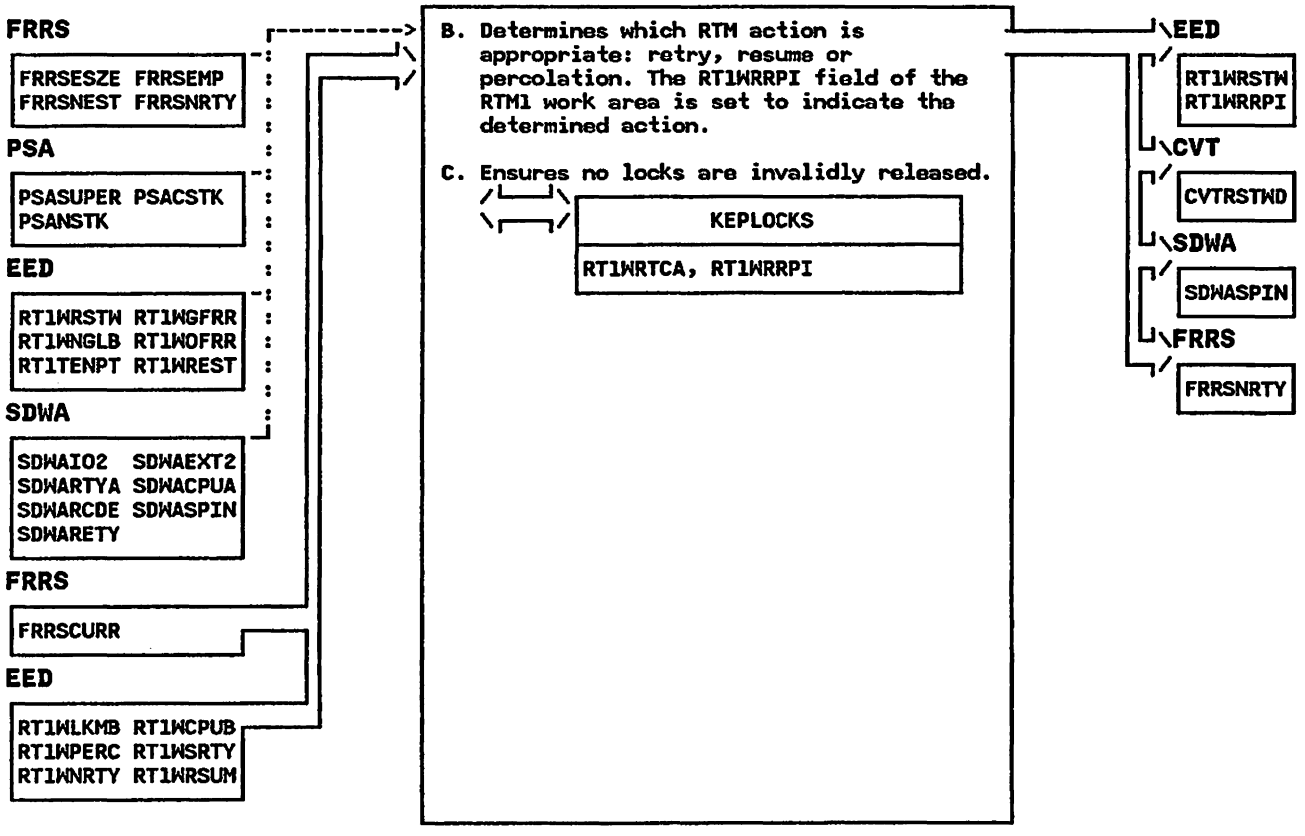
IEAVTR1C - Service Module for IEAVTRTS

STEP 03G

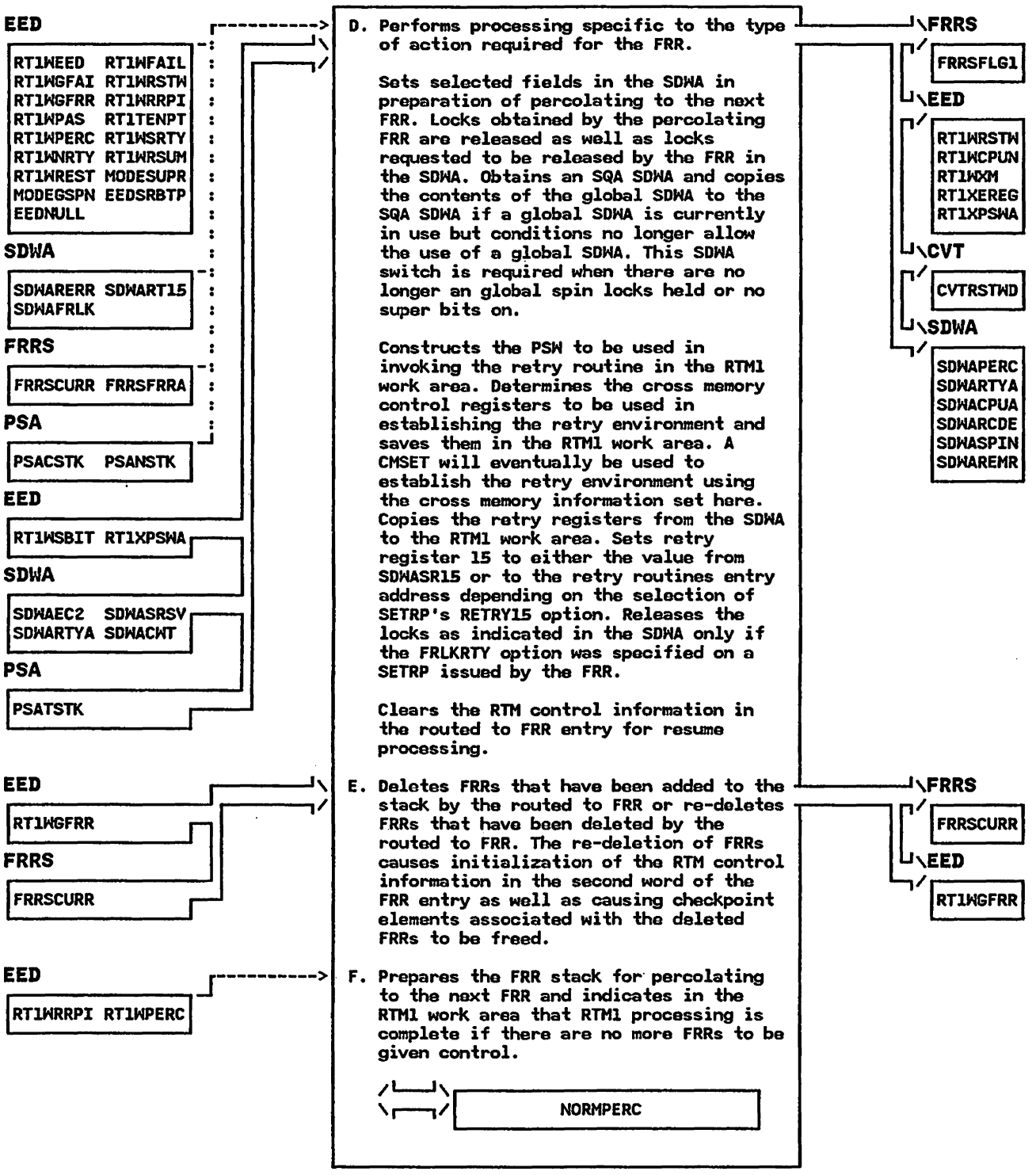


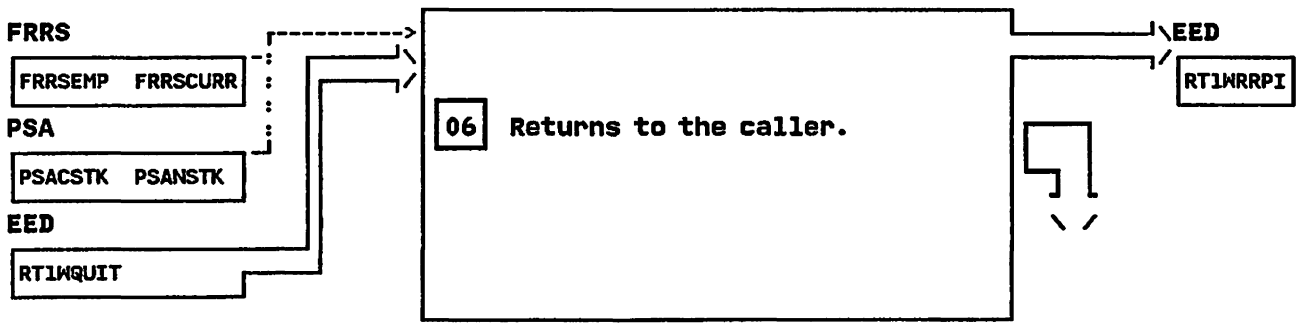
IEAVTR1C - Service Module for IEAVTRTS

STEP 05B



IEAVTR1C - Service Module for IEAVTRTS





IEAVTR1F - MODULE DESCRIPTION

DESCRIPTIVE NAME: RTM1 FRR Routing Pre-Processor

FUNCTION:
This module is called by RTM1 FRR processing to determine if there are any FRRs available and eligible to receive control. The results of this module's determination are returned to the caller via a return code.

ENTRY POINT: IEAVTR1F

PURPOSE: See function.

LINKAGE: BALR

CALLERS: IEAVTRTS

INPUT: The FRR stack

OUTPUT: Return code

EXIT NORMAL: Returns to the caller.

EXIT ERROR: Returns to the caller.

EXTERNAL REFERENCES:

ROUTINES:
- IEAVCCML - CML lock cleanup routine (An entry point in module IEAVLKRM).
- IEAVTR1D - RTM1 FRR termination processing module (An entry point in module IEAVTR1C).

DATA AREAS: No data areas used.

CONTROL BLOCKS:

Common name	Macro ID	Usage	Function
ASCB	IHAASCB	read	Obtains the ASID of the address space whose local lock is held.
CVT	CVT	write	Clears the CVT restart resource.
FRRS	IHAFRRS	read and write	Updates FRR entries and the FRR stack header.
LCCA	IHALCCA	read	Obtains address of CPU work save area.
PSA	IHAPSA	read	Obtains addresses of various FRR stacks and the LCCA. Also used by SETFRR expansion.
RT1W	IHART1W	read and write	Obtains and changes RTM control information.
SDWA	IHASDWA	read and write	Uses error information and sets status information.

TABLES: No tables used.

SERIALIZATION:
IEAVTR1F obtains the CPU lock to maintain disablement while processing in IEAVCCML.

IEAVTR1F - MODULE OPERATION

IEAVTR1F receives control to determine whether or not any FRR entries are eligible to receive control to perform recovery processing.

IEAVTR1F considers an FRR as eligible to receive control if one of the following is true:

- A super stack is the active FRR stack. A super stack is always considered to have an eligible FRR entry.
- The normal stack has at least one FRR entry and a DAT error did not occur.
- The normal stack has at least one FRR entry, a DAT error occurred, and at least one super bit is on.
- The normal stack has an entry that was established with the MODE=GLOBAL option, currently holds a global spin lock, a DAT error occurred, and all super bits are off. The FRR entries established with the MODE=GLOBAL attribute will be the only FRR entries to receive control. All other entries will not receive control.
- The normal stack has an entry that was established with the MODE=LOCAL option, no global spin locks are held, no MODE=GLOBAL FRR entries exist, a DAT error occurred, and all super bits are off. The local lock of another address space must be held for MODE=LOCAL FRRs to be processed. The MODE=LOCAL FRRs will run in another address space. The processing to accomplish this space switch is performed by IEAVCCML.

If an FRR entry does not meet one of the previous conditions to be considered eligible to receive control, IEAVTR1F deletes all entries from the stack.

In concluding its processing, IEAVTR1F issues a return code to the caller of either zero indicating that an eligible FRR entry does exist, or four indicating that there are no FRR entries to process.

RECOVERY OPERATION:

RTM1 default recovery processing (contained in module IEAVTRTR) protects most of IEAVTR1F's processing against errors. During MODE=LOCAL FRR processing involving module IEAVCCML, IEAVTR1F establishes an FRR environment.

IEAVTR1F - DIAGNOSTIC AIDS

ENTRY POINT NAME: IEAVTR1F

MESSAGES: None

ABEND CODES: None

WAIT STATE CODES: None

RETURN CODES:

EXIT NORMAL:

Register 15 contains one of the following return codes:
0 - There is an FRR eligible to receive control
4 - There are no more FRRs to process

EXIT ERROR:

Same as above

REGISTER CONTENTS ON ENTRY:

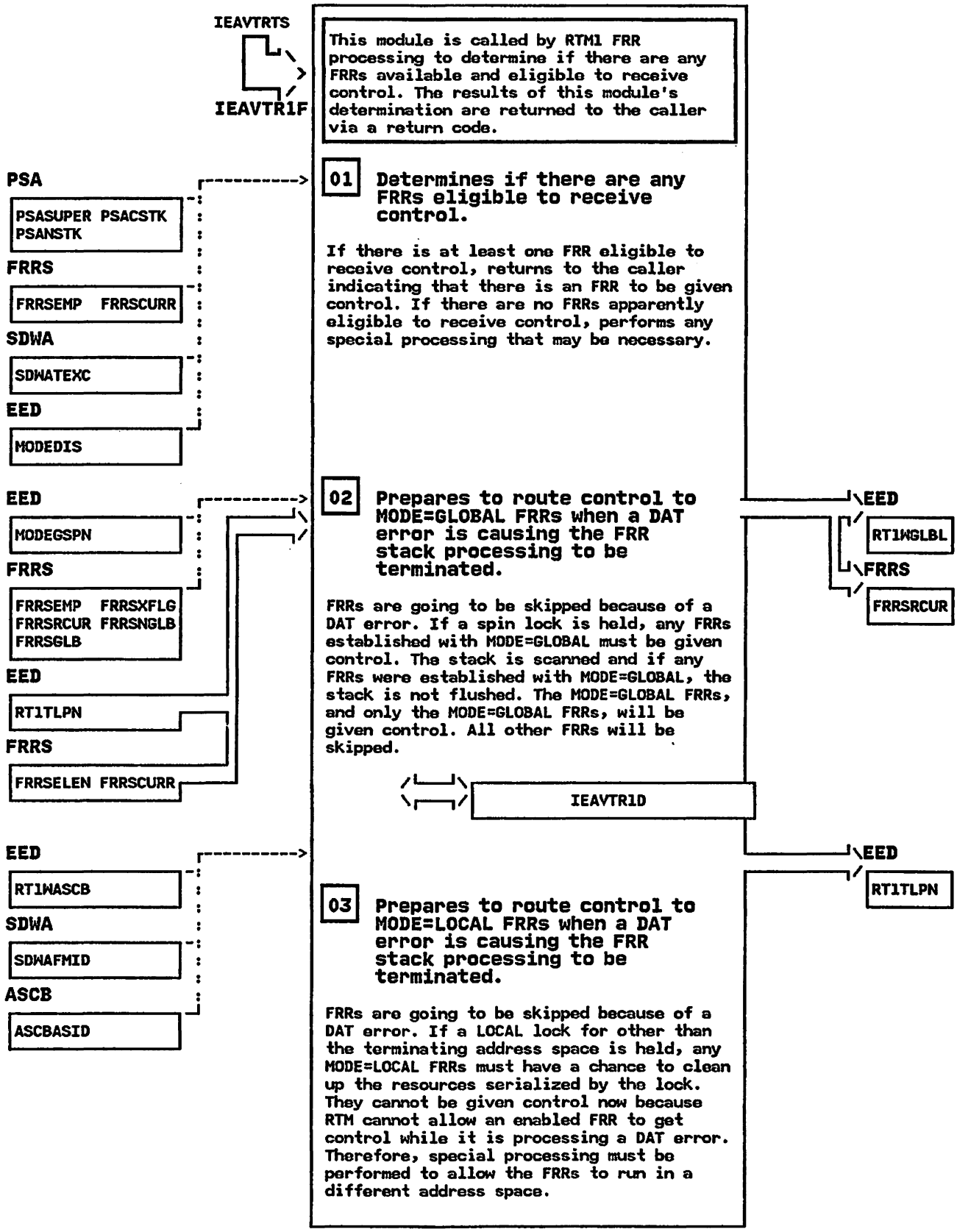
Registers 0 - 12 - Irrelevant
Register 13 - Standard register save area address
Register 14 - Return address
Register 15 - Entry point address

REGISTER CONTENTS ON EXIT:

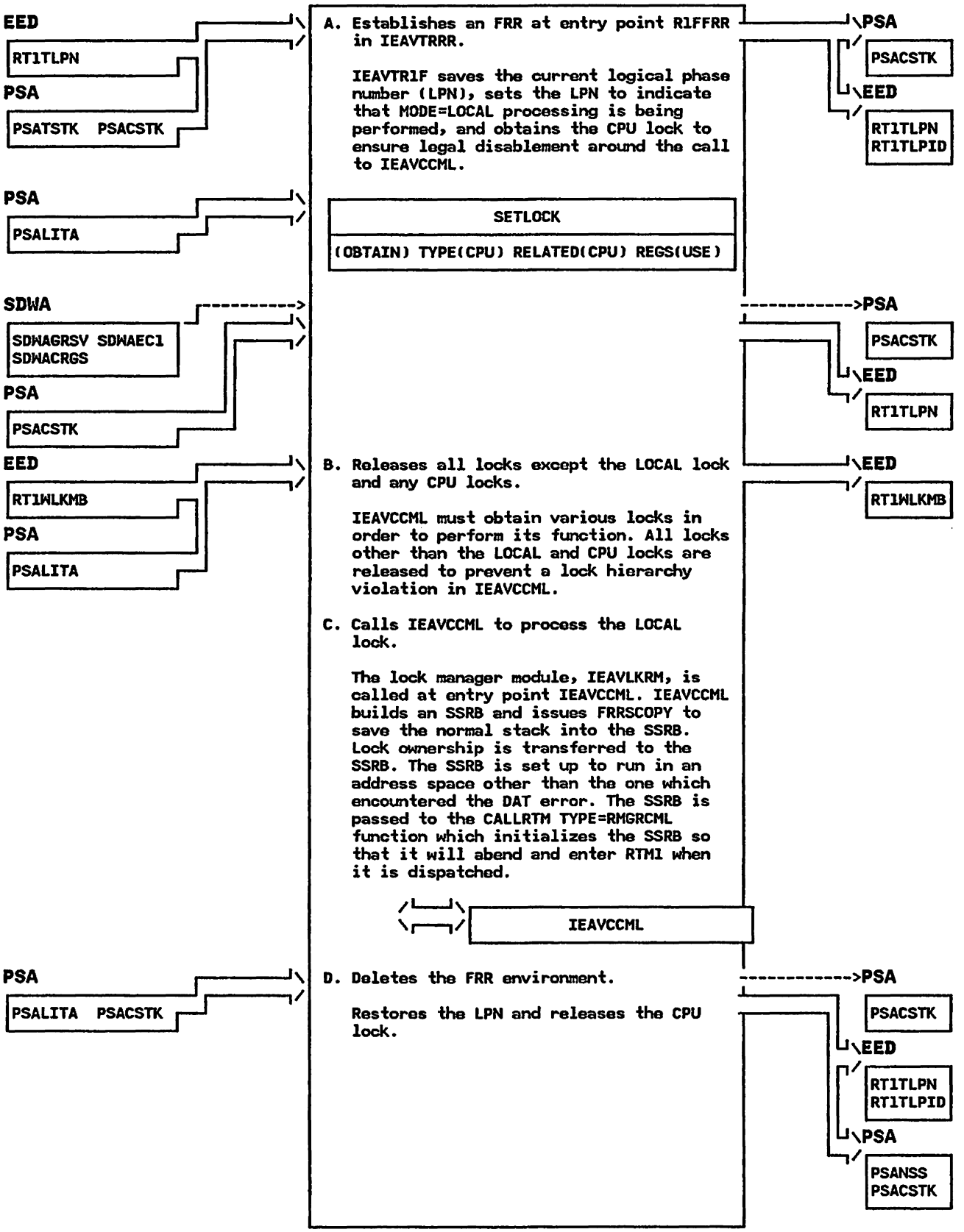
Registers 0 - 12 - Restored
Register 13 - Irrelevant
Register 14 - Restored
Register 15 - Contains a return code

IEAVTR1F - RTM1 FRR Routing Pre-Processor

STEP 01

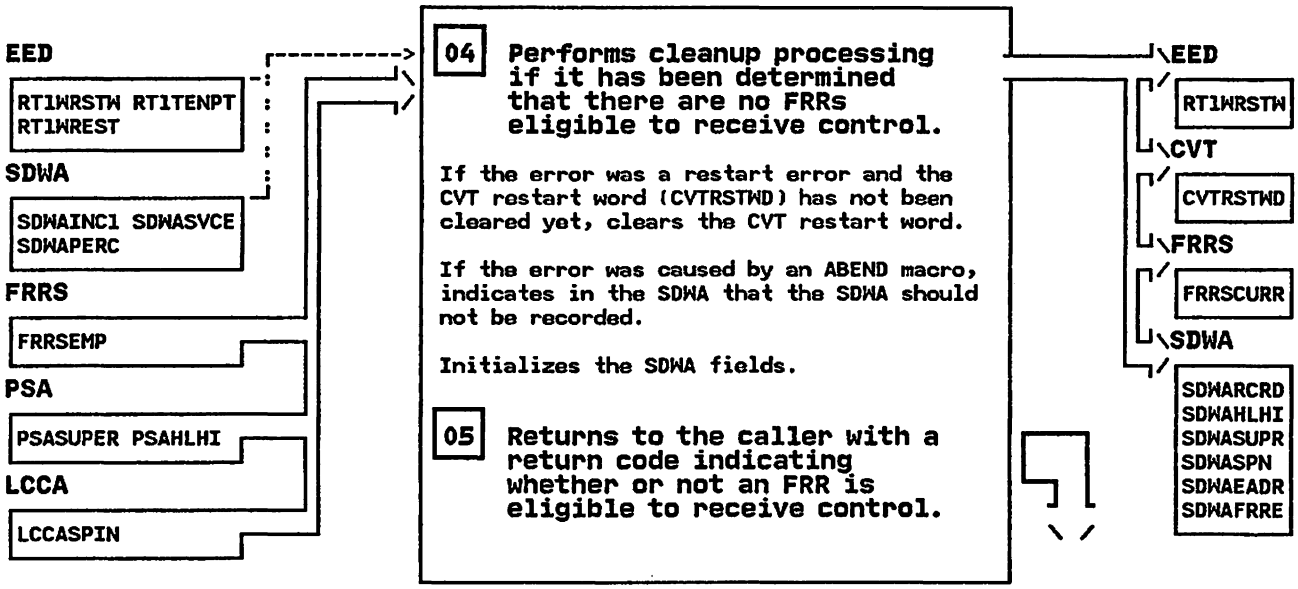


IEAVTR1F - RTM1 FRR Routing Pre-Processor STEP 03A



IEAVTR1F - RTM1 FRR Routing Pre-Processor

STEP 04



IEAVTR1G - MODULE DESCRIPTION

DESCRIPTIVE NAME: RTM1 GTF Processing Module

FUNCTION:
This module uses the general trace facility (GTF) to trace the event of receiving control back from a functional recovery routine (FRR).

ENTRY POINT: IEAVTR1G

PURPOSE: See function.

LINKAGE: BALR

CALLERS: IEAVTRTS

INPUT: None

OUTPUT: None

EXIT NORMAL: Returns to the caller.

EXIT ERROR: Returns to the caller.

EXTERNAL REFERENCES:

ROUTINES:
R1GFRR - FRR Recovery Routine for this module
(An entry point in IEAVTRRR).

DATA AREAS: No data areas used

CONTROL BLOCKS:

Common name	Macro ID	Usage	Function
FRRS	IHAFRRS	read and write	Used by SETFRR macro.
LCCA	IHALCCA	read	Obtains address of CPU work save area.
PSA	IHAPSA	read	Obtains addresses of various FRR stacks, and LCCA. Also used by SETFRR expansion.
RT1W	IHART1W	read	Obtains RTM control information.
WSAVT	IHAWSAVT	read	Obtains RTM work save area address.

TABLES: No tables used.

SERIALIZATION:
IEAVTR1G does not obtain any locks. IEAVTR1G runs disabled to serialize the RTM stacks.

IEAVTR1G - MODULE OPERATION

IEAVTR1G receives control whenever RTM1 FRR processing has completed the action of routing to an FRR.

IEAVTR1G performs the following processing:

- Saves the callers logical phase number (LPN).
- Establishes an FRR environment at entry point R1GFRR in module IEAVTRRR.
- Issues the HOOK macro to invoke GTF.
- Deletes the FRR environment and restores the callers LPN.

Note. The process of tracing FRR activity is bypassed if the current FRR stack is the PCFLIH stack.

RECOVERY OPERATION:

IEAVTR1G establishes an FRR environment to protect its processing errors.

IEAVTR1G - DIAGNOSTIC AIDS

ENTRY POINT NAME: IEAVTR1G

MESSAGES: None

ABEND CODES: None

WAIT STATE CODES: None

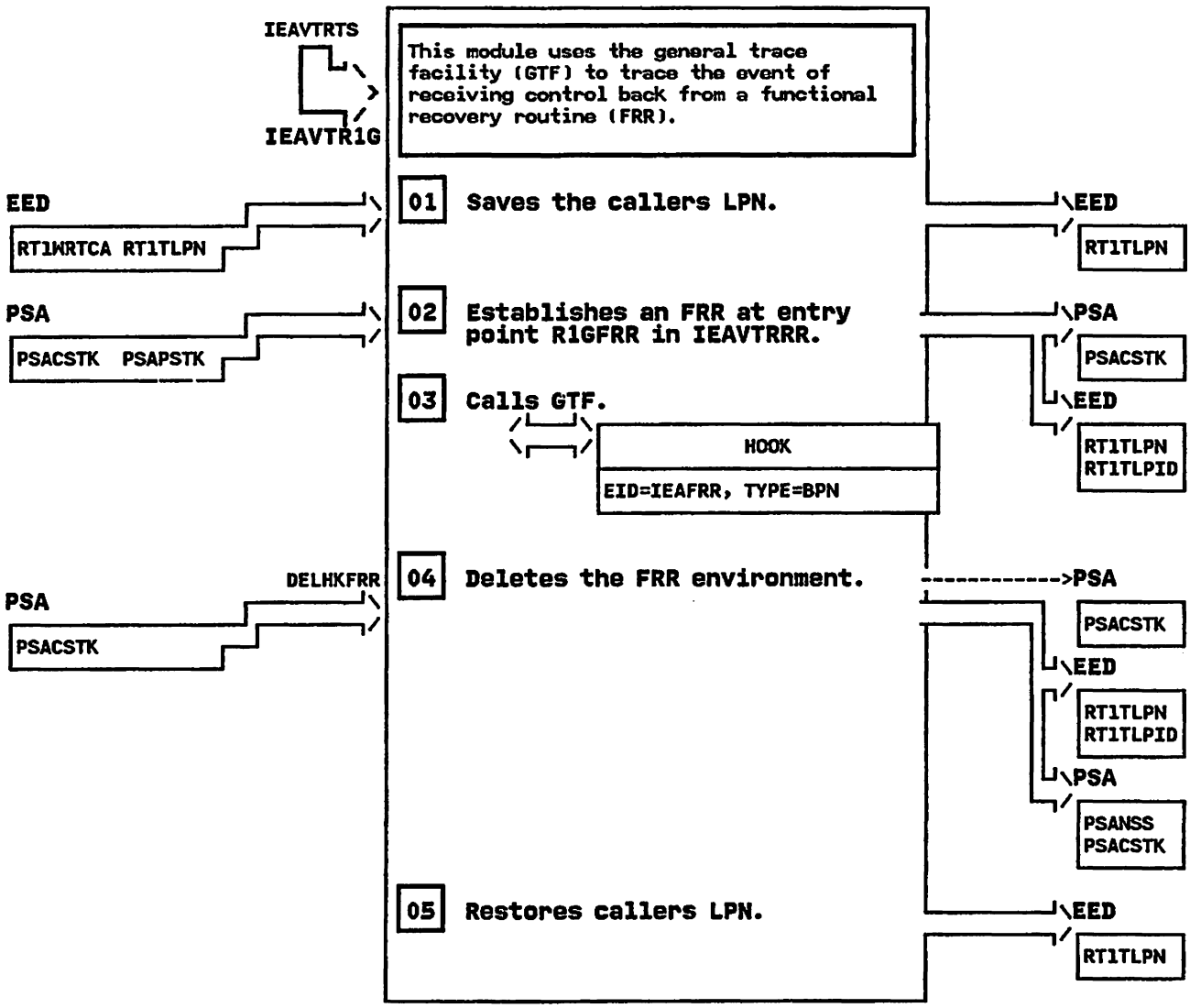
RETURN CODES: None

REGISTER CONTENTS ON ENTRY:

Registers 0 - 12 - Irrelevant
Register 13 - Standard register save area address
Register 14 - Return address
Register 15 - Entry point address

REGISTER CONTENTS ON EXIT:

Registers 0 - 15 - Restored



IEAVTR1I - MODULE DESCRIPTION

DESCRIPTIVE NAME: RTM1 General SDWA Initialization Module

FUNCTION:
This module is called by RTM1 FRR processing to initialize an SDWA with general error information.

ENTRY POINT: IEAVTR1I

PURPOSE: See function.

LINKAGE: BALR

CALLERS: IEAVTR1C

INPUT: The RTM1 work area and FRR stack

OUTPUT: An initialized SDWA

EXIT NORMAL: Returns to the caller.

EXIT ERROR: There are no exit error conditions.

EXTERNAL REFERENCES:

ROUTINES:
- R1IFRR - FRR Recovery Routine for this module (In module IEAVTRRR)
- IEAVTR1A - RTM1 Failing Instruction Processor

DATA AREAS: No data areas used.

CONTROL BLOCKS:

Common name	Macro ID	Usage	Function
ASCB	IHAASCB	read	Obtains the address space id (ASID).
ASTE	IHAASTE	read	Obtains the segment table address.
CVT	CVT	read	Obtains the RTCT address and time-of-day information.
FRRS	IHAFRRS	read	Obtains the RTM1 work area address. Also used by the SETFRR macro.
LCCA	IHALCCA	read	Obtains address of CPU work save area, translation exception address, SRB's related task information.
PSA	IHAPSA	read and write	Obtains addresses of various FRR stacks, the current ASCB and TCB, the LCCA and logical CPU id. Also used by SETFRR expansion.
RTCT	IHARTCT	read and write	Obtains error id sequence number.
RTM2WA	IHARTM2A	read	Obtains error id information.
RT1W	IHART1W	read and write	Obtains RTM control information and error information.
SDWA	IHASDWA	write	Contains initialized fields with error information.
TCB	IKJTCB	read	Obtains error type, mode at time of error and RTM2 work area.
WSAVT	IHAWSAVT	read	Obtains RTM work save area address.

TABLES: No tables used.

IEAVTR1I - MODULE DESCRIPTION (Continued)

SERIALIZATION:
IEAVTR1I runs disabled to serialize the FRR stack, the RTM1 work area, and other RTM1 control information.

IEAVTR1I - MODULE OPERATION

IEAVTR1I receives control from RTM1 FRR processing to initialize an SDWA with the following error information:

- Copies the abend completion code and reason code (specified on either the CALLRTM or ABEND macro) into the SDWA.

- Copies the error type (determined by the TYPE parameter of the CALLRTM macro that is used to invoke RTM processing) to set an error type indicator in the SDWA. The error types are program check, machine check, DAT error, SVC error, restart error and STERM re-entry. Some of the error types require additional processing in order to provide all the error information in the SDWA. These are:

  1) Machine check error
     The machine check handler passes RTM information about the machine check in EEDs. If the machine check handler provided an error id in the EEDs, RTM will use it instead of generating a new one in later processing. Also, various machine check error information is copied from the EEDs to the SDWA.

  2) SVC error
     Some SVC error entries might be caused by SRB to task percolation processing. For this type of SVC error, information about the error suffered by the SRB is passed to RTM via EEDs. The SDWA must be initialized with the SRBs error information obtained from these EEDs. The information taken from these EEDs includes the error id, 12 bytes of failing instruction stream, dump ranges and storage lists as well as the type of error suffered by the SRB (e.g., program check). Additional processing is required for some of the error types. For example, if the SRB suffered a machine check, there will be an EED passed containing the machine check error information. The information from this EED must be used to initialize machine check information in the SDWA.

  3) STERM Re-entry
     When RTM is entered for an STERM re-entry, the TCB contains the original error type. This error type is used to set the SDWA error type information. In addition, if the original error was a machine check, there will be an EED containing the machine check error data. The information from this EED must be used to initialize machine check information in the SDWA.

- Copies the error registers and error PSWs from the RTM1 work area extension (RT1X) into the SDWA. The source of the register and PSW information is dependent on the error type. Processing performed by IEAVTRT1 and IEAVTR10 has made the source transparent to this module. The failing instruction module, IEAVTR1A, is called to initialize the SDWAFAIN field with the 6 bytes of instruction stream on either side of the byte pointed to by the error PSW.

- Copies the mode of the system at the time of error in the SDWAERRB field. This field indicates whether the system was in SRB mode or task mode, whether a type 1 SVC was processing, or whether any locks were held.

- After an error id is generated, copies it into various SDWA fields. The error id can be used to associate an SDWA that is recorded in SYS1.LOGREC with an SVC dump taken by a recovery routine. It might also be used to associate

IEAVTR1I - MODULE OPERATION (Continued)

multiple occurrences of SDWAs in SYS1.LOGREC as a result of more than one FRR requesting recording of the same error. An error encountered by an FRR while processing for a preceeding error might result in more than one SDWA in SYS1.LOGREC, one for the preceeding error and one for the FRRs error. All parts of the error ids will be the same except for the time stamp. Thus, the error id can be used to associate these related errors.

. If dump options were provided, copies them into the SDWA. (Dump options might have been provided by the issuer of the CALLRTM or ABEND macro used to invoke RTM processing.) The dump options might point to a list of storage ranges or a list of subpools that must also be copied to the SDWA. FRR protection is established while copying these items.

RECOVERY OPERATION:

RTM1 default recovery processing (contained in module IEAVTRTR) protects most of IEAVTR1I's processing against errors. While copying any dump ranges, IEAVTR1I uses an FRR to protect against any errors occurring during this processing.

IEAVTR1I - DIAGNOSTIC AIDS

ENTRY POINT NAME: IEAVTR1I

MESSAGES: None

ABEND CODES: None

WAIT STATE CODES: None

RETURN CODES: None

REGISTER CONTENTS ON ENTRY:

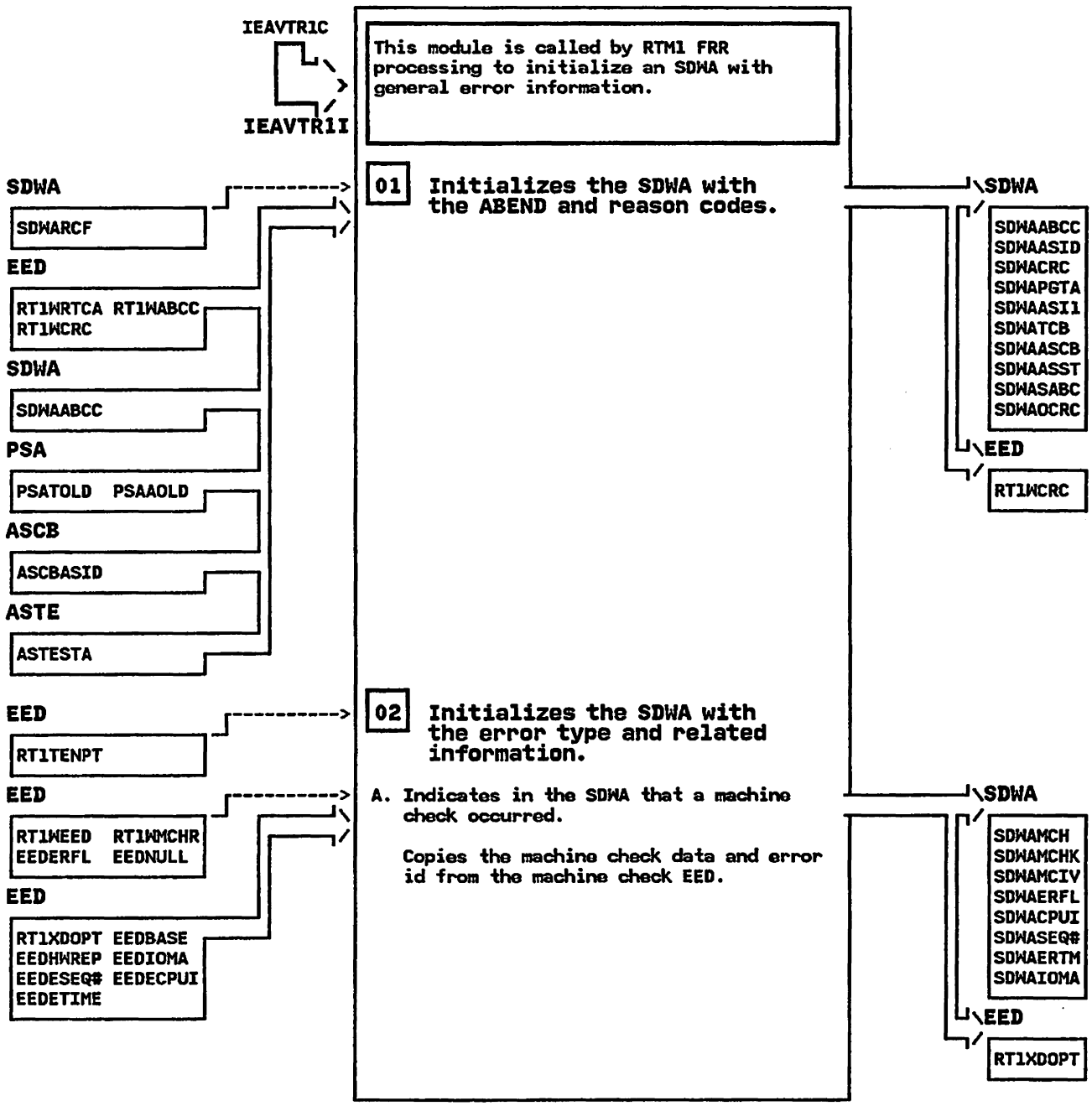
Registers 0 - 12 - Irrelevant
Register 13 - Standard register save area address
Register 14 - Return address
Register 15 - Entry point address

REGISTER CONTENTS ON EXIT:

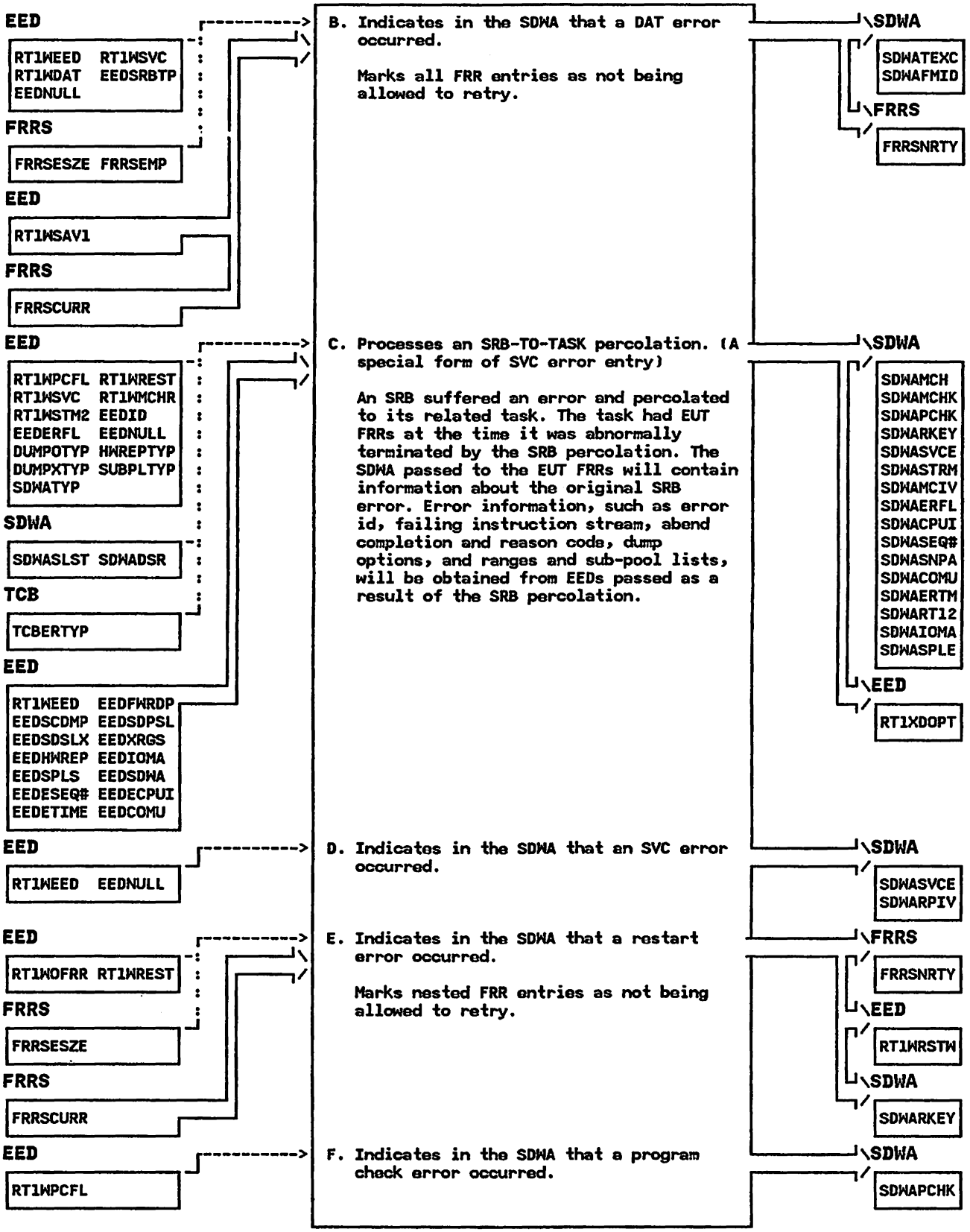
Registers 0 - 15 - Restored

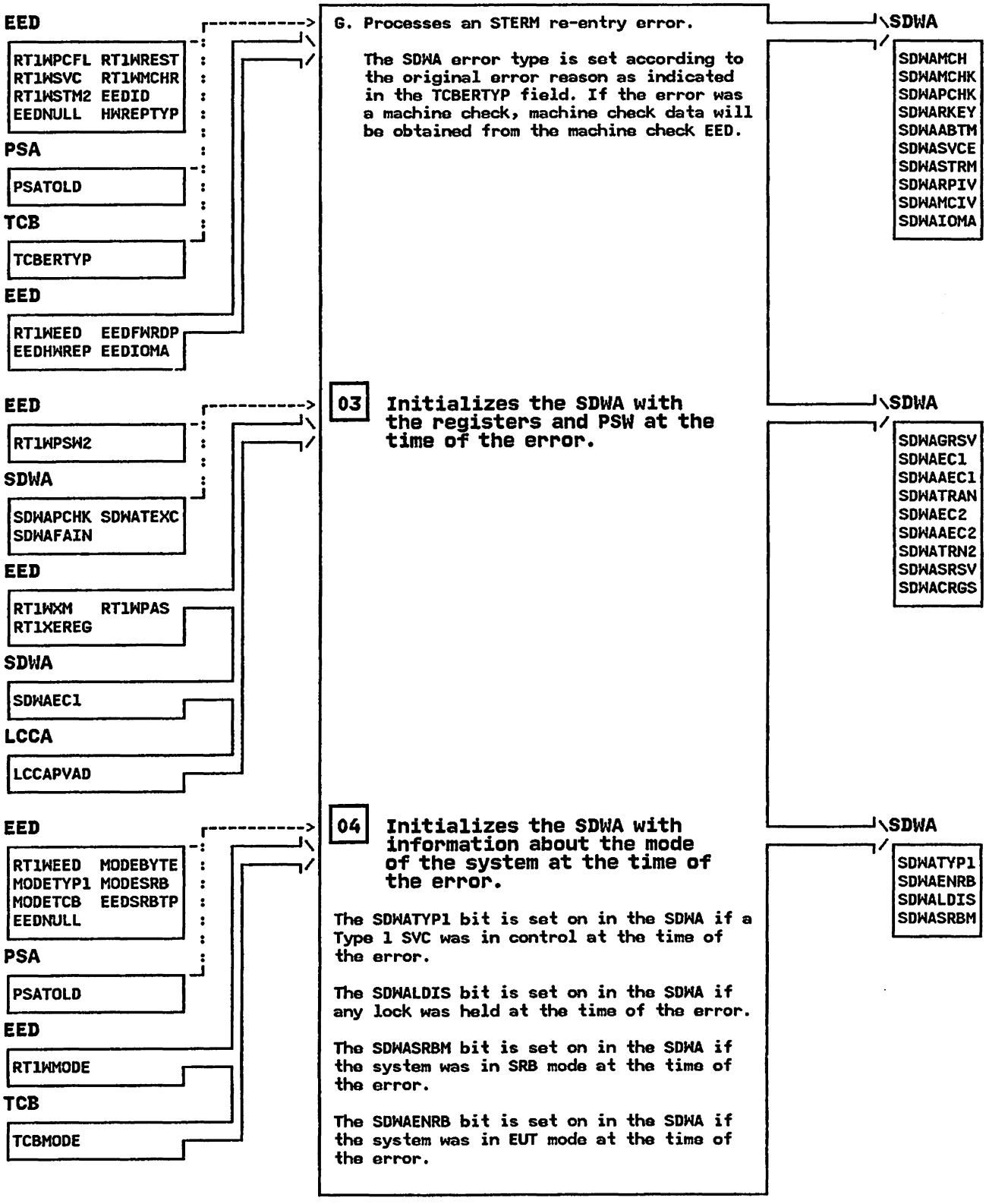
IEAVTR1I - RTM1 General SDWA Initialization Module

STEP 01

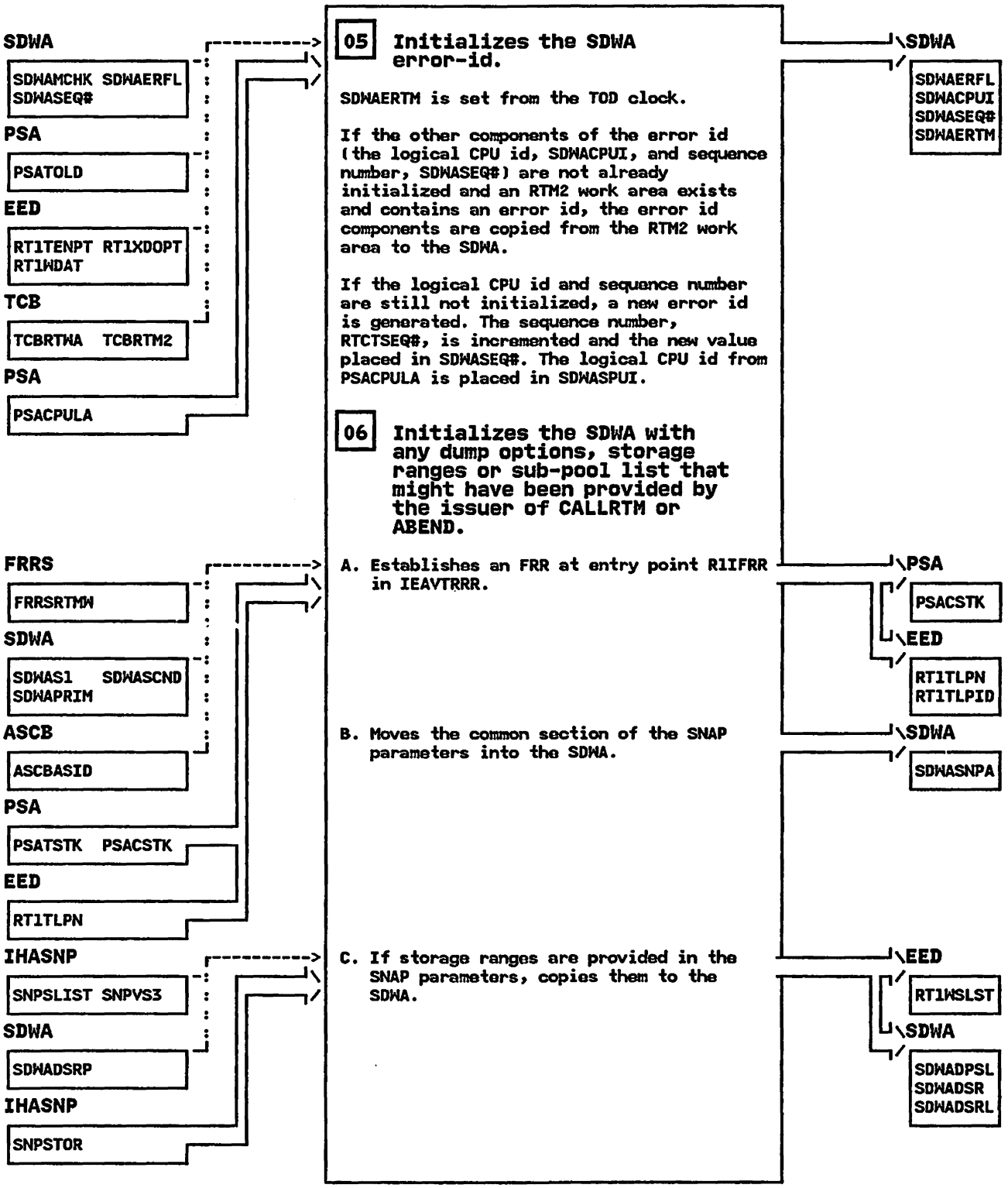


IEAVTR1I - RTM1 General SDWA Initialization Module STEP 02B



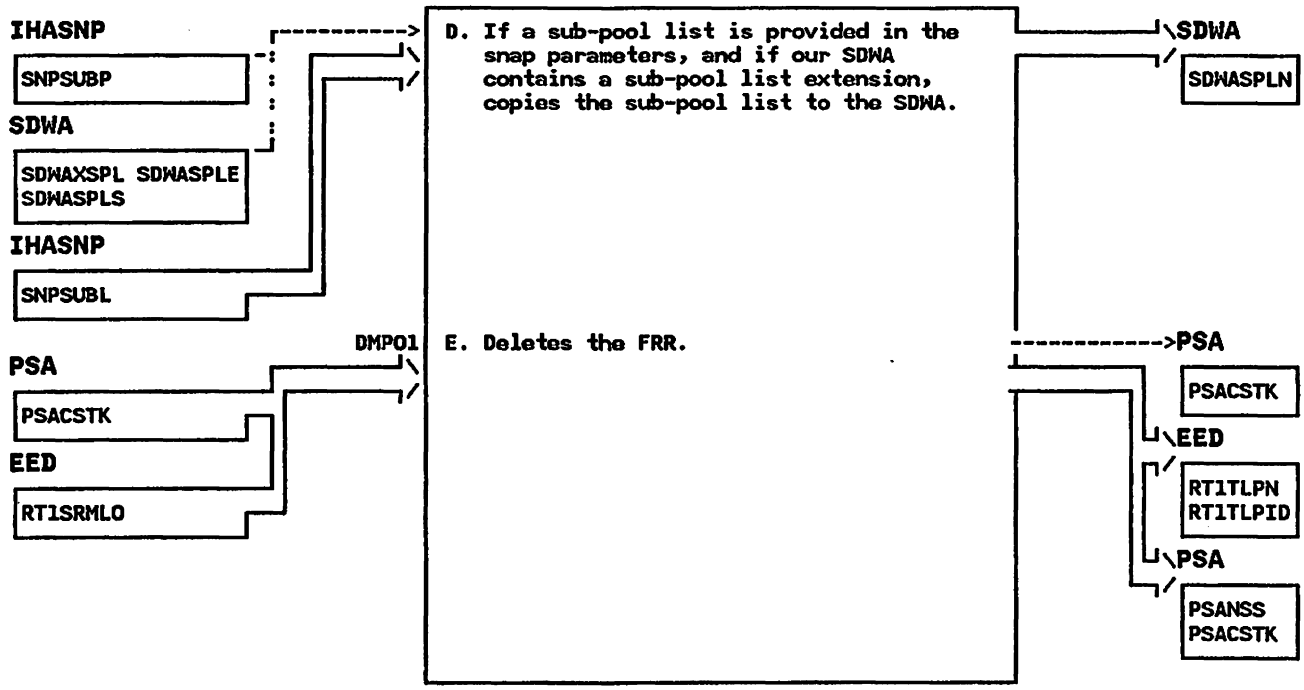


IEAVTR1I - RTM1 General SDWA Initialization Module STEP 05



IEAVTR1I - RTM1 General SDWA Initialization Module

STEP 06D



IEAVTR1N - MODULE DESCRIPTION

DESCRIPTIVE NAME: FRR Stack Initialization

FUNCTION:
This module initializes the FRR stacks during system initialization (NIP) and for Vary CPU (online) so that SETFRR recovery can be defined. The Recovery Stack Vector Table (RSVT) in the PSA is initialized to point to the FRR stacks.

ENTRY POINT: IEAVTR1N

PURPOSE: Initialize the FRR stack

LINKAGE: Standard linkage on procedure call

CALLERS: IEAVNIP0, IEEVCPR

INPUT:
Register 1 points to the PSA of processor to be initialized.
Register 13 points to a 72 byte register save area.
Also see Dependencies.
ENQ/Lock Conditions: Same as caller

OUTPUT: None

EXIT NORMAL: Returns to caller

EXTERNAL REFERENCES:

ROUTINES: None

DATA AREAS:
PSA, CVT, FRR stacks, vector table of lengths of FRR stacks (mapped by IHAYSTAK), LCCA

CONTROL BLOCKS: PSA, CVT, LCCA

IEAVTR1N - MODULE OPERATION

The Recovery Stack Vector Table (RSVT) in the PSA is initialized to point to the respective FRR stacks:

- PSACSTK - points to the current FRR stack and is initialized to point to the normal FRR stack
- PSASSTK - initialized to point to the SVC-I/O-Dispatcher FRR stack
- PSANSTK - initialized to point to the normal FRR stack
- PSAMSTK - initialized to point to the machine check FRR stack
- PSAPSTK - initialized to point to the program check FRR stack
- PSAESTK1 - initialized to point to the external FLIH FRR stack (non-recursive entries)
- PSAESTK2 - initialized to point to the external FLIH FRR stack (first level recursions)
- PSAESTK3 - initialized to point to the external FLIH FRR stack (second level recursions)
- PSARSTK - initialized to point to the restart FLIH FRR stack
- PSATSTK - initialized to point to the RTM1 FRR stack. Note that this pointer is not contained in the RSVT table
- PSAASTK - initialized to point to the ACR FRR stack. Note that this pointer is not contained in the RSVT table

The supervisor control FRR stack(s) initialization is performed as follows:

1) The entire stack is zeroed
2) The stack header (4 words) is initialized to appear as follows:
   - 1st word - points to the first FRR entry in the stack
   - 2nd word - points to the last FRR entry in the stack
   - 3rd word - contains the length of each FRR entry
   - 4th word - contains the same pointer as the 1st word
3) The first stack entry is initialized to point to the supervisor control FRR (IEAVSCRU). The CVTSPFRR field of the CVT contains the address of the super FRR. EXCEPTION: The first stack entry in the ACR stack will point to ACR's own FRR routine.

The normal FRR stack initialization is performed as follows:

1) The entire FRR stack is zeroed
2) The stack header (4 words) is initialized as follows:
   - 1st word - contains the address of the first FRR entry minus 32 bytes
   - 2nd word - contains the address of the last FRR entry in the stack
   - 3rd word - contains the length value of an FRR entry
   - 4th word - contains the same pointer as the 1st word

RECOVERY OPERATION: IEAVTR1N does not provide its own recovery environment.

IEAVTR1N - DIAGNOSTIC AIDS

ENTRY POINT NAME: IEAVTR1N

MESSAGES: None

ABEND CODES: None

WAIT STATE CODES: None

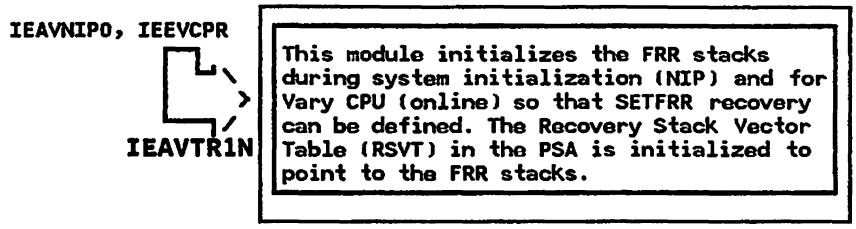
RETURN CODES: None

REGISTER CONTENTS ON ENTRY:

Register 1 - Address of PSA
Register 13 - Address of the caller's register save area
Register 14 - Return address
Register 15 - Entry point address

REGISTER CONTENTS ON EXIT: Irrelevant

IEAVTR1N - FRR Stack Initialization



IEAVTR1R - MODULE DESCRIPTION

DESCRIPTIVE NAME: RTM1 RECORD Interface Module

FUNCTION:
This module is called by RTM1 FRR processing to write diagnostic error information that is contained in the SDWA to the SYS1.LOGREC data set using the RECORD macro instruction.

ENTRY POINT: IEAVTR1R

PURPOSE: See function.

LINKAGE: BALR

CALLERS: IEAVTRTS and IEAVTR1C

INPUT:
A parameter indicating the SDWA to be recorded. A parameter value of 0 indicates that the primary SDWA pointed to by the RTM1 work area is to be recorded. A nonzero parameter value indicates that a checkpoint SDWA pointed to by a checkpoint element (RT1I) is to be recorded. The parameter value is the address of the checkpoint element (an RT1I).

OUTPUT: SYS1.LOGREC contains diagnostic information.

EXIT NORMAL: Returns to the caller.

EXIT ERROR: Returns to the caller.

EXTERNAL REFERENCES:

ROUTINES:
R1RFRR - FRR Recovery Routine for this module (Entry in module IEAVTRRR)

DATA AREAS: No data areas used.

CONTROL BLOCKS:

Common name	Macro ID	Usage	Function
ASCB	IHAASCB	read	Obtains the job name to be used in the SYS1.LOGREC record.
CVT	CVT	read	Obtains RECORD service routine entry address.
FRRS	IHAFRRS	read	Obtains FRR entry information and FRR stack header.
LCCA	IHALCCA	read	Obtains address of CPU work save area.
PSA	IHAPSA	read and write	Obtains addresses of various FRR stacks, the ASCB and LCCA. Also used by SETFRR expansion.
RT1W	IHART1W	read and write	Obtains RTM control information.
SDWA	IHASDWA	read and write	Obtains diagnostic error information.
WSAVT	IHAWSAVT	read	Obtains RTM work save area address.

TABLES: No tables used.

SERIALIZATION:
IEAVTR1R does not obtain any locks. IEAVTR1R runs disabled to serialize the RTM1 work area.

IEAVTR1R - MODULE OPERATION

IEAVTR1R receives control to determine if diagnostic error information that is contained in an SDWA should be recorded in SYS1.LOGREC data set. Information in an SDWA is recorded if the FRR requested recording, an RTM1 error condition forced recording or SLIP forced recording as a result of the specification of ACTION=RECORD on a SLIP command.

If recording is required, the following processing is performed:

- If recording is forced because of an error in RTM1 processing, places information about the RTM1 error in the variable recording area of the SDWA. There are two errors that will cause RTM1 to force the recording of the SDWA. They are: (1) an inability to obtain storage for an SDWA (2) a failure to establish the FRRs cross memory environment.

- Establishes an FRR environment at entry point R1RFRR in module IEAVTRRR to protect RTM1 against errors in the RECORD process. The RTM super stack is made the active stack during this processing.

- Issues the RECORD macro to write the SDWA's diagnostic error information to SYS1.LOGREC.

- Deletes the FRR environment. The stack that was active at the time IEAVTR1R received control is restored as the active stack and control returns to the caller.

RECOVERY OPERATION:

RTM1 default recovery processing (contained in module IEAVTRTR) protects most of IEAVTR1R's processing against errors. During the RECORD processing, any errors occurring will be protected through an FRR.

IEAVTR1R - DIAGNOSTIC AIDS

ENTRY POINT NAME: IEAVTR1R

MESSAGES: None

ABEND CODES: None

WAIT STATE CODES: None

RETURN CODES: None

REGISTER CONTENTS ON ENTRY:

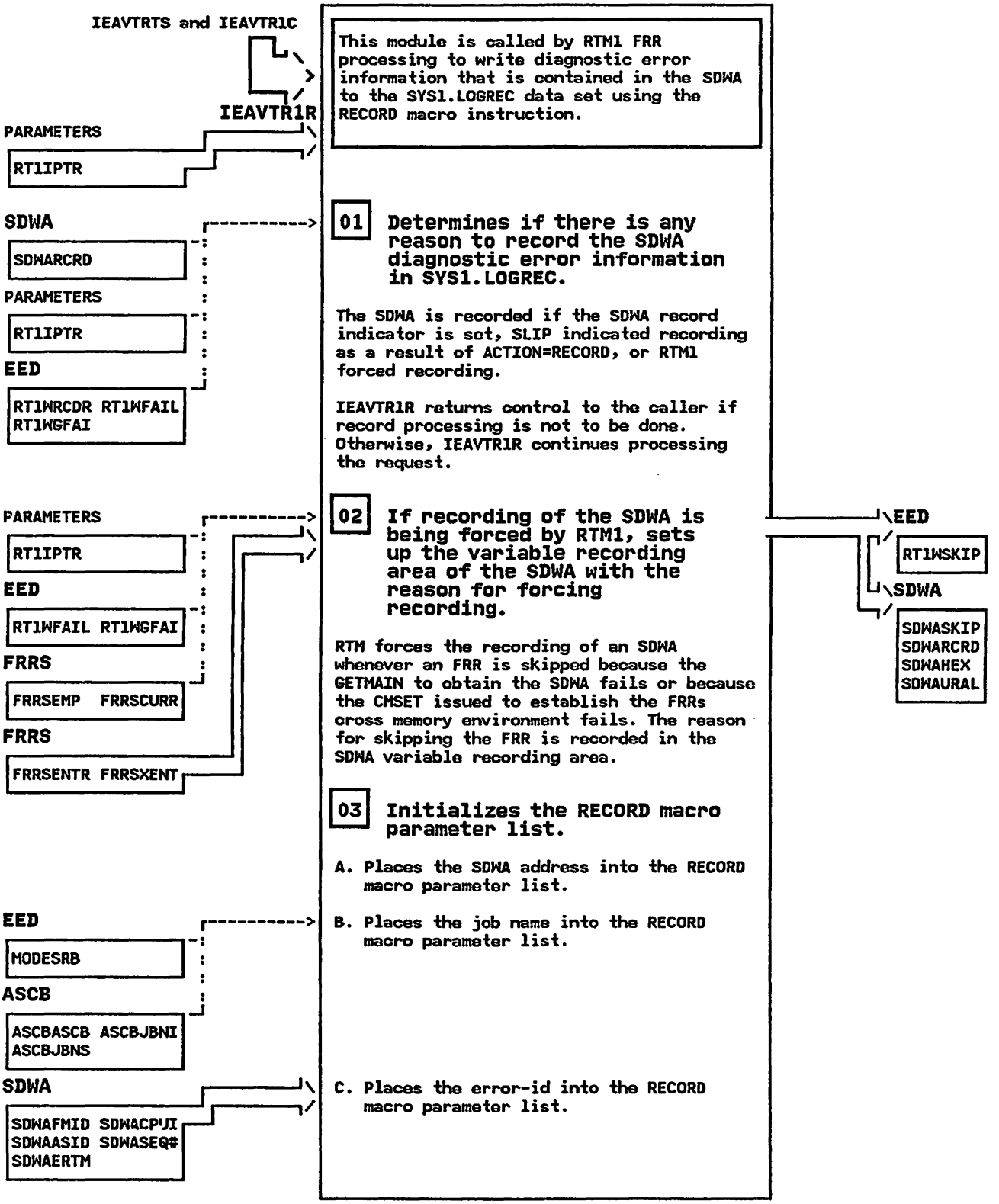
Register 0 - Irrelevant
Register 1 - Parameter address
Registers 2 - 12 - Irrelevant
Register 13 - Standard register save area address
Register 14 - Return address
Register 15 - Entry point address

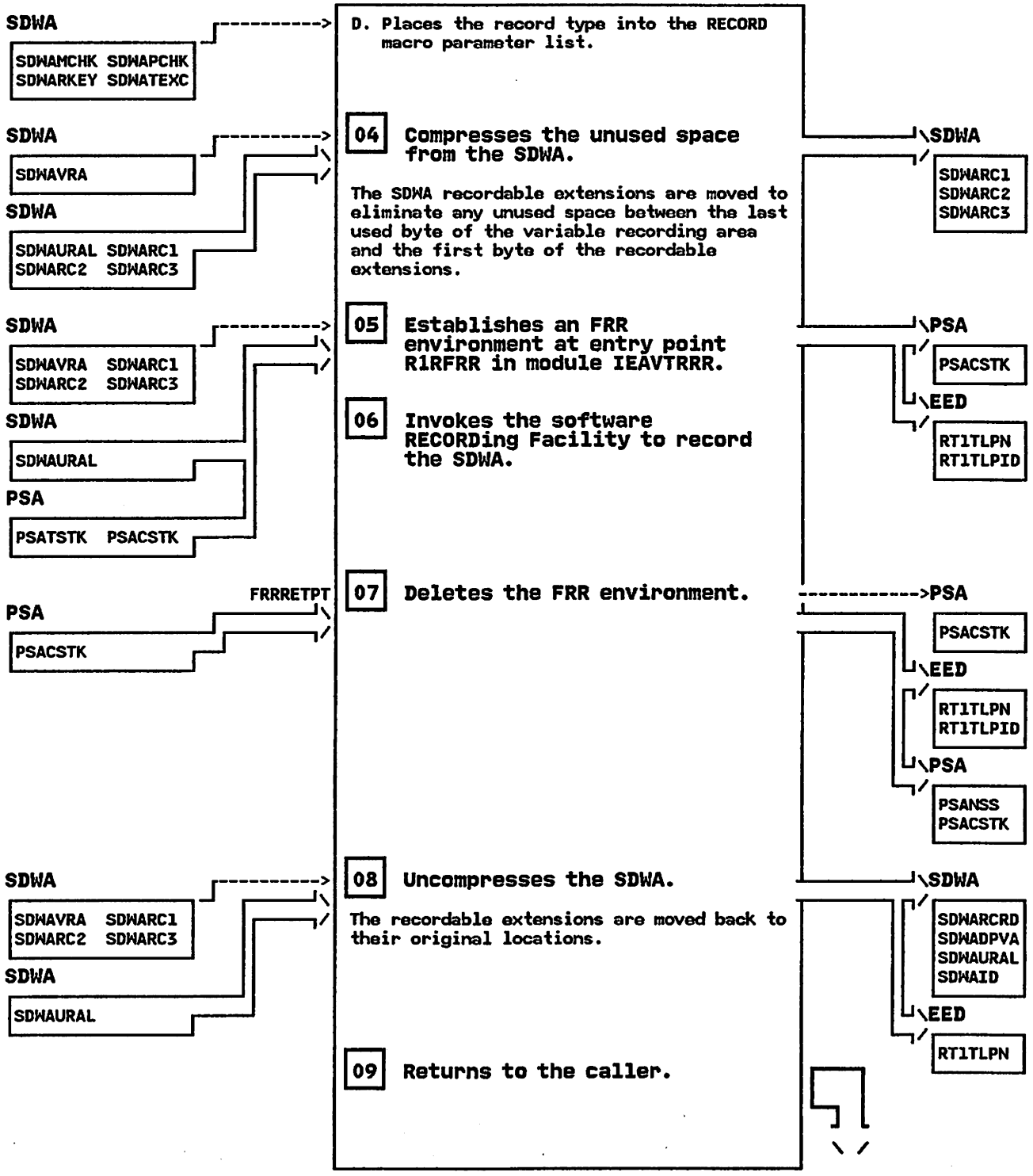
REGISTER CONTENTS ON EXIT:

Registers 0 - 15 - Restored

IEAVTR1R - RTM1 RECORD Interface Module

STEP 01





IEAVTR1S - MODULE DESCRIPTION

DESCRIPTIVE NAME: RTM1 SDWA Allocation Module

FUNCTION:
This module is called by RTM1 FRR processing whenever a request has been made to either allocate or release an SDWA.

ENTRY POINT: IEAVTR1S

PURPOSE: See function.

LINKAGE: BALR

CALLERS: IEAVTR1C and IEAVTR1O

INPUT:
Standard parameters are received as input. The parameter defines the request type, either to allocate or to free an SDWA, and whether the request is for a normal or a checkpoint SDWA.

The first parameter is a single EBCDIC character defining the request type as follows:
'A' - Allocate an SDWA
'F' - Free an SDWA

The second parameter is a single EBCDIC character further defining the action to be taken as follows:
'R' - A regular SDWA is to be processed
'C' - A checkpoint SDWA is to be processed

OUTPUT: Either an allocated or released SDWA

EXIT NORMAL: Returns to the caller.

EXIT ERROR: There are no exit error conditions.

EXTERNAL REFERENCES:

ROUTINES:
R1SFRR - FRR Recovery Routine for this module (In module IEAVTRRR)

CONTROL BLOCKS:

Common name	Macro ID	Usage	Function
ASCB	IHAASCB	read and write	Obtains the SDWA queue anchor (RTMC).
FRRS	IHAFRRS	read	Obtains address of the RTM1 work area and FRR stack header information.
LCCA	IHALCCA	read	Obtains address of CPU work save area.
PSA	IHAPSA	read and write	Obtains addresses of various FRR stacks, the ASCB and LCCA. Also used by SETFRR expansion.
RT1W	IHART1W	read and write	Obtains RTM control information.
SDWA	IHASDWA	write	Contains diagnostic information.
WSAVT	IHAWSAVT	read	Obtains RTM work save area address.
YSTAK	IHAYSTAK	read	Obtains size of an FRR stack.

TABLES: No tables are used.

IEAVTR1S - MODULE DESCRIPTION (Continued)

SERIALIZATION:

IEAVTR1S runs disabled to serialize the FRR stack, RTM1 work area, and other RTM1 control information. IEAVTR1S obtains the SALLOC lock to serialize the ASCBRTMC list.

IEAVTR1S - MODULE OPERATION

IEAVTR1S receives control whenever there is a request made by RTM1 FRR processing to either allocate or free an SDWA.

Allocating an SDWA:

The caller requests that an SDWA be allocated by indicating in the first parameter an "A" and in the second parameter either an "R" for a regular (non-checkpoint) SDWA or a "C" for a checkpoint SDWA.

If the current stack is a super FRR stack, at least one super bit is on, a global spin lock is held, or the error is a DAT error, IEAVTR1S allocates a global SDWA. IEAVTR1S then checks the second parameter to determine the type of global SDWA to allocate.

If the request is to allocate a regular SDWA, the request is satisfied with a regular global SDWA. (There is one regular global SDWA associated with each of the FRR stacks;the current stack is used to determine which of the global SDWAs must be allocated.)

If the request is for a checkpoint SDWA, the request is satisfied from the pool of global checkpoint SDWAs. (Unlike the regular global SDWAs, global checkpoint SDWAs are not preassigned to a particular stack. They are allocated as needed and any global checkpoint SDWA might be allocated to any of the stacks.) Associated with the pool is an allocation table. The table is used to track which global checkpoint SDWAs are currently allocated and, as diagnostic information, to which stack they are allocated.

If an enabled unlocked task (EUT), unlocked SRB or locally locked function suffers an error other than a DAT error, IEAVTR1S allocates an SQA SDWA from subpool 239. To satisfied the request, IEAVTR1S first determines if there are any free SDWAs anchored in the ASCB's RTMC list. If an SDWA cannot be obtained from the RTMC list, IEAVTR1S issues a GETMAIN macro for an SQA SDWA. IEAVTR1S then checks the second parameter to determine the type of global SDWA to allocate.

If the second parameter requested a regular SDWA, storage is allocated from below the 16 Meg line and the allocated SDWA is added to the ASCB's RTMC list.

If the second parameter requested a checkpoint SDWA, storage is allocated from above the 16 Meg line and the allocated SDWA is added to the ASCB's RTMC list.

After the SDWA is allocated, IEAVTR1S initializes the SDWA structure.

Freeing an SDWA:

The caller requests that an SDWA be freed by indicating in the first parameter an "F". The second parameter is irrelevant for a free request.

IEAVTR1S frees the first checkpoint element anchored off the RTM1 work area's RT1I chain. The release process is dependent on the type of checkpoint, which is indicated in the RT1IRTYP field. If the checkpoint is a global checkpoint, IEAVTR1S marks the global SDWA allocation table entry that corresponds to the checkpoint being released as free or no longer in use. If the checkpoint is an SQA

IEAVTR1S - MODULE OPERATION (Continued)

checkpoint, IEAVTR1S marks the RTMC header for the checkpoint as free or not in use. The storage for an RTMC is not actually released (via a FREEMAIN macro) until RTM1 processing has completed.

RECOVERY OPERATION:

RTM1 default recovery processing (contained in module IEAVTRTR) protects most of IEAVTR1S's processing. IEAVTR1S uses an FRR to protect against most errors while it is obtaining storage for an SDWA via a GETMAIN macro.

IEAVTR1S - DIAGNOSTIC AIDS

ENTRY POINT NAME: IEAVTR1S

MESSAGES: None

ABEND CODES: None

WAIT STATE CODES: None

RETURN CODES:

EXIT NORMAL:

0 - The SDWA was allocated or freed successfully
4 - The SDWA was not allocated

REGISTER CONTENTS ON ENTRY:

Register 0 - Irrelevant
Register 1 - Address of parameter list
Registers 2 - 12 - Irrelevant
Register 13 - Standard register save area address
Register 14 - Return address
Register 15 - Entry point address

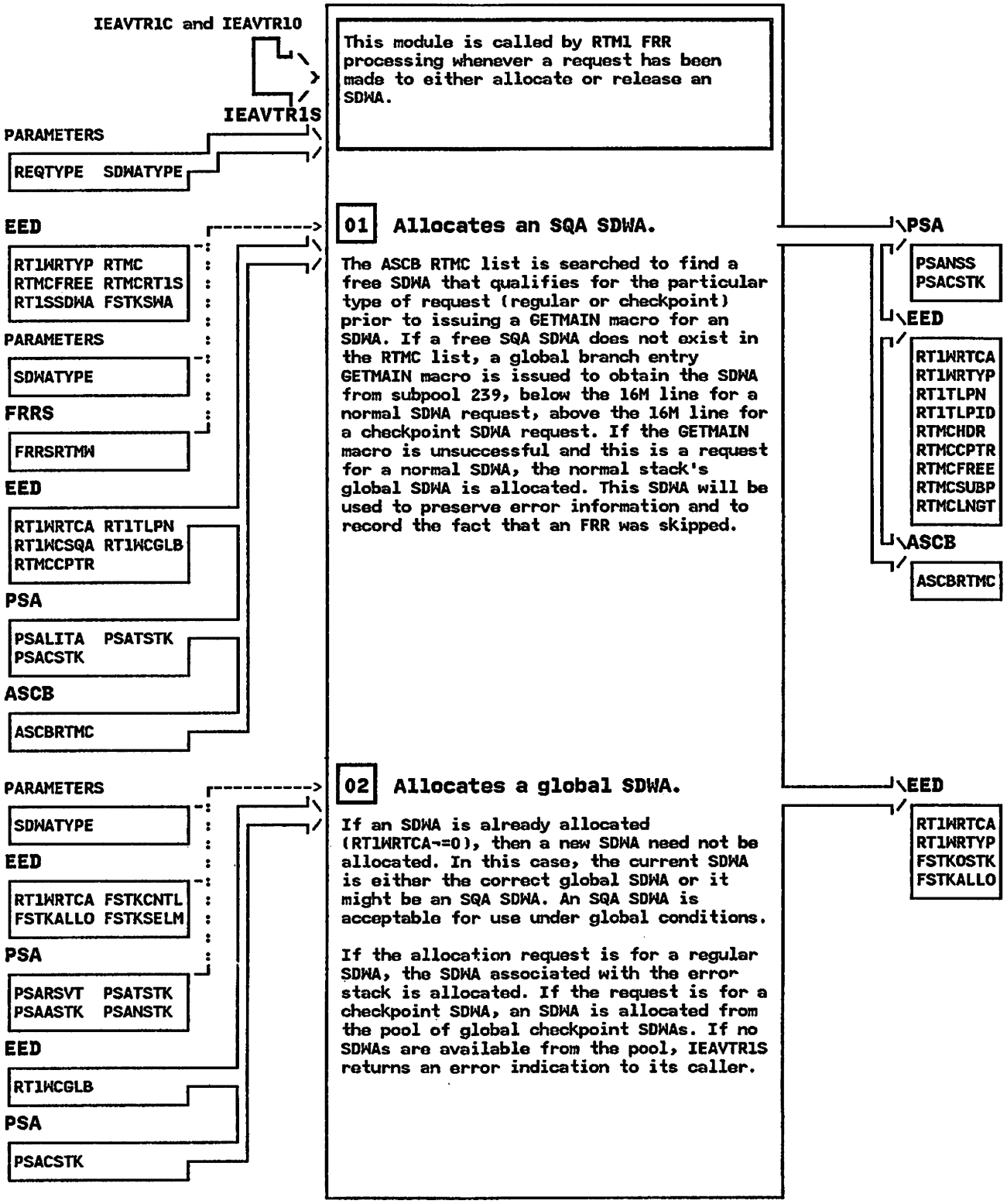
REGISTER CONTENTS ON EXIT:

EXIT NORMAL:

Registers 0 - 14 - Restored
Register 15 - Return Code

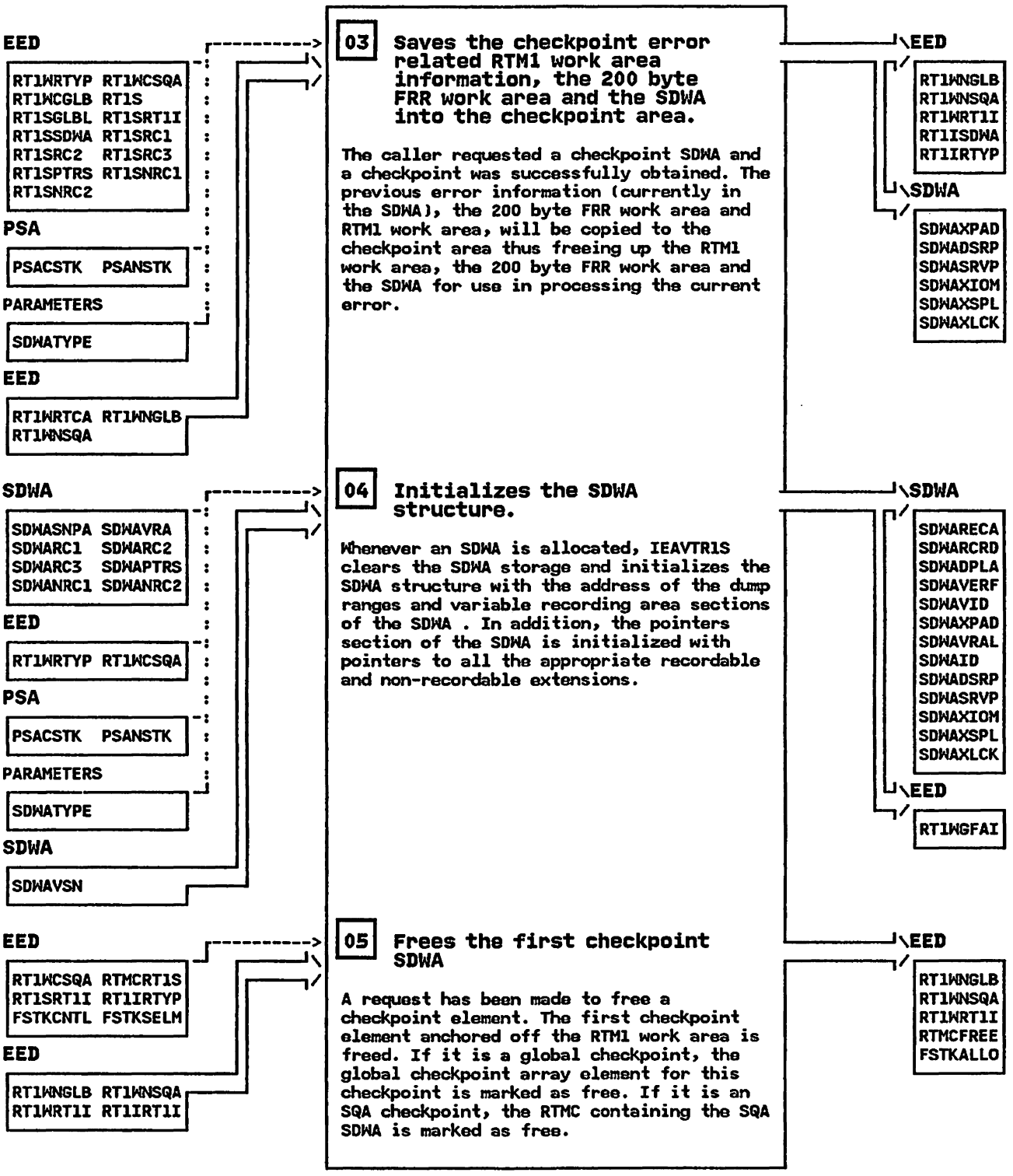
IEAVTR1S - RTM1 SDWA Allocation Module

STEP 01



IEAVTR1S - RTM1 SDWA Allocation Module

STEP 03



IEAVTR1X - MODULE DESCRIPTION

DESCRIPTIVE NAME: RTM1 CMSET Interface Module

FUNCTION:
This module establishes the cross memory addressing environment required by the FRR. The following options on the SETFRR macro control the environment: MODE=PRIMARY, MODE=FULLXM, MODE=LOCAL and MODE=GLOBAL.

ENTRY POINT: IEAVTR1X

PURPOSE: See function.

LINKAGE: BALR

CALLERS: IEAVTRTS

INPUT: Current FRR entry from the FRR stack

OUTPUT: Cross memory environment of the FRR

EXIT NORMAL: Returns to the caller.

EXIT ERROR: Returns to the caller.

EXTERNAL REFERENCES:

ROUTINES:
R1XFRR - FRR Recovery Routine for this module. R1XFRR is an entry point in module IEAVTRRR.

DATA AREAS: No data areas are used.

CONTROL BLOCKS:

Common name	Macro ID	Usage	Function
ASCB	IHAASCB	read	Obtains the ASID of the locally locked address space.
CVT	CVT	read	Obtains various system control block addresses.
FRRS	IHAFRRS	read	Obtains various FRR status information.
LCCA	IHALCCA	read	Obtains address of CPU work save area.
PSA	IHAPSA	read and write	Obtains addresses of various FRR stacks, the ASCB and LCCA. Also used by SETFRR expansion.
RT1W	IHART1W	read and write	Obtains and sets RTM control information.
SDWA	IHASDWA	read and write	Obtains control information and sets FRR status indicators.
SVT	IHASVT	read	Used by CMSET macro expansion
TCB	IKJTCB	read	Obtains the TCB key.
WSAVT	IHAWSAVT	read	Obtains RTM work save area address.

TABLES: No tables used.

SERIALIZATION:
IEAVTR1X does not obtain any locks. IEAVTR1X runs disabled to serialize the RTM1 work area.

IEAVTR1X - MODULE OPERATION

IEAVTR1X determines if special processing is required to establish the entry environment for the current FRR.

Special processing is required if MODE=PRIMARY or MODE=FULLXM was specified on the SETFRR macro that was issued to establish the current FRR, if RTM is processing a CALLRTM TYPE=RMGRCML macro instruction, or if RTM is processing a nested FRR that was established by a MODE=LOCAL or MODE=GLOBAL SETFRR macro. If any one of of these conditions is true, IEAVTR1X does the following processing:

- Obtains the cross memory control information saved by the SETFRR macro from the FRR stack extension area. This information will be used to establish the cross memory environment for the FRR.
- Establishes an FRR environment, R1XFRR, to protect RTM1 against errors.
- If required, issues an CMSET instruction and sets the S-bit on in the PSW.
- If the CMSET instruction failed or was not necessary, the following CMSET recovery processing is performed to establish either a:
  A) MODE=LOCAL environment for the FRR -
     This is done if either the MODE=LOCAL option was specified on the SETFRR macro and a local lock is held or if a subsequent FRR is established by a local resource manager.
     IEAVTR1X indicates in the SDWA that the FRR is receiving control as a local resource manager.
  or
  B) MODE=GLOBAL environment for the FRR -
     This is done if the MODE=GLOBAL option was specified on the SETFRR macro and a global spin lock is held, if the FRR will run disabled, if the FRR is on the super stack, if at least one super bit is set, or if a subsequent FRR is established by the MODE=GLOBAL resource manager.
     IEAVTR1X indicates in the SDWA that the FRR is receiving control as a global resource manager.
- Deletes IEAVTR1X's FRR environment.

If no special processing is needed to establish the FRR's environment, the FRR will run in the current environment. IEAVTR1X sets the key mask value in control register 3 to either the TCB key (if in task mode) or to the time of error key (if in SRB mode or if the error was a DAT error.)

RECOVERY OPERATION:

RTM1 default recovery processing (contained in module IEAVTRTR) protects some of IEAVTR1X's processing against errors. IEAVTR1X uses an FRR to protect against most errors during its processing.

IEAVTR1X - DIAGNOSTIC AIDS

ENTRY POINT NAME: IFAVTR1X

MESSAGES: None

ABEND CODES: None

WAIT STATE CODES: None

RETURN CODES: None

REGISTER CONTENTS ON ENTRY:

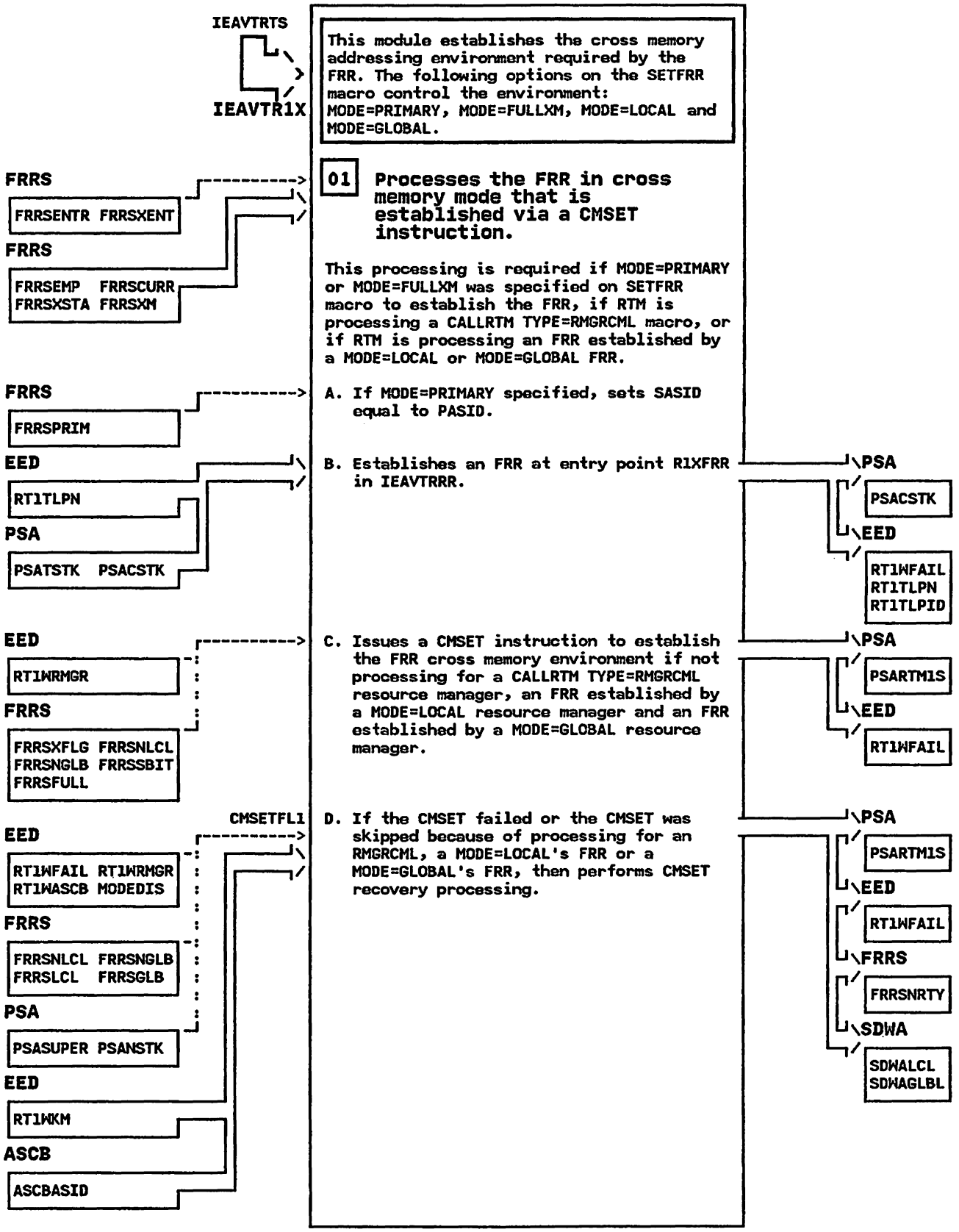
Registers 0-12 - Irrelevant
Register 13 - Register save area address
Register 14 - Return address
Register 15 - Entry point address

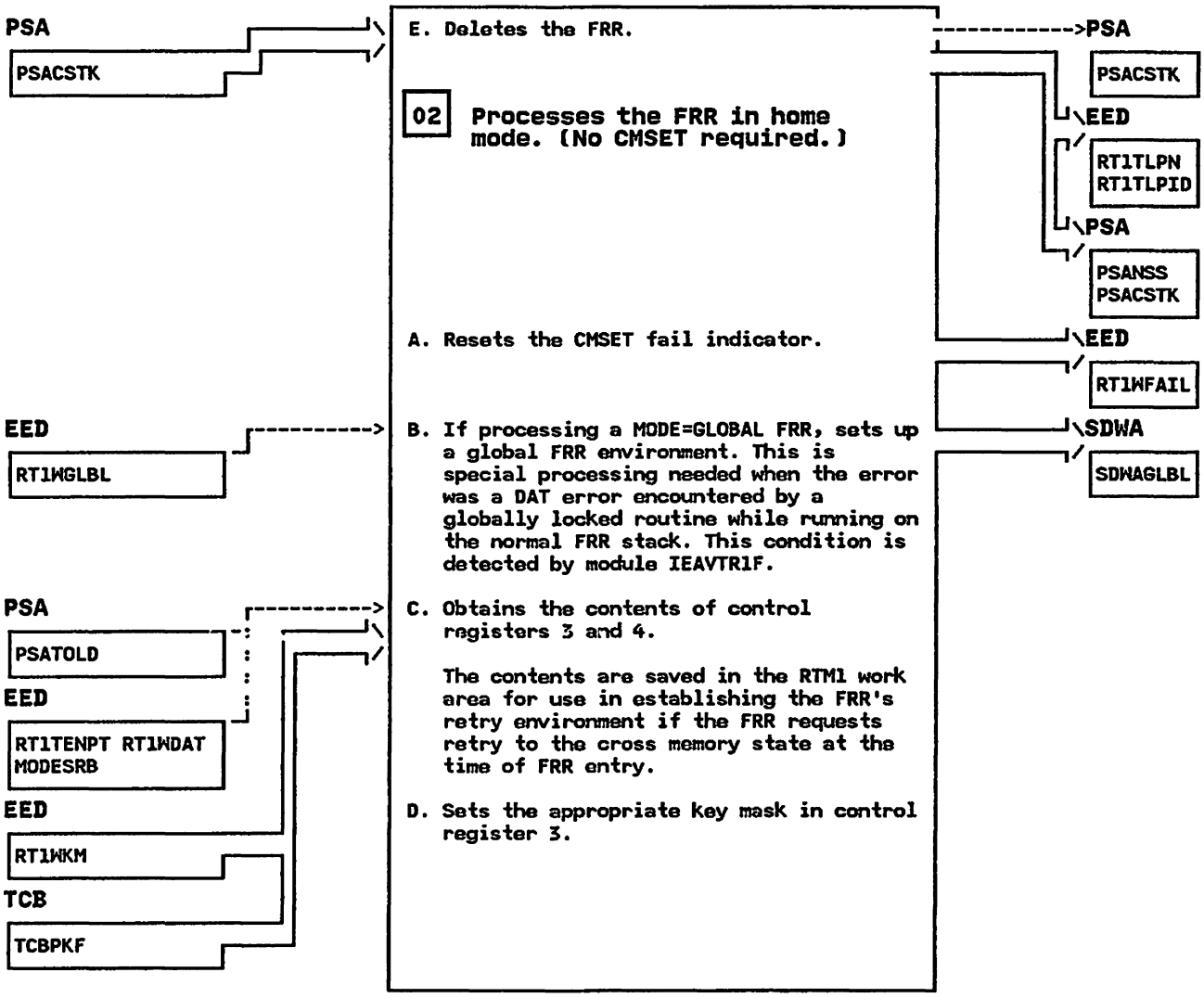
REGISTER CONTENTS ON EXIT:

Registers 1-15 - Restored

IEAVTR1X - RTM1 CMSET Interface Module

STEP 01





IEAVTR10 - MODULE DESCRIPTION

DESCRIPTIVE NAME: RTM Mainline SLIH Mode Processing

FUNCTION:
RTM1 is the primary interface between supervisory routines that detect errors (for example, the first level interrupt handlers for program checks, machine checks, restart interrupts or any other supervisor routines that detect error situations) and the recovery routines that protect a supervisory path from errors. RTM1 also schedules SVC 13 (RTM2) to inform tasks of an error detected by the supervisor and to recover and terminate the task or memory as required.

ENTRY POINT: IEAVTR10

PURPOSE: See function.

LINKAGE: BALR

CALLERS: IEAVTRT1

INPUT:
The current FRR stack and its RTM1 work area, the abnormal termination reason code (saved in RT1WCRC by IEAVTRT1), and various other information contained in registers.

OUTPUT: None

EXIT NORMAL: IEAVTRTM

EXIT ERROR: There are no exit error conditions.

EXTERNAL REFERENCES:

ROUTINES:
- ESTAECHK - Entry point in the SCB Resource Manager module, IEAVTSBP
- FREEDCEL - Entry point in the RTM Recovery Routines module, IEAVTRTR
- FREESPI - Entry point in the RTM Recovery Routines module, IEAVTRTR
- IEAVESAR - Supervisor Analysis Router Module via CALL
- IEAVTRTM - Mainline CALLRTM Service Processor via CALL
- IEAVTRTS - RTM1 FRR Processing Module via CALL
- IEAVTR1S - RTM1 SDWA Allocation Module via CALL
- RECVRRTM - Entry point in the RTM Recovery Routines module, IEAVTRTR

CONTROL BLOCKS:

Common name	Macro ID	Usage	Function
ASCB	IHAASCB	read and write	Obtains the ASID of the current address space, sets the type 1 SVC indicator, updates the count of active CPUs and updates the count of active TCBs.
CSD	IHACSD	read	Obtains the mask of on-line CPUs.
CVT	CVT	read	Obtains various system control block addresses.
FRRS	IHAFRRS	read	Obtains various FRR status information.
LCCA	IHALCCA	read	Obtains address of CPU work save area.
PCCAT	IHAPCCAT	read	Obtains PCCA address for an on-line CPU.
PSA	IHAPSA	read	Obtains addresses of various

IEAVTR10 - MODULE DESCRIPTION (Continued)

			FRR stacks, the ASCB and LCCA. Also used by SETFRR expansion.
RB	IHARB	read	Obtains PSW information from RB.
RTIW	IHARTIW	read and write	Obtains and sets RTM control information.
SDWA	IHASDWA	read and write	Obtains control information and sets FRR status indicators.
SPI	IHASPI	read	Obtains information about the error encountered by the SRB percolating to the current task.
SVT	IHASVT	read	Used by CMSET macro expansion.
TCB	IKJTCB	read	Obtains the TCB protect key.
WSAVT	IHAWSAVT	read	Obtains RTM work save area.

IEAVTR10 - MODULE OPERATION

IEAVTR10 receives control as the primary interface between supervisory routines that detect errors and the recovery routines that protect a supervisory path from errros. IEAVTR10 performs the following processing:

- If RTM is entered recursively for the same recovery environment, calls the logical phase recovery routine RECVRRTM, which is an entry point in IEAVTRTR. Recursion is determined by checking the logical phase number in the RTM1WA of the current FRR stack. Some recursion is considered valid and does not result in a call to the logical phase recovery routine.

- Determines if system level recovery is required. If system level recovery is required, IEAVTR10 calls IEAVTRTS to process the FRR stack. IEAVTRTS is the main router for FRR environment recovery processing. Control returns to IEAVTR10.

- Determines whether to perform percolation or to support retry or resume processing as indicated by the FRR.

  If all FRRs indicated that percolation should occur or if the system state was such that FRRs were not processed, percolates to RTM2 for task level ((E)STAE) recovery. If an SRB is percolating and has a related task, RTM1 will force the related task to abnormally terminate. All information about the error encountered by the SRB is passed to the task's recovery routines. Recovery processing is complete if the error was caused by an SRB that had no related task.

  If final processing is needed to support retry or resume processing from an FRR, establishes the cross memory addressing environment of the retry or resume routine, reloads the retry or resume processing registers, invokes the retry or resume processing entry point, and adjusts the RTM control structure if the retry or resume processing is from a nested FRR.

- If a task's FRR attempts to retry, performs SRB-to-task serial percolation processing. The task will be abnormally terminated if an SRB related to the task encountered an error and caused serialized percolation to the task. The first serial percolation information (SPI) element anchored off the TCB is dequeued and the SRBs error information contained in it is presented to the current unit of work.

RECOVERY OPERATION:

RTM1 default recovery processing (contained in module IEAVTRTR) protects most of IEAVTR10's processing.

IEAVTR10 - DIAGNOSTIC AIDS

ENTRY POINT NAME: IEAVTR10

MESSAGES: None

ABEND CODES:

An ABEND code of X'071' with a reason code of X'0C' is issued when an attempt to resume a unit of work fails.

WAIT STATE CODES: None

RETURN CODES: None

REGISTER CONTENTS ON ENTRY:

Register	0 - Function code (e.g., 1 for a program check)
Register	1 - Abnormal termination completion code and options
Register	2 - Address of the error PSW
Register	3 - Address of the ILC and interrupt code or 0
Register	4 - Irrelevant
Register	5 - Address of Dump options or 0
Register	6 - Address of EEDs or 0
Registers 7 - 12	- Irrelevant
Register	13 - Address of error registers
Register	14 - Return address
Register	15 - Entry point address

REGISTER CONTENTS ON EXIT:

Following is the register content when exiting via a call to IEAVTRTM:

Register	0 - Function code (e.g., 11 for an ABTERM request)
Register	1 - Abnormal termination completion code and options
Register	2 - Irrelevant
Register	3 - Address of the task control block (TCB) to be terminated or 0
Register	4 - Irrelevant
Register	5 - Address of Dump options or 0
Register	6 - Address of EEDs or 0
Registers 7 - 12	- Irrelevant
Register	13 - Address of the error registers
Register	14 - Return address into IEAVTR10
Register	15 - Entry point address of IEAVTRTM

Following is the register content when exiting to the FRRs retry routine:

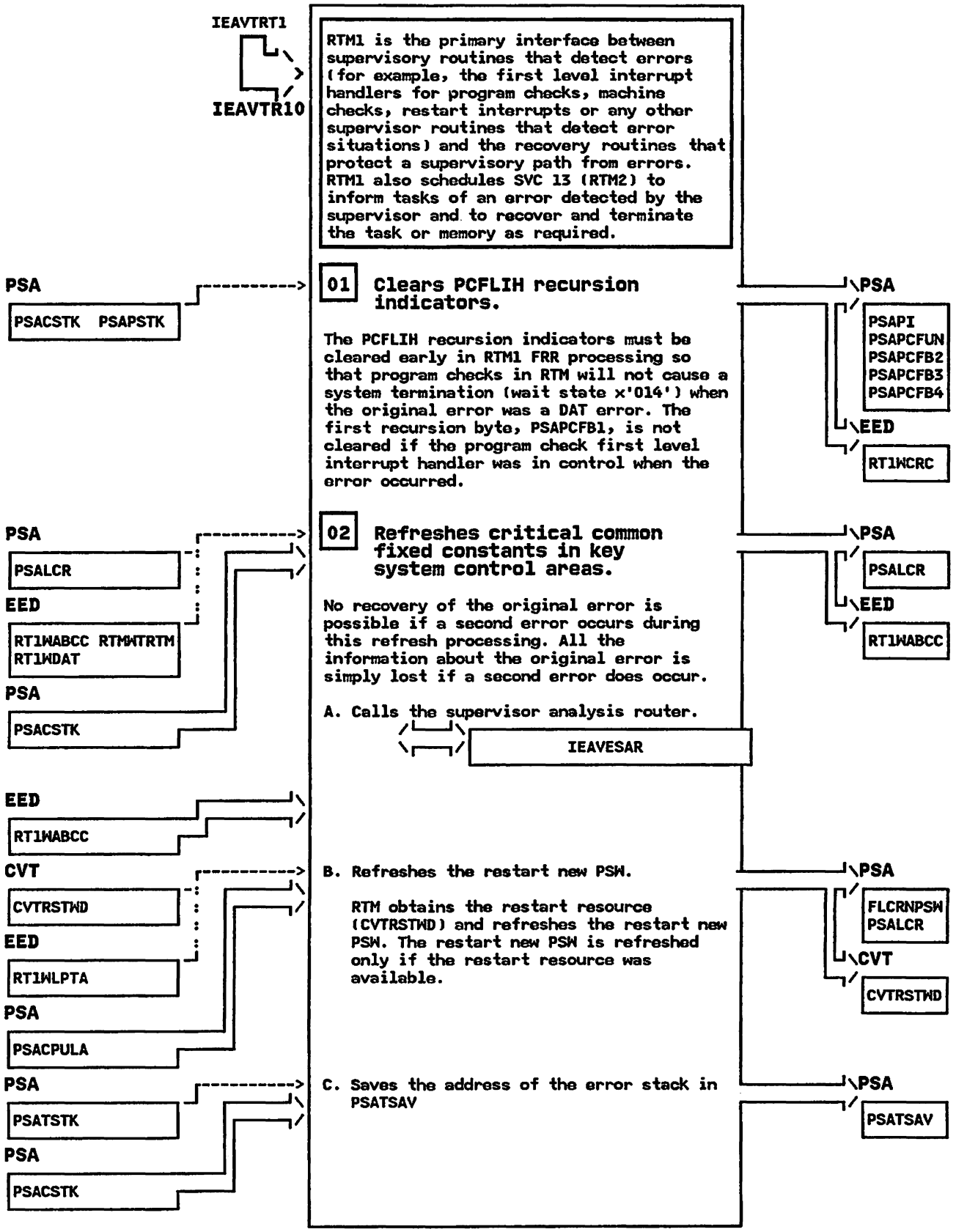
Registers 0 - 14	- Same content as the corresponding register positions in the SDWASRSV register area of the SDWA
Register	15 - Either the entry point address of the FRR retry routine or the register 15 value from the SDWASRSV register area of the SDWA if the RETRY15 option of the SETRP macro

IEAVTR10 - DIAGNOSTIC AIDS (Continued)

was specified.

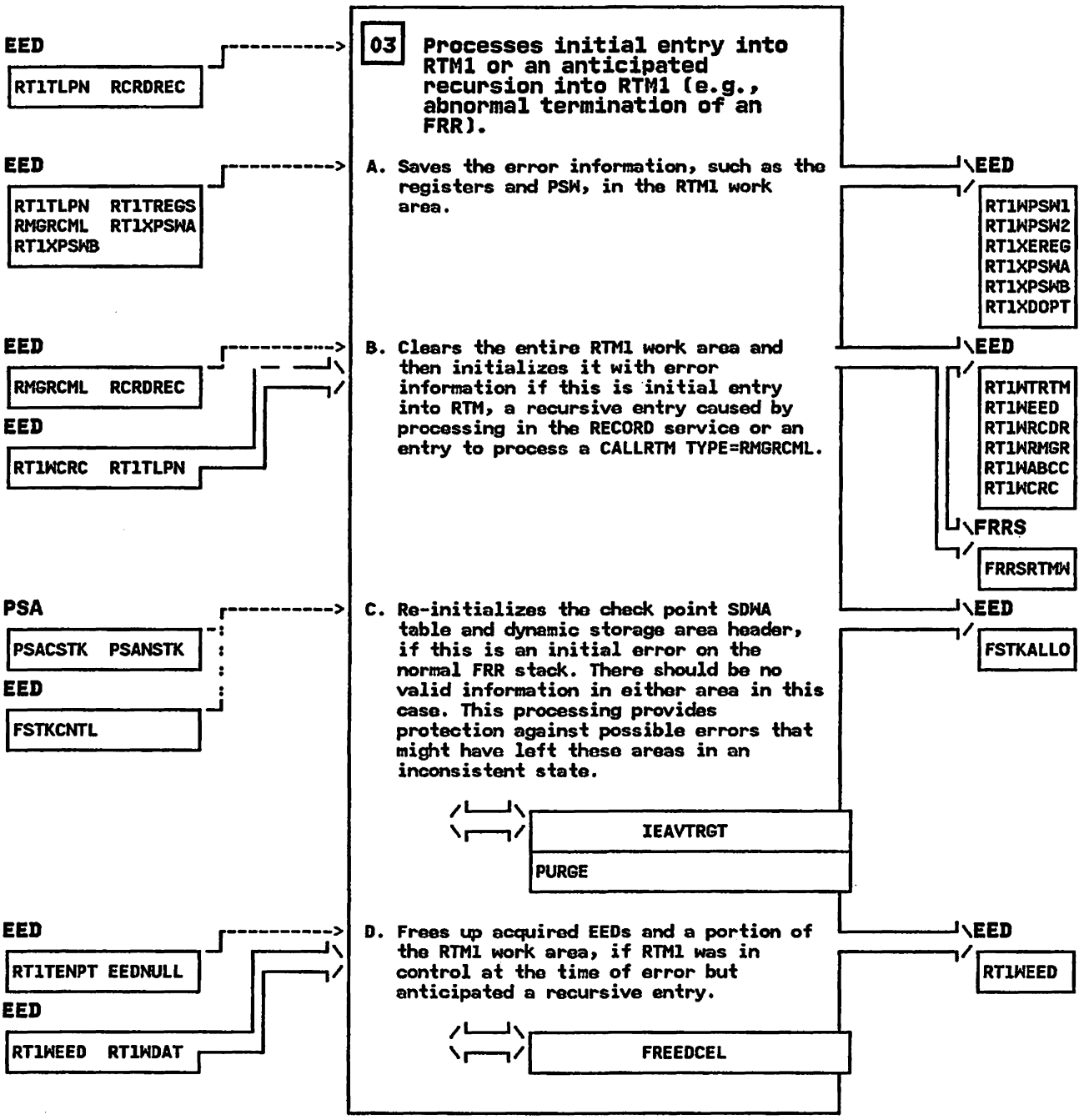
Following is the register content when exiting to the resume point:

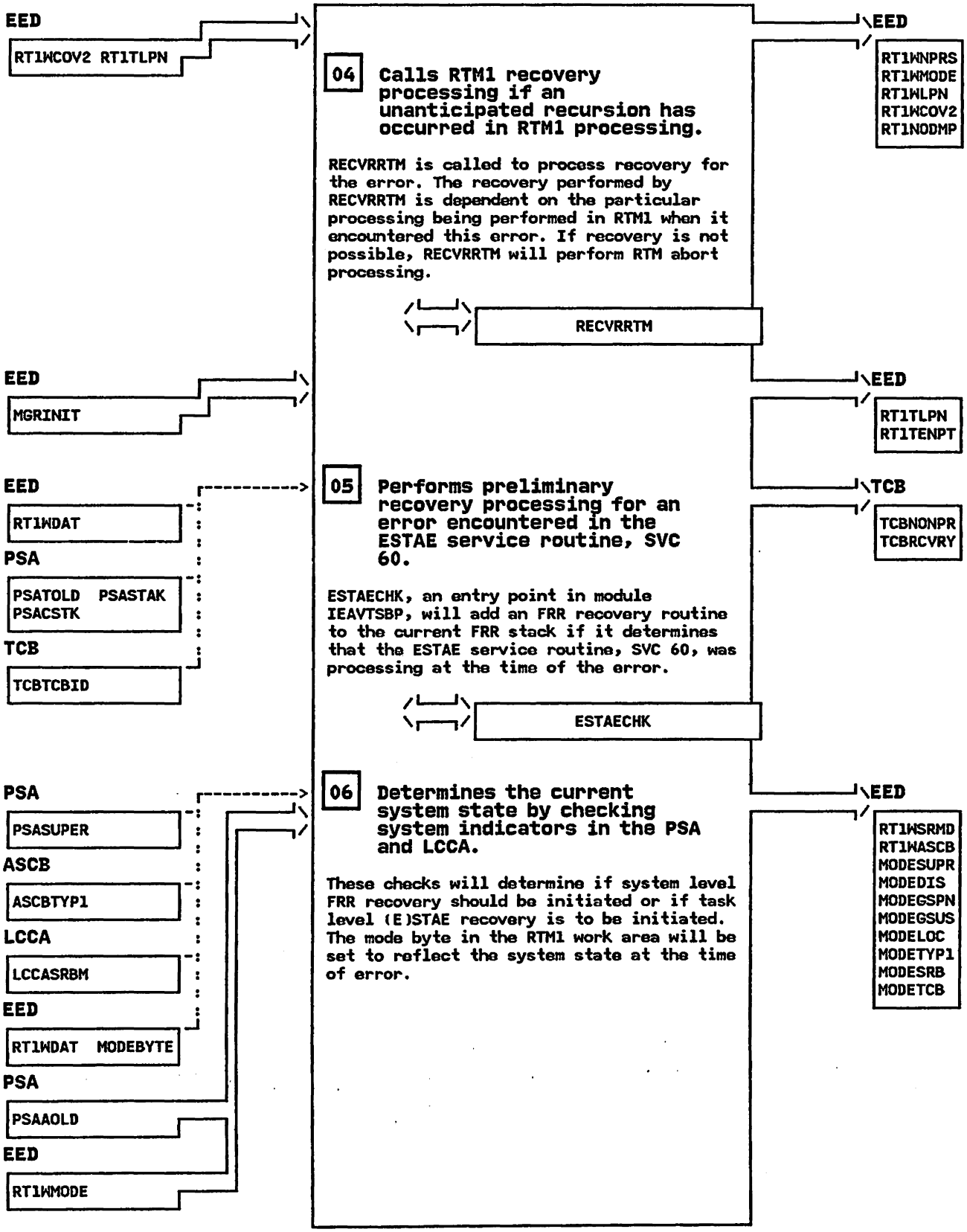
Registers 0 - 15 - Same content as when the system restart was initiated. Obtained from the restart FLIH save area, LCCARSGR.



IEAVTR10 - RTM Mainline SLIH Mode Processing

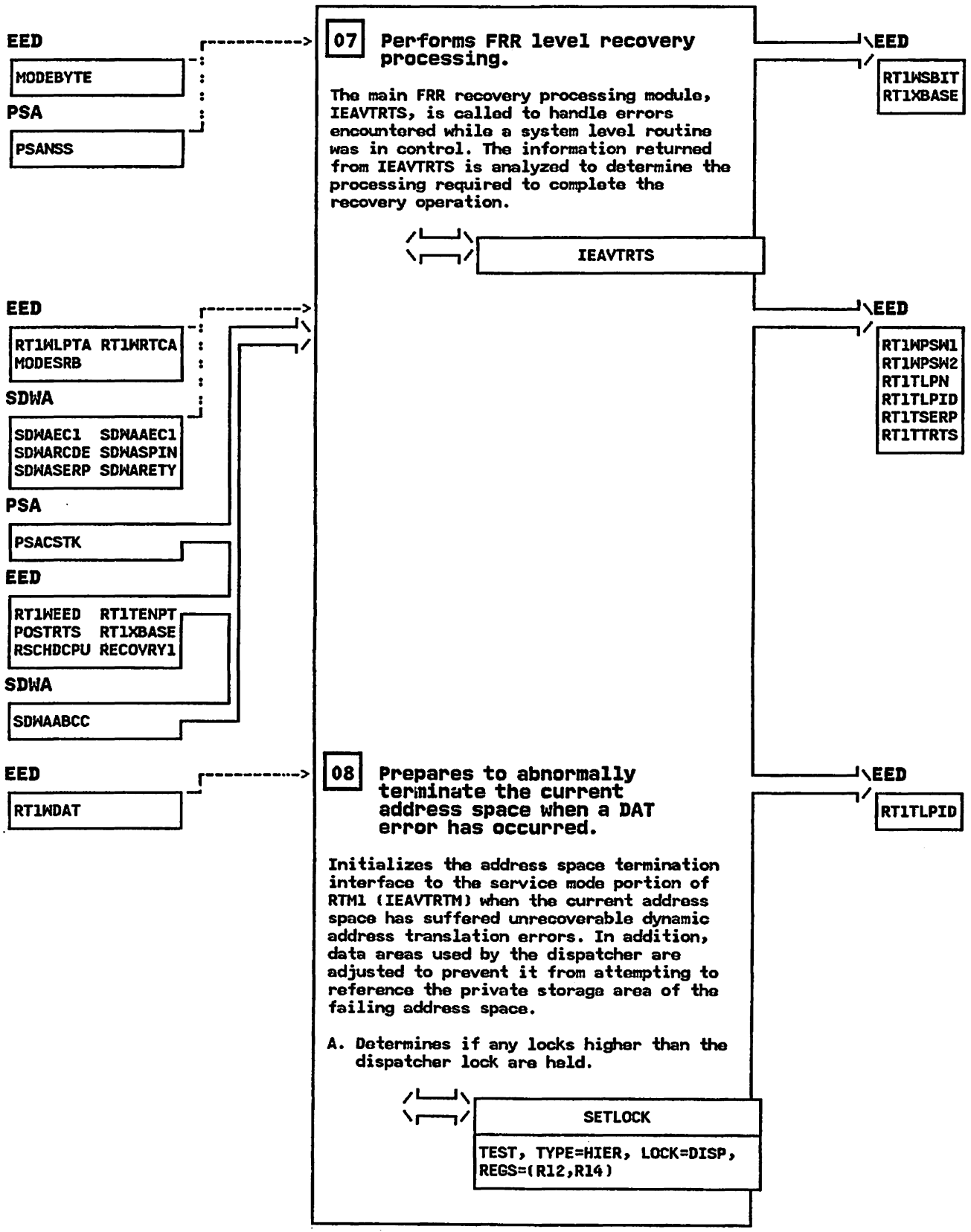
STEP 03

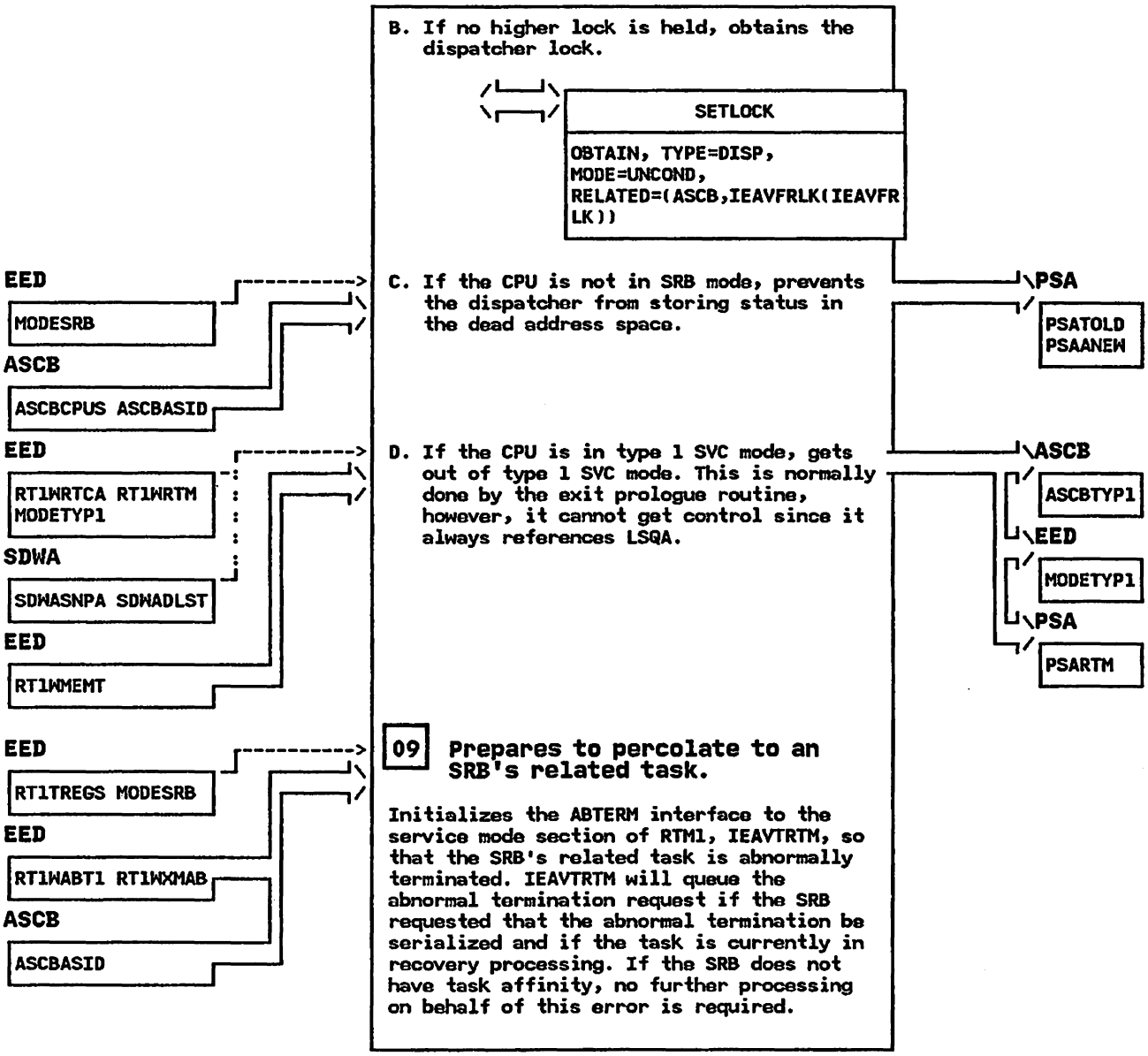




IEAVTR10 - RTM Mainline SLIH Mode Processing

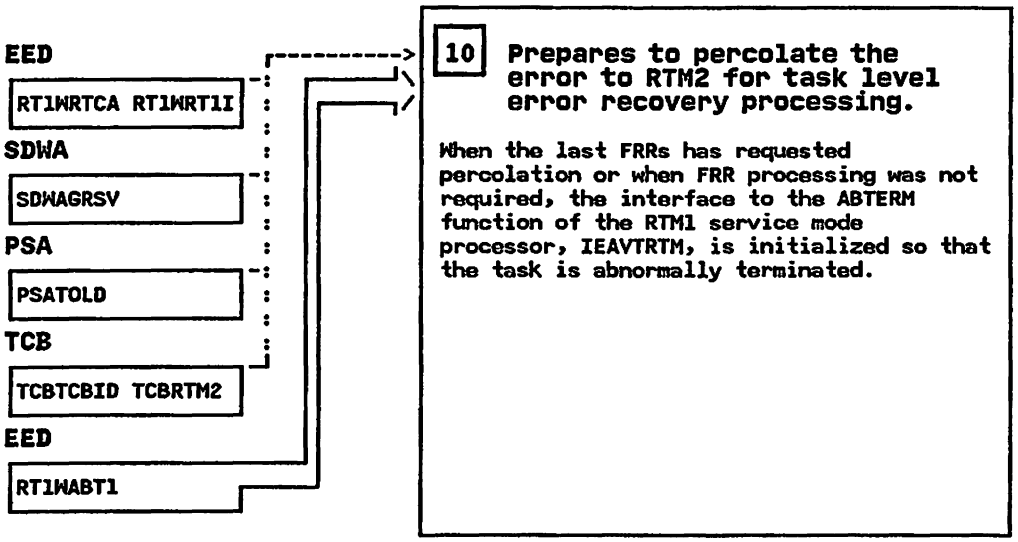
STEP 07

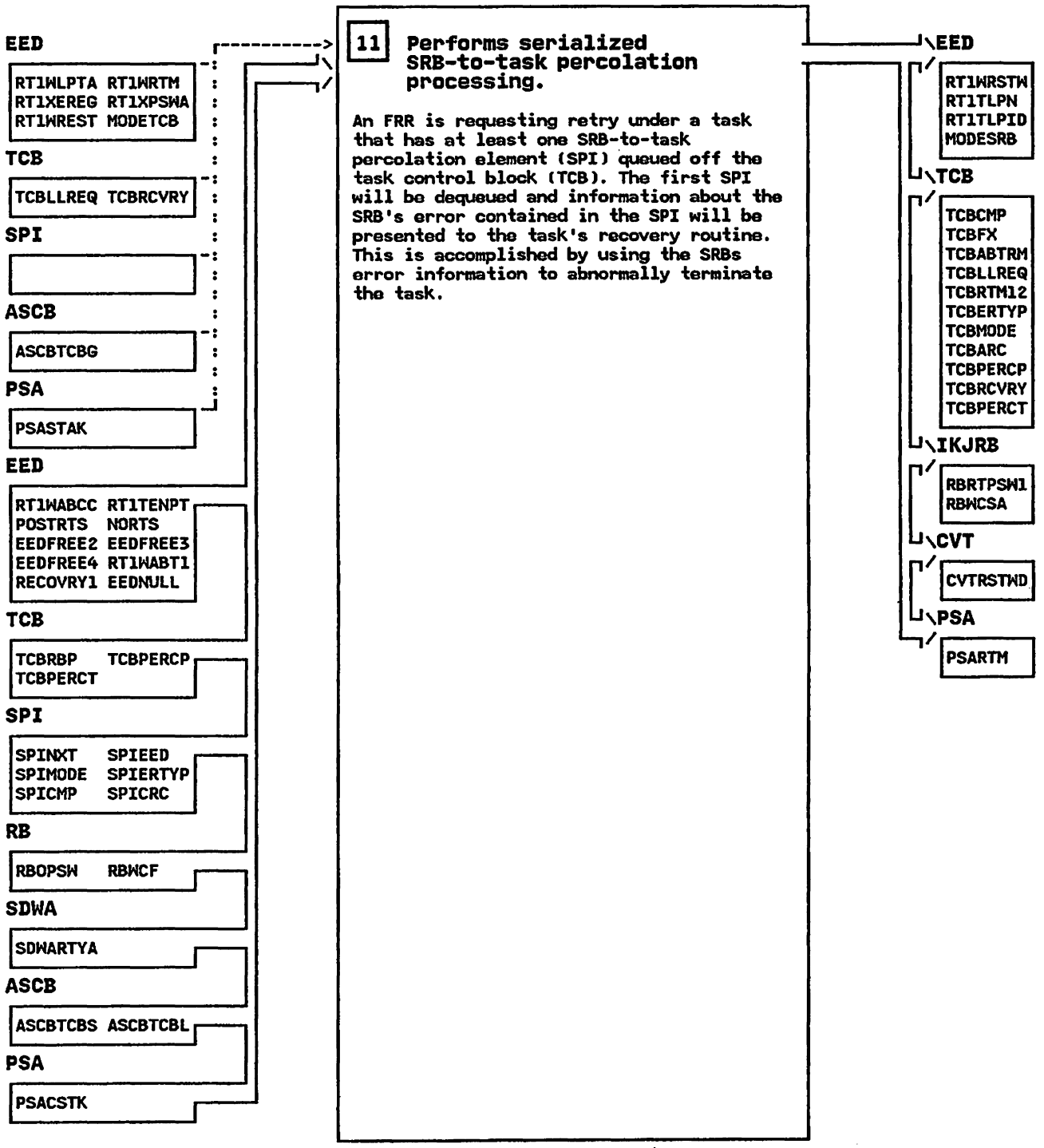


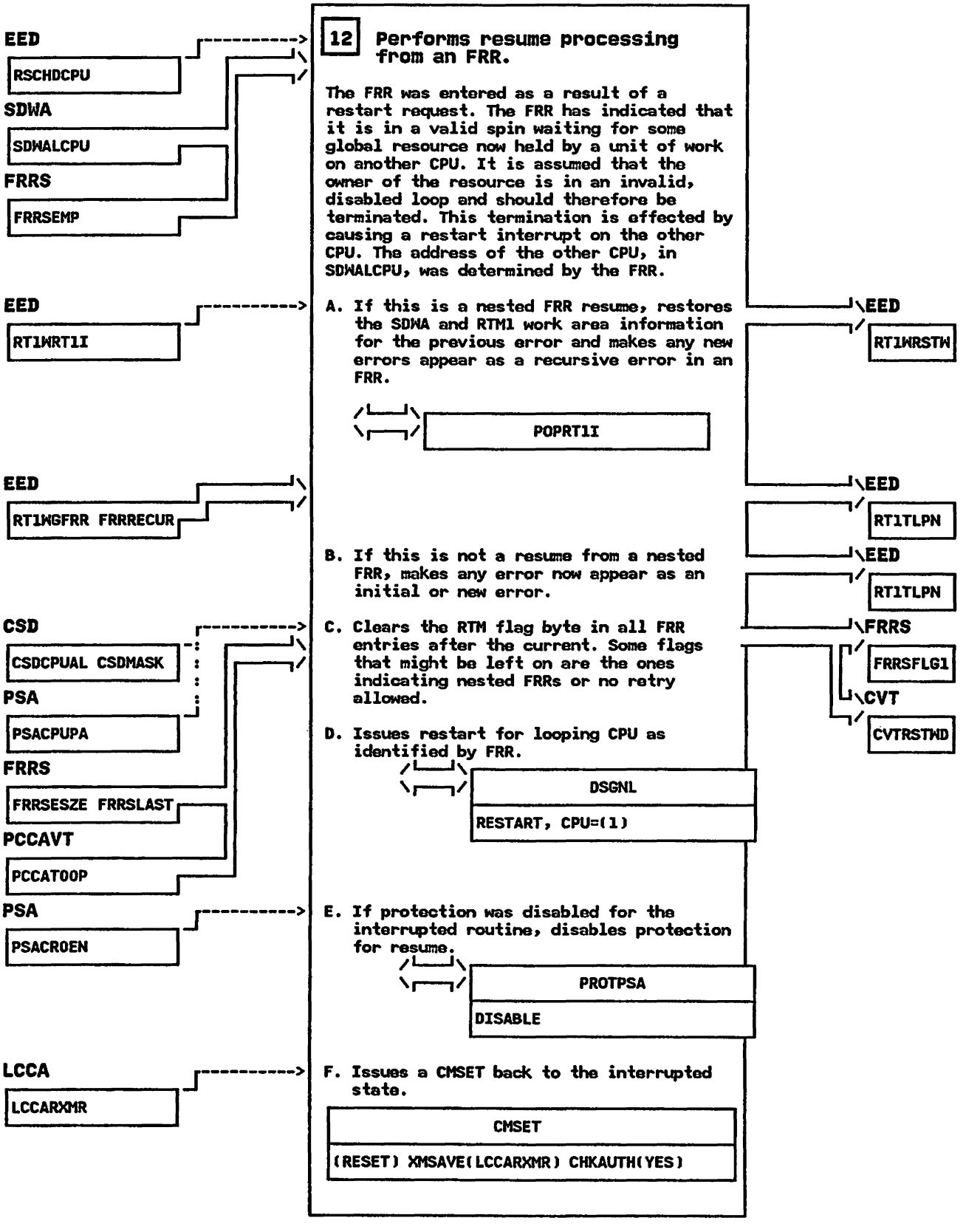


IEAVTR10 - RTM Mainline SLIH Mode Processing

STEP 10

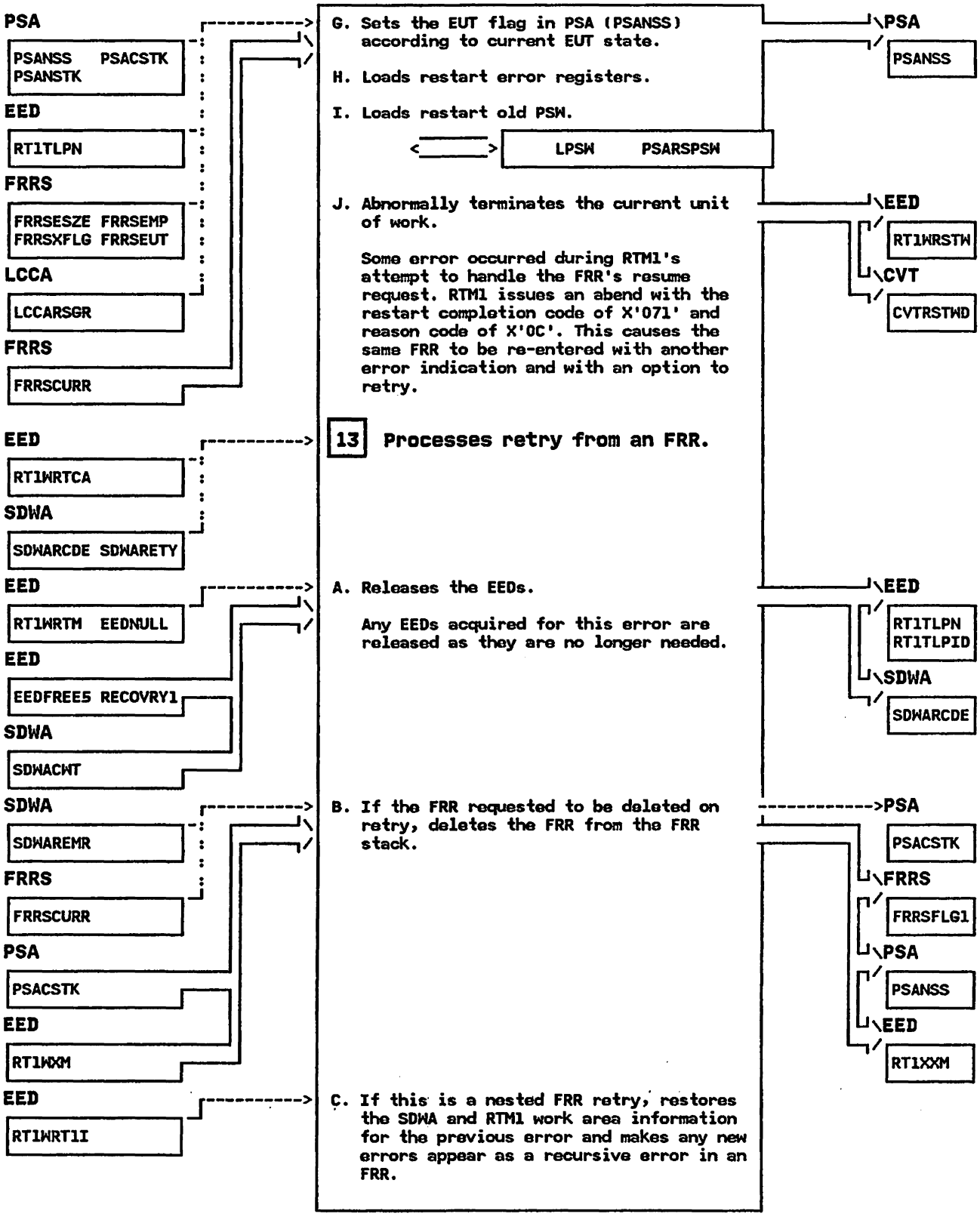






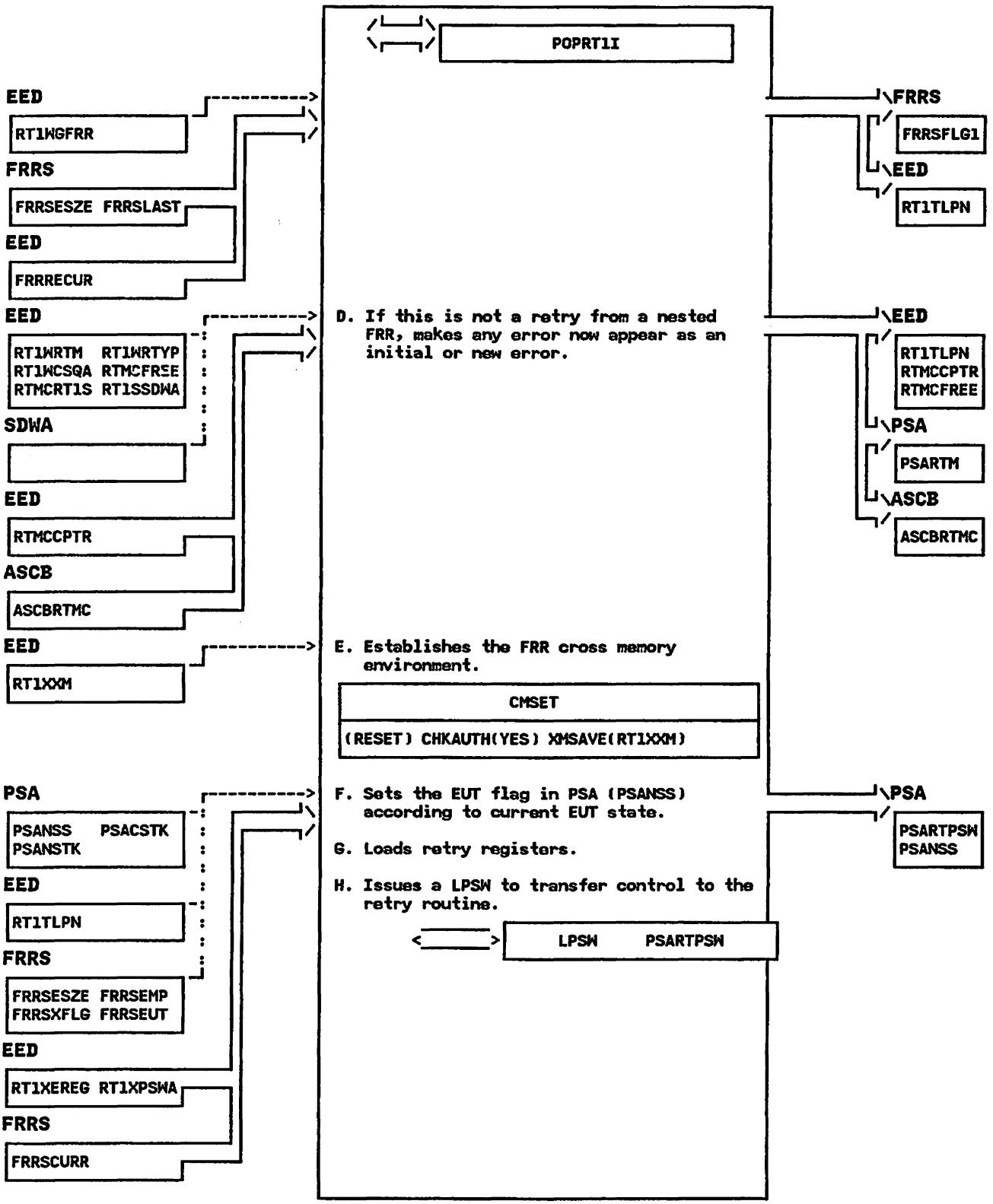
IEAVTR10 - RTM Mainline SLIH Mode Processing

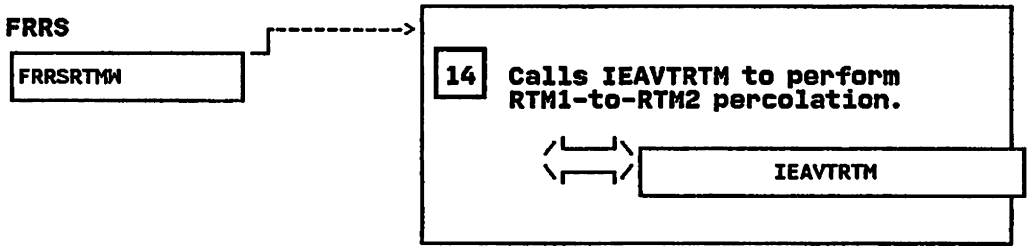
STEP 12G



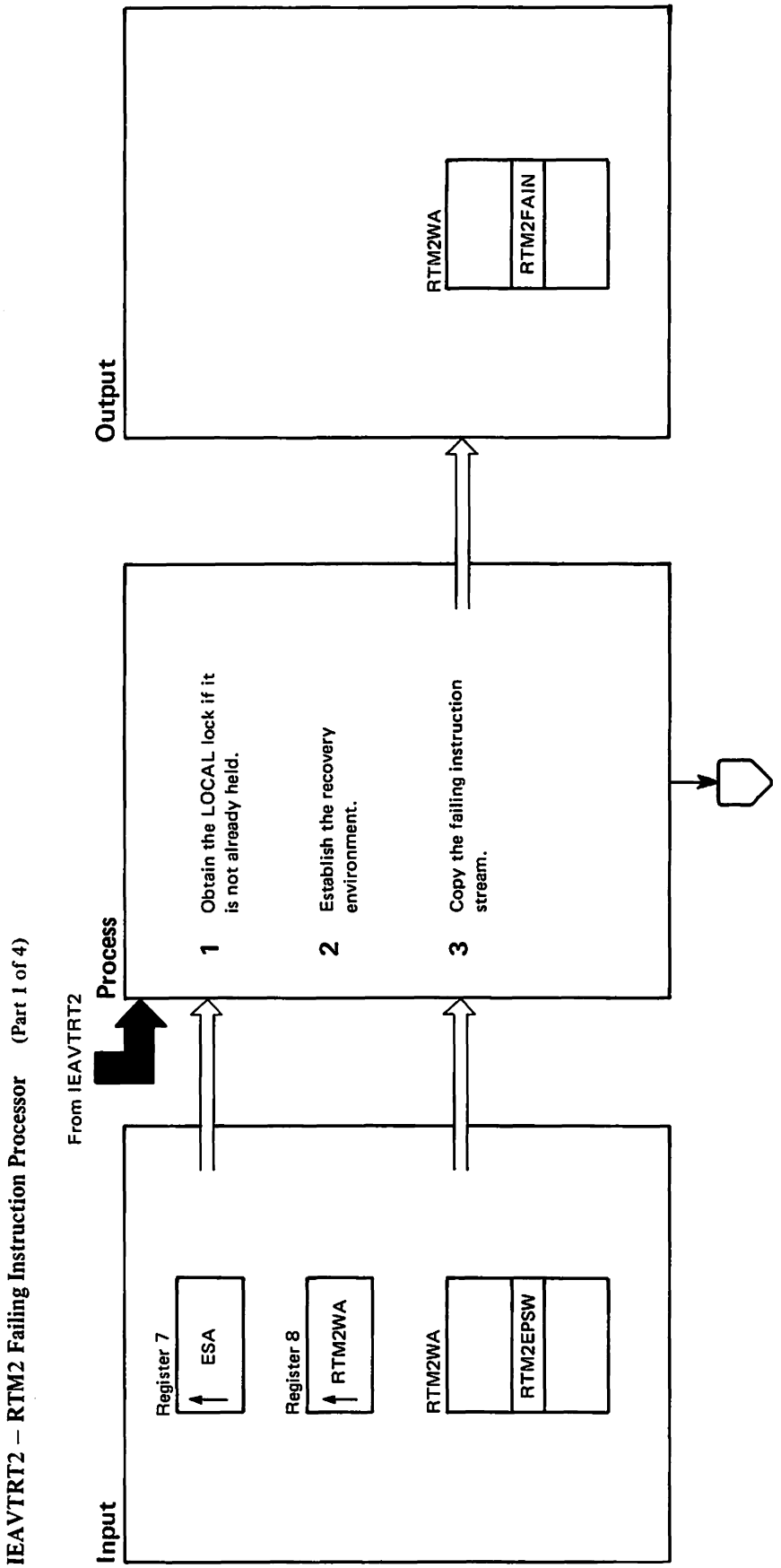
IEAVTR10 - RTM Mainline SLIH Mode Processing

STEP 13D





IEAVTR2A - RTM2 FAILING INSTRUCTION PROCESSOR



IEAVTRT2 – RTM2 Failing Instruction Processor (Part 1 of 4)

IEAVTRT2 – RTM2 Failing Instruction Processor (Part 2 of 4)

Extended Description | **Module** | **Label**

The RTM2 function provides additional information regarding an error in RTM2WA. This additional information consists of twelve bytes of instruction stream around the instruction counter (IC) in the failing PSW. During RTM2's initialization processing (see M.O. diagram IEAVTRT2 – RTM2 Initialization), IEAVTR2A obtains the six bytes of storage that precede and the six bytes of storage that follow the IC in the failing PSW and puts them in the RTM2FAIN field. RTM2 propagates this instruction stream from ESTAE (extended specify task abnormal exit) to ESTAE during recovery.

1 IEAVTR2A must hold the LOCAL lock to serialize functions within the home address space.

2 To protect against possible errors when referencing the failing instruction stream, IEAVTR2A establishes an FRR environment. (See recovery processing at the end of this diagram).

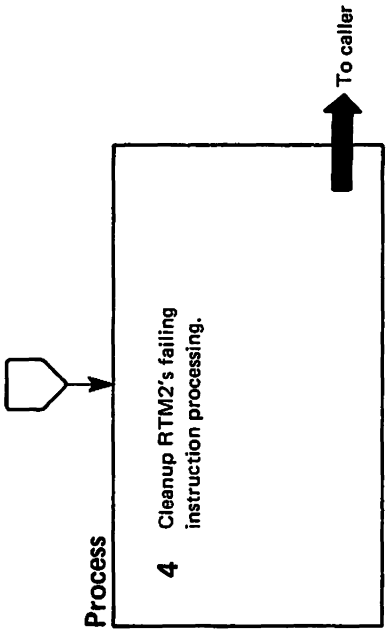
3 IEAVTR2A reads the failing instruction stream in the user's PSW key and copies it to the RTM2FAIN field. The number of moves required to copy the failing instruction stream depends upon the location of the IC (PSWIC).

- If the PSWIC is not within five bytes of a page or storage boundary, IEAVTR2A copies the twelve bytes in one move.
- If the PSWIC is within five bytes of location 0 or the last addressable storage location, IEAVTR2A copies the less than twelve bytes in one move.
- If the PSWIC is within five bytes of the start or end of a page, IEAVTR2A copies the twelve bytes in two moves.

If the failing instruction stream is not accessible on the first and only move, then IEAVTR2A places hexadecimal zeroes in the RTM2FAIN field. RTM2 requests a retry. (See recovery processing).

If the failing instruction stream is not accessible on the first of a two-part move, then IEAVTR2A attempts the second move and the RTM2FAIN field might be partially filled-in. RTM2 requests a retry for the second move. (See recovery processing).

RETRY1

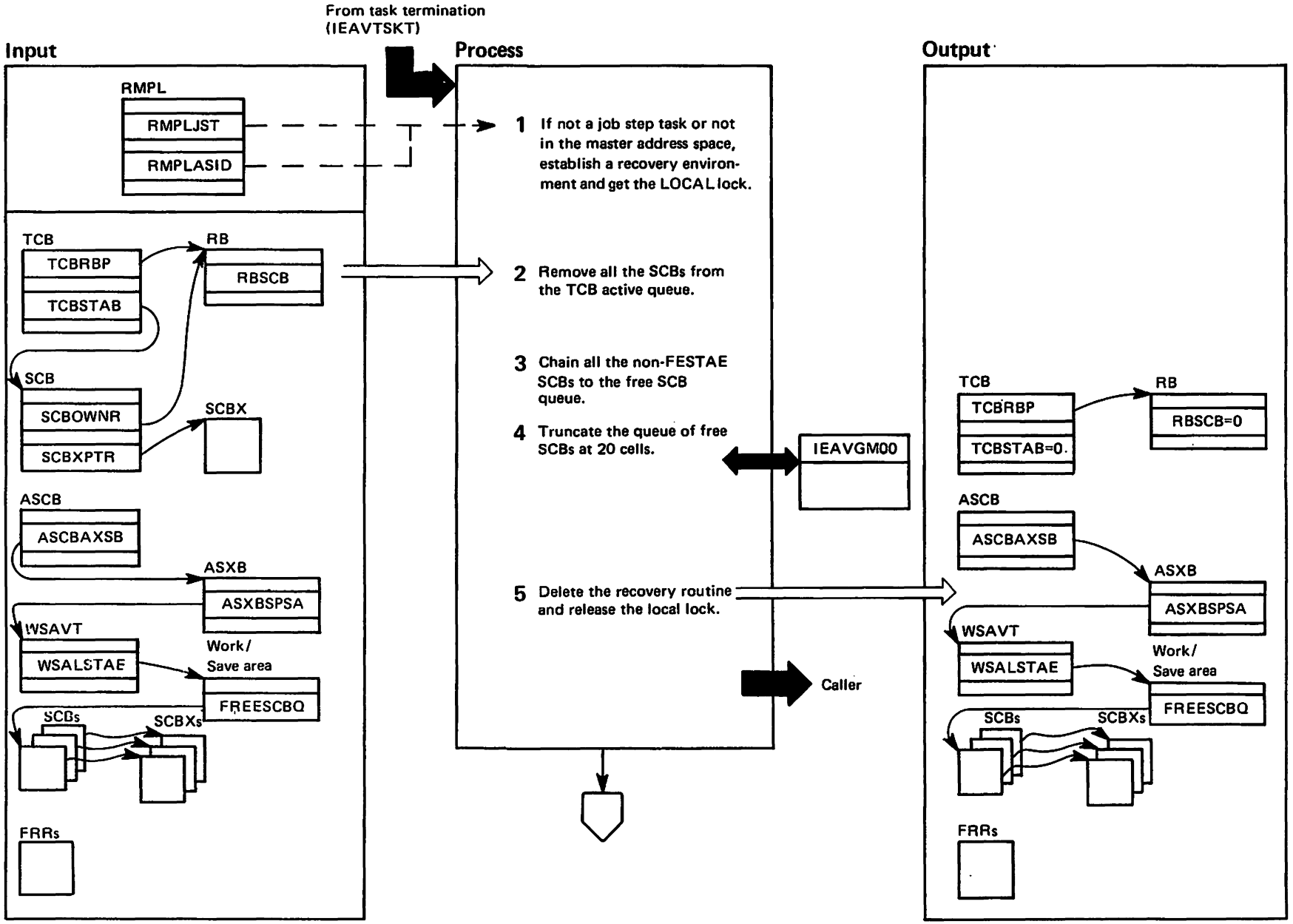


IEAVTRT2 – RTM2 Failing Instruction Processor (Part 3 of 4)

Extended Description	Module	Label
4 IEAVTR2A performs the following cleanup functions: • Issues the SETFRR macro to delete the FRR environment. • Issues the SETLOCK macro to free the LOCAL lock if obtained on entry to IEAVTR2A. IEAVTR2A then returns to its caller.		
Recovery Processing IEAVTR2A attempts to recovery from an interruption via its FRR. The FRR requests a retry from RTM at location RETRY1 if an error occurs while IEAVTR2A is copying the failing instruction stream in the first of a two-part move. RETRY1 will attempt the second move. RTM does not record this error. The FRR requests a retry from RTM at location RETRY2 if: • An error occurs while IEAVTR2A is executing (other than copying the failing instruction stream). RTM records this error on SYS1.LOGREC. • An error occurs while IEAVTR2A is copying the failing instruction stream in the last or only move. RTM does not record this error.		
RETRY2 performs cleanup (see step 4). If a retry is not allowed (SDWACLUP bit is on), the FRR requests percolation. RTM1 records this error on SYS1.LOGREC. After recovery processing, IEAVTR2A returns to RTM1. The following chart summarizes IEAVTR2A's recovery processing.	RETRY2	

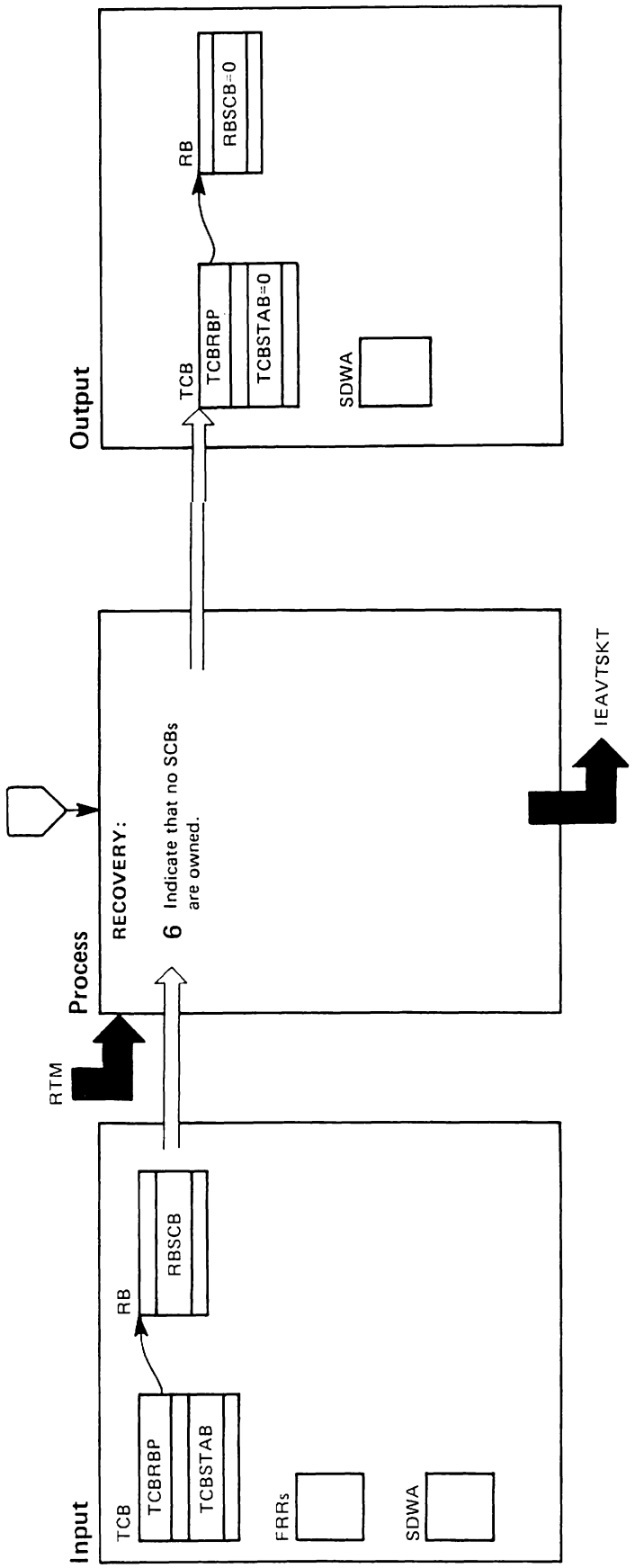
Condition	Retry	Location	Error Recorded on SYS1.LOGREC
Failure to copy the failing instruction stream in a two-part move	Yes	RETRY1	No
IEAVTR2A execution failure	Yes	RETRY2	Yes
Failure to copy the failing instruction stream in a one-part move	Yes	RETRY2	No
SDWACLUP bit is on	(Percolation occurs)	N/A	Yes

IEAVTSCB – SCB FREEMAIN Routine (Part 1 of 4)



IEAVTSCB – SCB FREEMAIN Routine (Part 2 of 4)

Extended Description	Module	Label
The SCB FREEMAIN routine limits the number of cells on the free-SCB-queue and frees any excessive storage. The task termination routine (IEAVTSKT) gives control to IEAVTSCB.		
1 If a non-job step task terminates in an address space other than the master address space, IEAVTSCB obtains the LOCAL lock and sets up the FRR to provide recovery.	IEAVTSCB	LOOPSCB
2 IEAVTSCB removes all the SCBs associated with the terminating task that have not yet been deleted from the TCB's related active chain. This module sets to zero the TCB field pointing to the active SCB chain (TCBSTABB). IEAVTSCB also sets to zero the field in every RB associated with the terminating task that indicates that a SCB is associated with the RB (RBSCB).		
3 IEAVTSCB places the SCBs that needed storage obtained by a GETMAIN on the queue of free SCBs.		REPEATF
4 IEAVTSCB scans the queue of free SCBs and limits the number of SCB cells on the queue to 20. IEAVTSCB returns any no longer needed storage to subpool 255.		LOOP3
5 Before returning control to IEAVTSKT, IEAVTSCB deletes the FRR and frees the LOCAL lock.		ENDPR

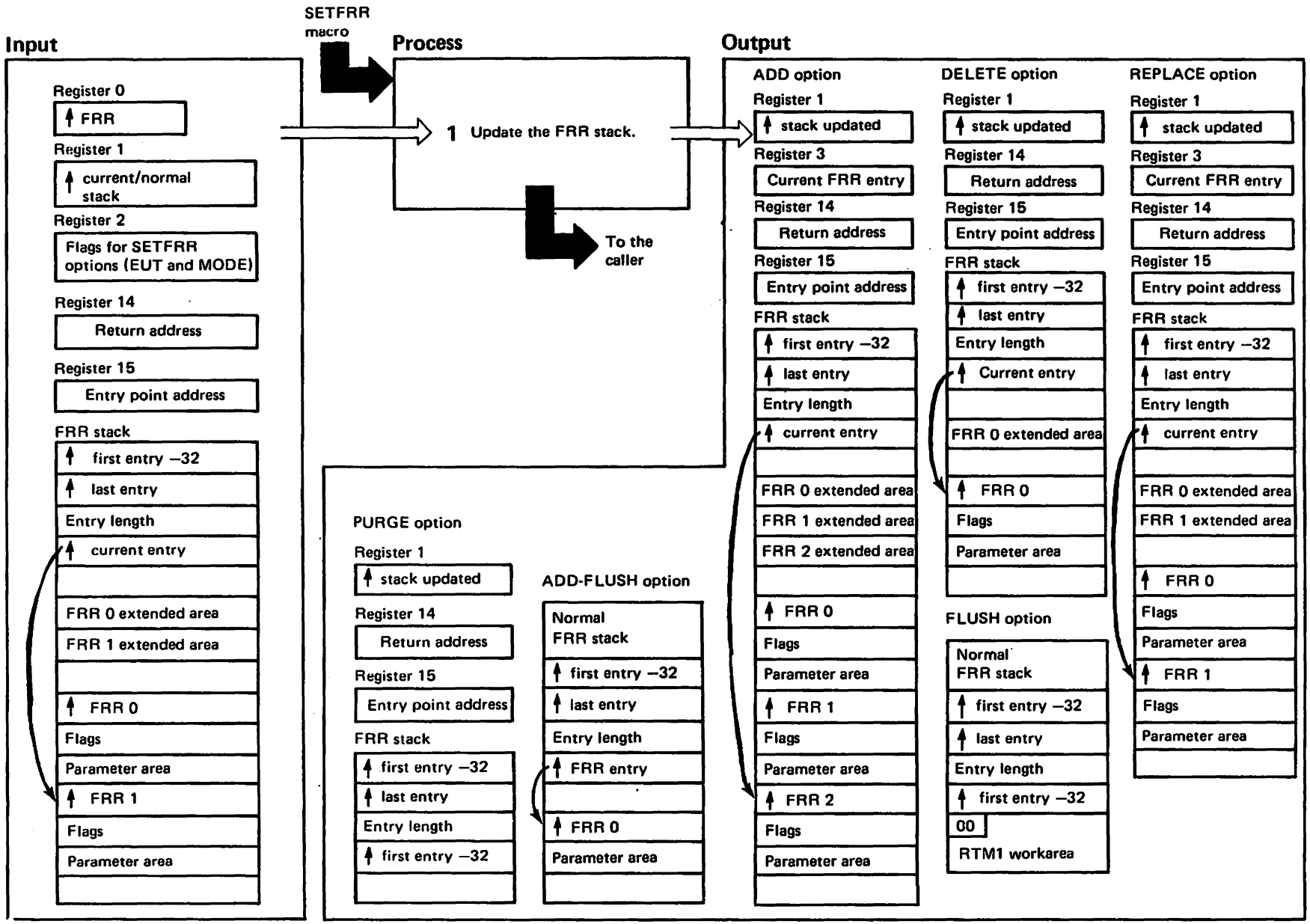


IEAVTSCB – SCB FREEMAIN Routine (Part 3 of 4)

IEAVTSCB – SCB FREEMAIN Routine (Part 4 of 4)

Extended Description	Module	Label
6 If an unexpected error occurs, IEAVTSCB sets to zero the TCB field that points to the active SCB chain and the field in all task-related RBs that indicates an associated SCB. IEAVTSCB deletes the FRR and releases the LOCAL lock.		IEAVFRB

IEAVTSFR – SETFRR (Part 1 of 2)



IEAVTSFR – SETFRR (Part 2 of 2)

Extended Description | **Module** | **Label**

IEAVTSFR alters the contents of an appropriate FRR stack based on the supplied options.

1 The SETFRR macro calls this module to update the FRR stacks. Depending on the request, IEAVTSFR will add an entry to the current stack, delete an entry on the current stack, purge all the entries on the current stack, flush all the entries on the normal stack, or flush all the entries on the normal stack and then add an entry to the normal stack. The six mutually exclusive options do the following:

- **ADD** – The FRR address supplied as input is added to the stack and the current FRR entry pointer is updated to point to this new FRR address. If the caller specified the MODE=PRIMARY or FULLXM parameter on the SETFRR macro, IEAVTSFR saves the cross memory information in the extended area. IEAVTSFR sets flags in the second word of the entry to indicate the other options chosen. If the stack is full, a X'07D' ABEND will occur if the caller requests another FRR be added.
- **REPLACE** – Performs a replacement of the FRR address pointed to by the fourth word of the stack header by the input FRR address. If the caller specified the MODE=PRIMARY or FULLXM parameter on the SETFRR macro, IEAVTSFR saves the cross memory information in the extended area. IEAVTSFR sets flags in the second word of the entry to indicate the other options chosen. If the FRR stack is *empty*, an addition equivalent to A is performed.

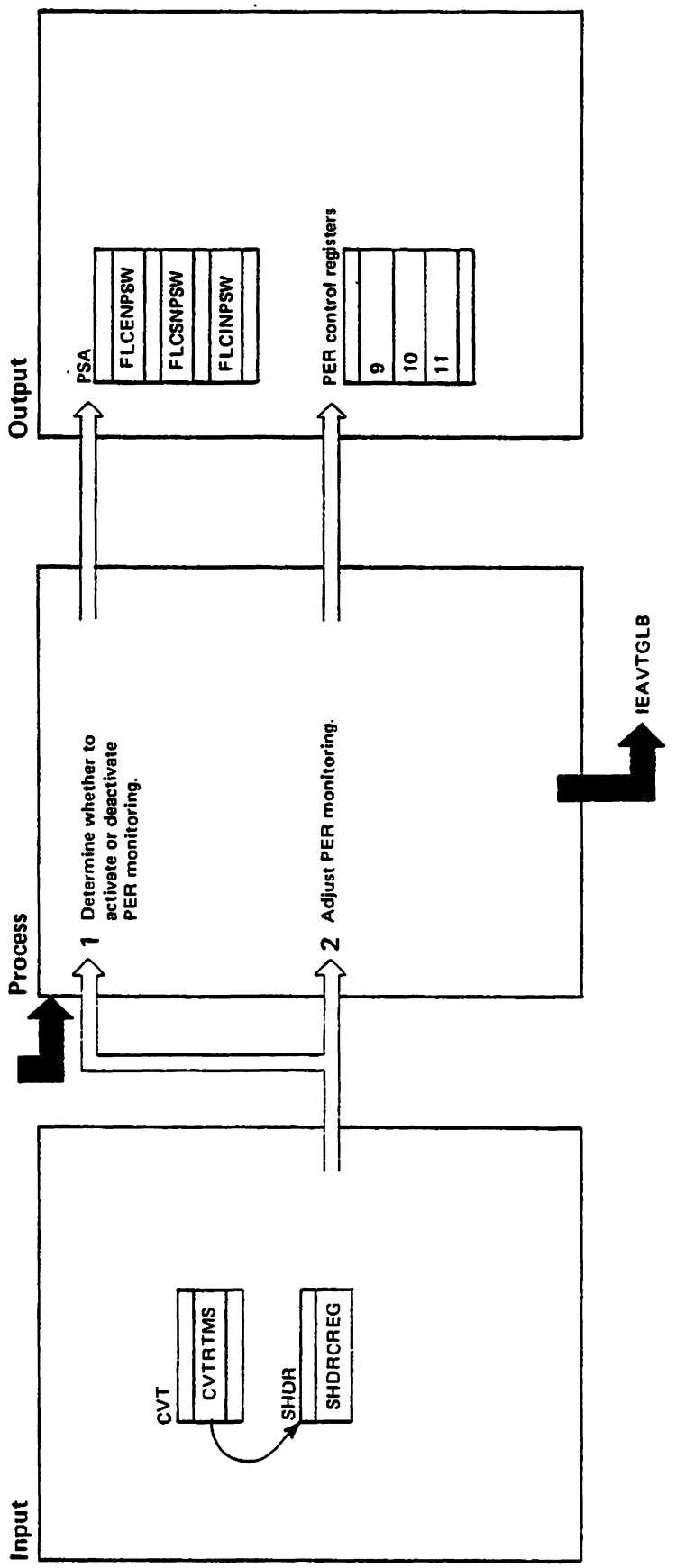
Extended Description | **Module** | **Label**

- **DELETE** – Removes an FRR address from the stack by adjusting the fourth word of the stack header to point to the preceding FRR entry. If the stack is empty this delete function is a NOP.
- **PURGE** – Adjusts the stack header to reflect an empty stack (sets the fourth word equal to the first word of the stack header).
- **FLUSH** – A special option to be used only by the dispatcher, purges the normal FRR stack (making it empty) and zeroes RTM recursion indicators in the RTM1 work area portion of the normal FRR stack.
- ADD-FLUSH – A special option used only by the dispatcher, purges the normal FRR stack and adds the FRR for the SRB to be dispatched.

Note:

Stacks depicted represent normal FRR stacks. Supervisor control FRR stacks have the first word of the header pointing to the first FRR entry rather than the address of the first entry –32.

IEAVTSIG - SLIP PER RISGNL ROUTINE



IEAVTSIG – SLIP PER RISGNL Routine (Part 1 of 2)

IEAVTSIG – SLIP PER RISGNL Routine (Part 2 of 2)

Extended Description **Module** **Label**

This module either activates or deactivates PER monitoring on each active processor in the system.

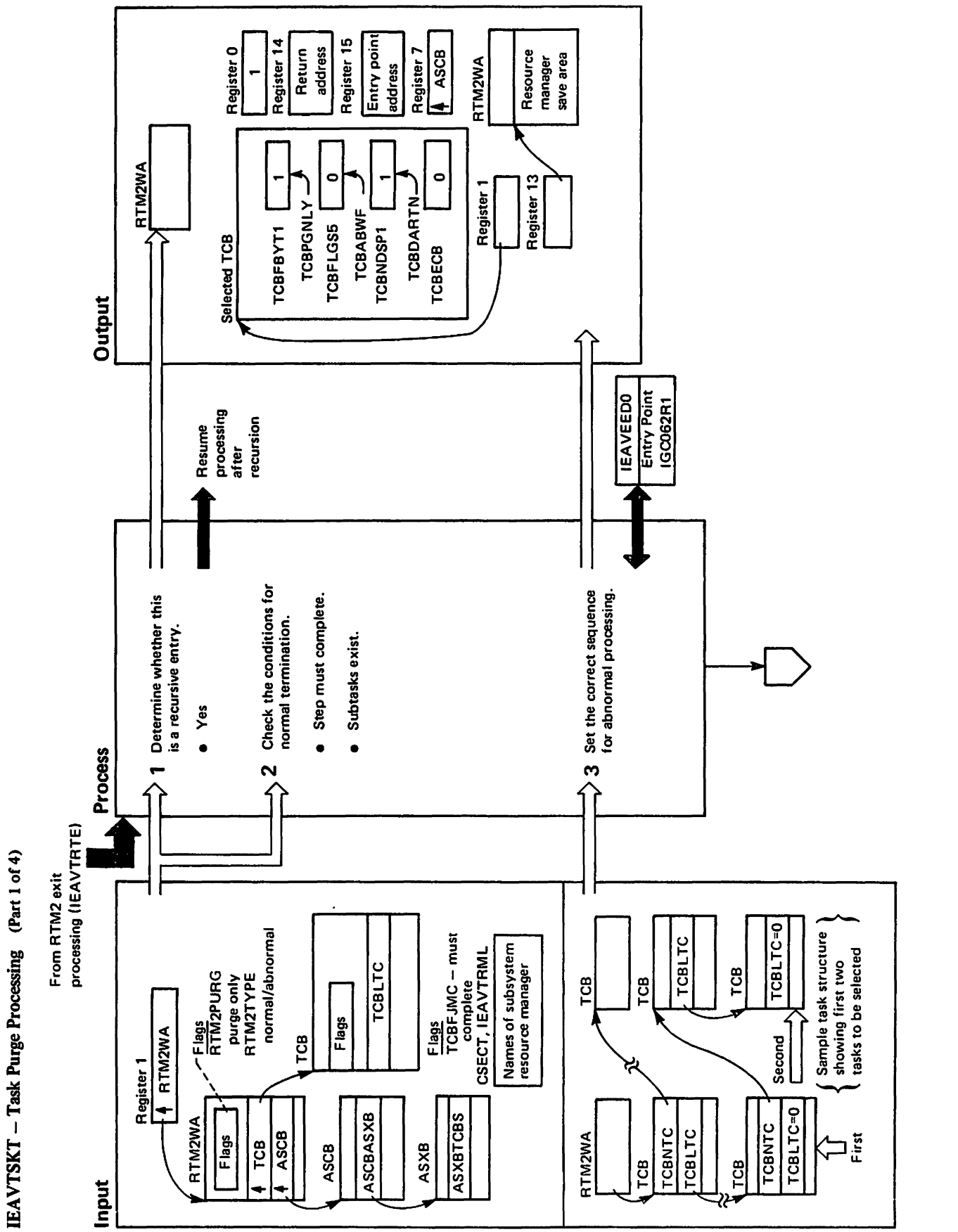
1 The instruction fetch and storage alteration monitoring flags in the SHDR (the SHDRC9IF and SHDRC9SA bits of the SHDRCREG field) indicate whether the caller's request is to turn PER monitoring on or off. If either flag equals one, IEAVTSIG activates PER monitoring by turning on the PER bit in the I/O new PSW, the external new PSW, and the SVC new PSW (FLCINPSW, FLCENPSW, and FLCSNPSW, respectively). Otherwise, IEAVTSIG deactivates PER monitoring by resetting these bits to zero. PSA protection is disabled to allow alteration of the PSW PER bits.

2 IEAVTSIG copies the SHDRCREG field values into PER control registers 9-11.

Recovery processing:

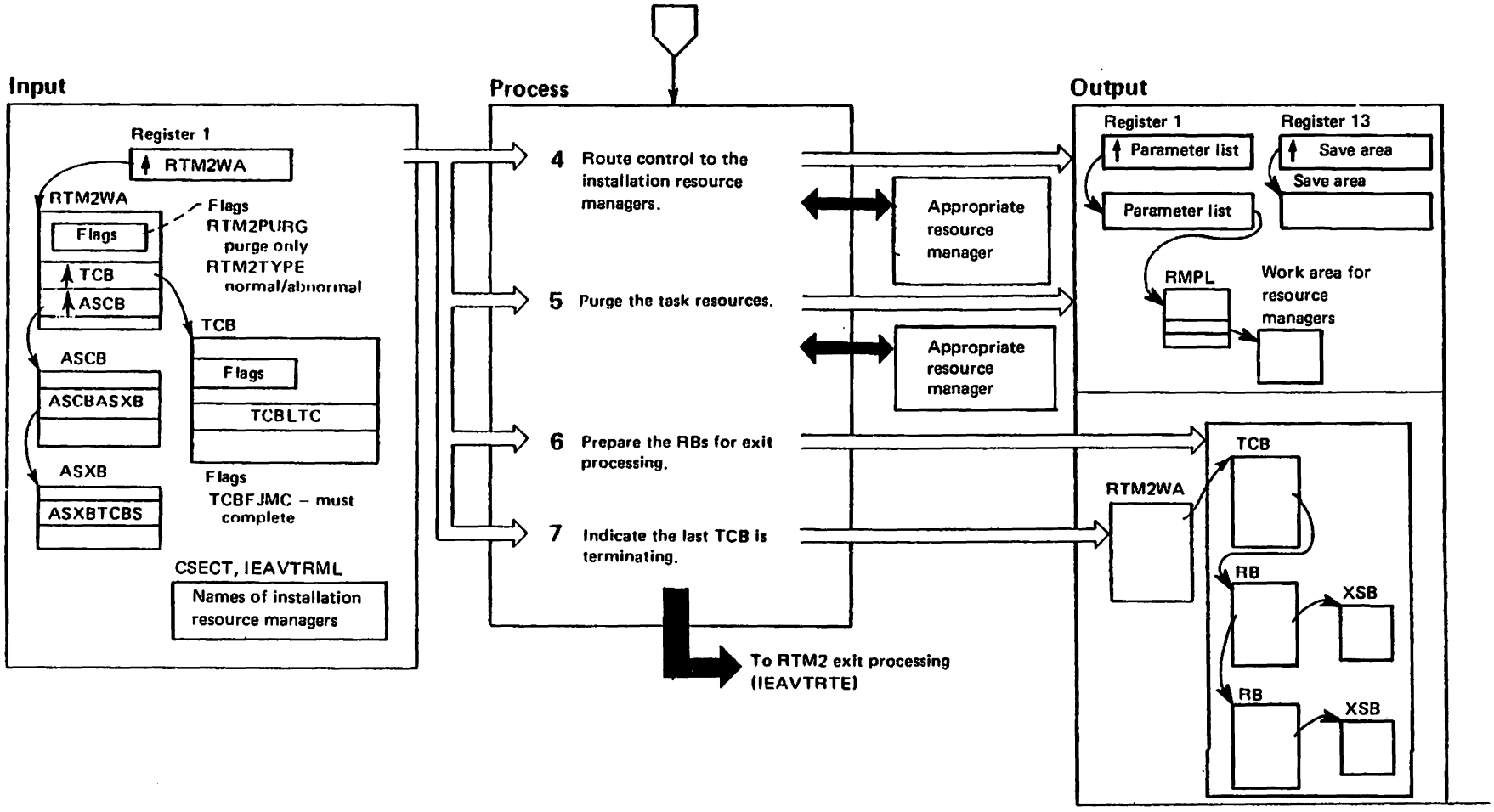
None is established for this module.

IEAVTSKT - TASK PURGE PROCESSING



Extended Description	Module	Label
Task purge processing removes the resources used by a task. RTM2 uses the task purge processing function to route control sequentially to installation-defined and IBM-defined resource manager routines to remove their task-related resources.		
Task purge processing will remove the resources of the lowest task in the TCB family queue first, and then ascend the queue to the current task, removing each task's resources.		
Task purge processing receives control from RTM2 exit processing. Input for task purge processing comes from M.O. diagram IEAVTRT2 – RTM2 Initialization, which shows the creation and initialization of the RTM2WA.		
1 Task purge processing performs recursion processing, as described in M.O. diagram IEAVTRT2 – Recursion Processor 1.	IEAVTSKT	
The RTM2TRRA field contains the addresses of routines that handle recursions for processes in steps 3, 4, and 5.		
• If a CANCEL recursion occurs for step 3, restart step 3 by selecting the lowest task in the family and detaching it. For any other type of recursion, terminate the address space.		
• If a subsystem resource manager fails, skip the failing subsystem resource manager on a recursive entry. If more than 2 failures occur, skip all the subsystem resource managers, and go to step 5.		
• If an IBM-defined resource manager fails, skip it on any recursive entries and continue processing the others.		
2 For a normally terminating task, task purge processing checks the terminating task for step must complete status, for open data sets, and for existing subtasks.		
• For tasks having step must complete status, terminate with a X'E03' ABEND code.		
• If subtasks exist, task purge processing terminates the task being terminated with a X'A03' ABEND code. RTM2 will then regain control as a result of the SVC 13 instruction issued to terminate the task.		
3 The terminating task may have active subtasks. In this case, task purge processing follows down the TCBLTC chain until it finds the lowest TCB (as indicated by a 0 in TCBLTC). Task purge processing then issues a DETACH (see the section "Task Management" for a description of DETACH processing) for that TCB, with an indicator to perform termination purging. DETACH will terminate the task if it is still active. Task purge processing detaches all the subtasks, and then purges the resources for the current task.		

IEAVTSKT – Task Purge Processing (Part 3 of 4)



IEAVTSKT – Task Purge Processing (Part 4 of 4)

Extended Description	Module	Label

4 Task purge processing gives control sequentially to installation-defined resource manager routines so they can free task-related resources. The module IEAVTRML contains the names of the installation routines.

5 Go to the data management resource manager to close all data sets. (See M.O. diagram IEAVTSKT – Task Purge Resource Managers for a description of the task purge resource mangers.) If the data sets cannot be closed for a task terminating normally, terminate the task with a X'C03' ABEND code.

Task purge processing gives control sequentially, in the addressing mode indicated in the address field, to IBM-defined resource manager routines to free task-related resources. For normal termination, these routines are called in the following sequence:

SVC dump	IEAVTSDR
Subsystem interface	IEFJRECF
Data management	IFG0TC0A
Timer	IEAQPGTM
System trace	IEAVETRM
IQE	IEAVEEEP
Type 1 message	IEAVTPMT
SPIE	IEAVSPIE
MSSFCALL SVC	IEAVMFRM
ENQ/DEQ	IEAVENQ2
WTOR	IEAVMED2
Region control task	IEAVAR07
VTAM	ISTRAMAI
TCAM	IEDQOT01
Subsystem interface	IEFJRECM
TIOC	IEDAY8
Virtual fetch	CSVVFMEM
POST	IEARPOST
PCAUTH	IEAVXPAM
Real storage management	IEAVTERM
Timer	IEAQPGTM
IQE	IEAVEEEP
SCB	IEAVTSCB
3850 mass storage system	ICB2AIR
ENQ/DEQ	IEAVENQ2
Type 1 message	IEAVTPMT
SRB purge	IEAVEPD0

Extended Description	Module	Label

For abnormal termination, these routines are called in the following sequence:

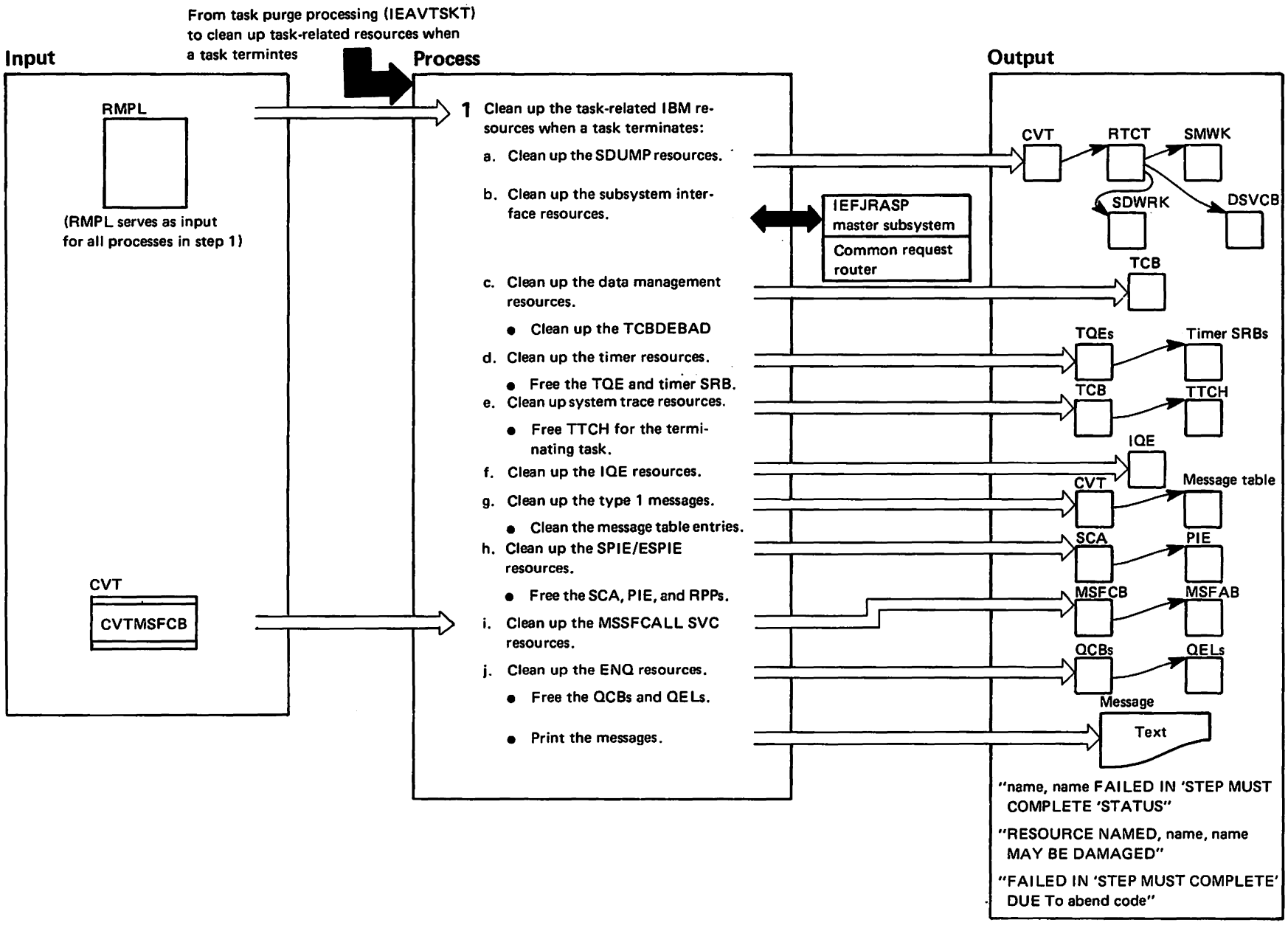
SVC dump	IEAVTSDR
Subsystem support	IEFJRECF
Timer	IEAQPGTM
System trace	IEAVETRM
IQE	IEAVEEEP
Data management	IFG0TC0A
Type 1 message	IEAVTPMT
SPIE	IEAVSPIE
MSSFCALL SVC	IEAVMFRM
ENQ/DEQ	IEAVENQ2
WTOR	IEAVMED2
Region control task	IEAVAR07
VTAM	ISTRAMAI
TCAM	IEDQOT01
Subsystem interface	IEFJRECM
Allocation	IEFAB4E5
TIOC	IEDAY8
Virtual fetch	CSVVFMEM
POST	IEARPOST
PCAUTH	IEAVXPAM
Real storage management	IEAVTERM
IQE	IEAVEEEP
SCB	IEAVTSCB
3850 mass storage system	ICB2AIR
ENQ/DEQ	IEAVENQ2
Type 1 message	IEAVTPMT
SRB purge	IEAVEPD0

These routines free any control blocks related to the task. Control returns from these routines to the task purge processing function.

6 Task purge processing prepares the RBs (request blocks) of the failing tasks to exit by placing the address of an SVC 3 (EXIT) in their RBOPSW field. When these RBs receive control, they will go to EXIT. To ensure running in home mode, IEAVTSKT sets PASID=SASID=HASID in the XSB, and sets the RBOPSW S-bit to zero.

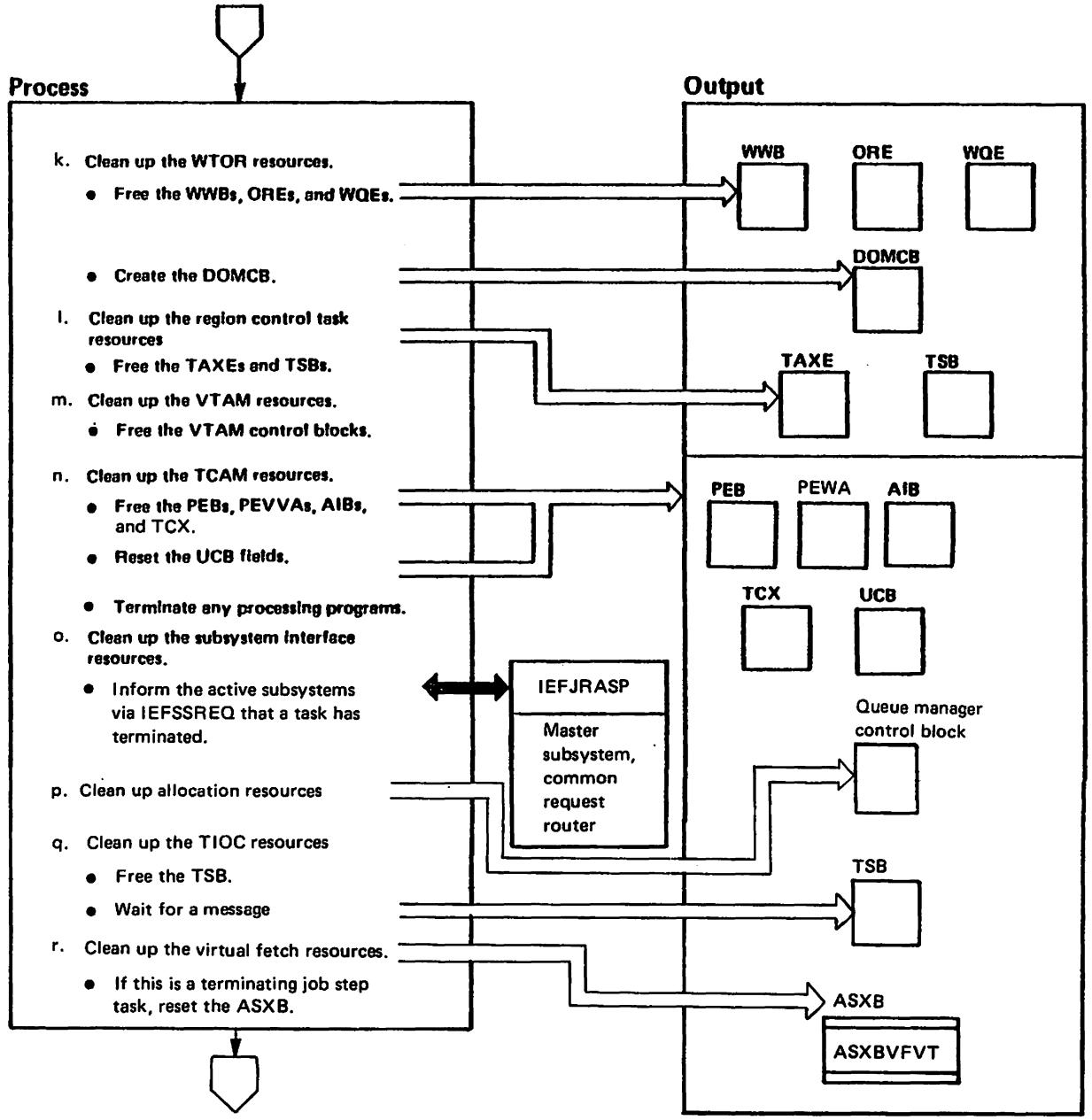
7 Task purge processing indicates, in the RTM2WA, if it is purging the last TCB in the address space. Control then goes to the exit processing, as shown by M.O. diagram IEAVTRTE – RTM2 Exit Processing.

IEAVTSKT – Task Purge Resource Managers (Part 1 of 8)



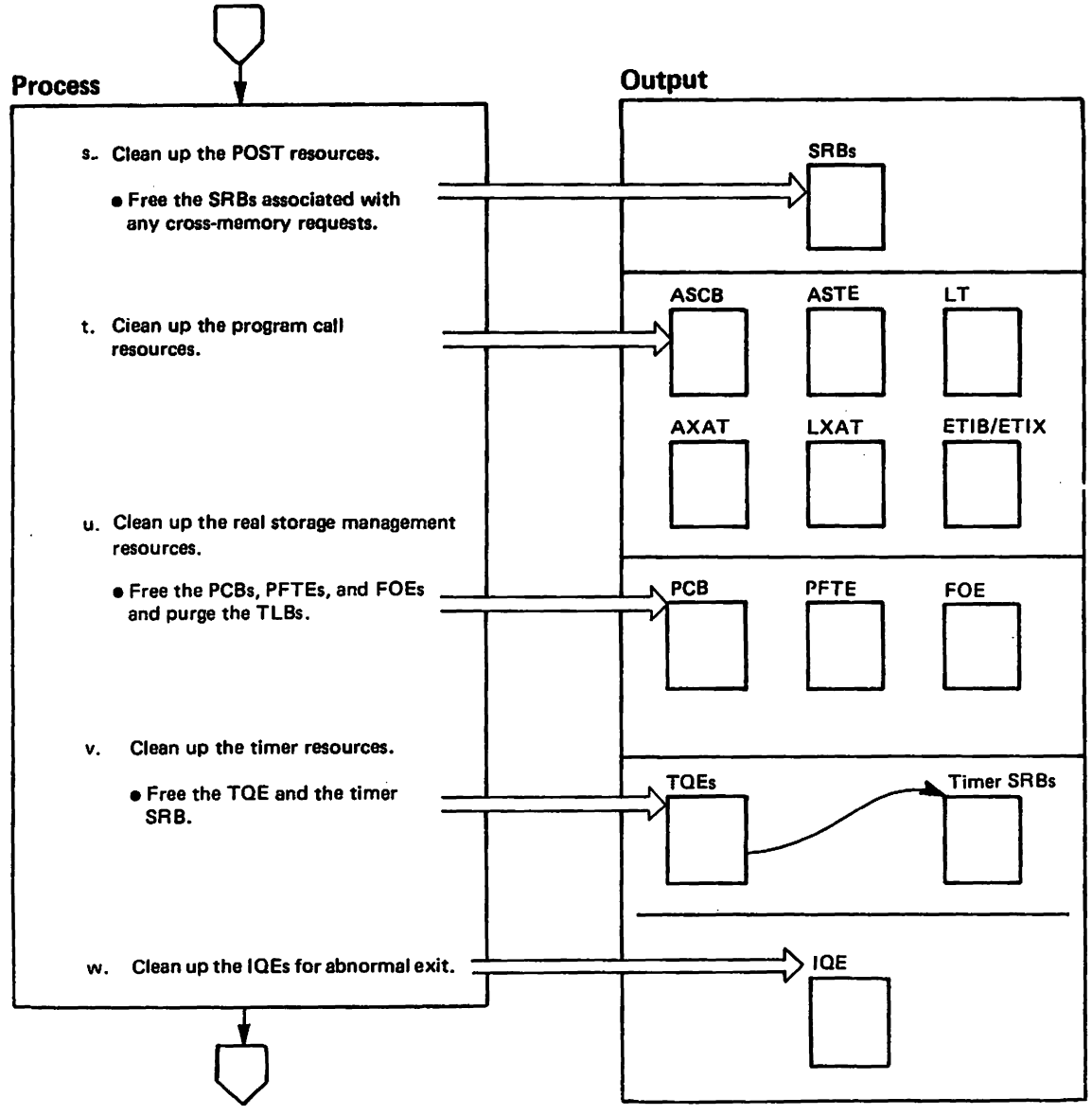
IEAVTSKT – Task Purge Resource Managers (Part 2 of 8)

Extended Description	Module	Label
The IBM-defined task cleanup resource manager routines free resources held during task processing. The task purge processing routine, module IEAVTSKT, routes control to these resource managers after establishing an interface via the RMPL (resource manager parameter list) in the RTM2WA. Control goes to each resource manager sequentially, in the appropriate addressing mode, until all resource managers have performed their clean up processing.		
1 The task purge routine routes control to each of the task purge routine, which routes control to the next resource manager. This continues until all the resource managers have performed cleanup.	IEAVTSKT	TPURG1
a. The SDUMP resource manager frees system resources for: • The SDUMP SRB • Tasks and address spaces involved in an SDUMP • The DUMPSRV address space (See the M.O. diagram IEAVTSDR–SVC Dump Resource Manager in the section "Dumping Services" for a description of the SDUMP resource manager.)	IEAVTSDR	
b. The subsystem support manager builds a broadcast subsystem interface to give control to all interested subsystems.	IEFJRECF	
c. The data management resource manager cleans up the TCBDEBAD field of the TCB. (See *Open/Close/EOV Logic* for more information about the data management resource manager.)	IFG0TC0A	
d. The timer resource manager frees the TQEs and timer SRBs associated with the task being terminated. (See the M.O. diagram IEAQPGTM – Timer Supervision, in section "Timer" for a description of the timer purge routine.)	IEAQPGTM	
e. The system trace resource manager removes all trace table copy headers (TTCH) for the terminating task from the TTCH queue and frees them. (See M.O. diagram IEAVETRM – System Trace in the section "Trace" for a description of the system trace resource manager.)	IEAVETRM	
f. The abnormal exit resource manager cleans up the resources for the task being terminated by freeing the IQE (interruption queue element).	IEAVTPMT	
g. The type 1 message resource manager cleans up the message table pointed to from the CVTQMSG field of the CVT.	IEAVTPMT	
h. SPIE delete processing frees the SPIE resources used by the terminating task by freeing the associated SCA (SPIE control area) and the PIE (program interruption element).	IEAVSPIE	IGC0001D
i. The MSSFCALL SVC resource manager dequeues the MSSFCALL control blocks.	IEAVMFRM	
j. The ENQ resource manager frees the associated ENQ resources used by the terminating task by freeing the QCBs (queue control block) and QELs (queue element). The ENQ resource manager also prints messages explaining which task failed while it controlled the resource. (See the section "Global Resource Serialization" for a description of ENQ processing.)	IEAVENQ2	



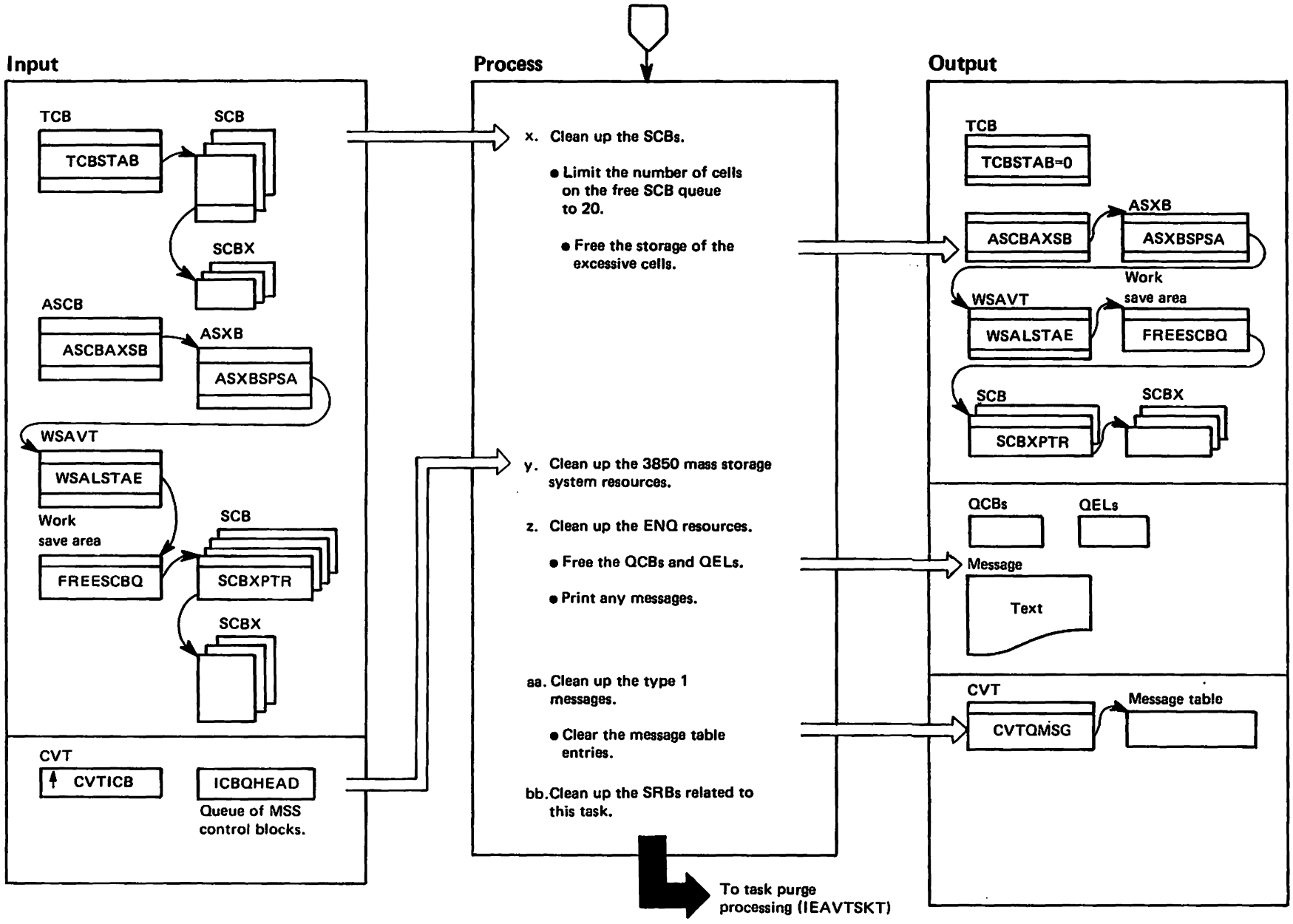
IEAVTSKT – Task Purge Resource Managers (Part 4 of 8)

Extended Description	Module	Label
k. The communications task resource manager cleans up WTOR (write to operator with reply) resources associated with the task being terminated, by freeing control blocks.	IEAVMED2	
l. The region control task resource manager cleans up the resources associated with the task being terminated by freeing the TAXEs (terminal attention exit element) and TSBs (terminal status block). (See the M.O. diagrams for the region control task in the section "Address Space Services" for a complete description of the region control task resource manager.)	IEAVAR07	
m. The VTAM resource manager cleans up resources associated with the VTAM user task. These resources include storage, VTAM locks, and the following control blocks associated with the VTAM devices and applications active for the terminating task: • Active CRAs (component recovery area) • DEBs (data extent blocks) • FMCBs (function management control block) • NCBs (node control block) • ICEs (inactive connection element) • ACEs (active connection element) • DCEs (DEB chain element) • PST (process scheduling table) • Application RDTEs (resource definition table) • Destination RDTEs • DVTs (destination vector table) • EPTs (entry point table) (See the publication *VTAM Logic* for a description of VTAM processing)	ISTRAMAI	
n. The TCAM (telecommunications access method) resource manager frees the resources associated with the terminating task. This resource manager frees the PEBs (process extension block), PEWAs (process entry work areas), AIBs (application interface blocks), and TCX (TCAM CVT extension) associated with the failing task, and it resets UCB (unit control block) fields. (See the publication *TCAM Logic* for a description of the TCAM resource manager.)	IEDQ0T01	
o. The subsystem interface resource manager cleans up the resources associated with the failing task by notifying the tactive subsystems, via the IEFSSREQ macro, of the task that just terminated.	IEFJRASP	
p. Only at abnormal termination does the allocation resource manager clean up the queue manager block for the falling task.	IEFAB4E5	
q. The TIOC (terminal input/output coordinator) resource manager cleans up the TSB for the task being terminated.	IEDAY8	
r. If a job step task is terminating, the virtual fetch resource manager sets the ASXBVFVT field in the ASXB to zero.	CSVVFMEM	



IEAVTSKT – Task Purge Resource Managers (Part 6 of 8)

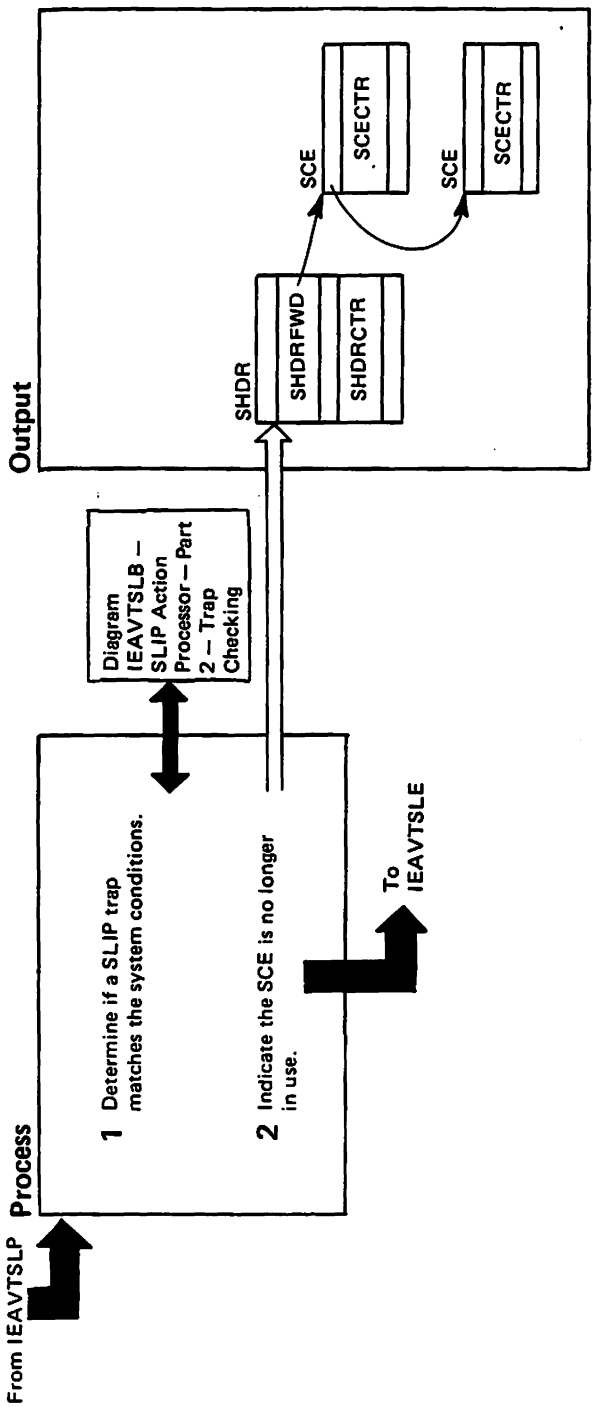
Extended Description	Module	Label
s. The POST resource manager cleans up the resources associated with the task being terminated by freeing the SRB associated with any cross-memory POST requests. (See the M.O. diagram IEAVEPST – POST Processing in the section "Task Management" for more details.)	IEAVEPST	
t. An inline macro (PCARM) gives the program call authorization resource manager control. This resource manager cleans up the program call resources.	IEAVXPAM	
u. The real storage management resource manager cleans up the resources associated with the task being terminated by freeing the PCBs (page control block), PFTE (page frame table entry), FOE (fix ownership entry), and purging TLB (translation lookaside buffer).	IEAVTERM	
v. The timer resource manager frees the TQEs and timer SRBs associated with the terminated task. (See the M.O. diagram IEAQPGTM – Timer Supervision in the section "Timer", for a description of the timer purge routine.)	IEAQPGTM	
w. The abnormal exit resource manager cleans up the resources for the task being terminated by freeing the IQE (interruption queue element.)	IEAVEEEP	



IEAVTSKT – Task Purge Resource Managers (Part 8 of 8)

Extended Description	Module	Label
x. The SCB freemain routine frees the storage occupied by SCBs no longer needed. The SCB freemain routine limits the queue of available SCBs to 20 cells. (See the M.O. diagram IEAVTSCB – SCB Freemain, for a more detailed description.)	IEAVTSCB	
y. The 3850 mass storage system resource manager marks invalid all delayed response queue elements relating to the terminating task.	ICB2AIR	
z. The ENQ resource manager frees the associated ENQ resources used by the terminating task by freeing the QCBs (queue control block) and QELs (queue elements). The ENQ resource manager prints messages explaining which task failed while the task controlled the resource. (See the section "Global Resource Serialization" for a description of ENQ processing.)	IEAVENQ2	
aa. The type 1 message resource manager cleans up the message table pointed to by the CVT (CVTQMSG).	IEAVTPMT	
bb. The task purge routine uses the PURGEDQ function to clean up any SRBs related to the terminating task.	IEAVEPD0	

IEAVTSLB - SLIP ACTION PROCESSOR - PART 2

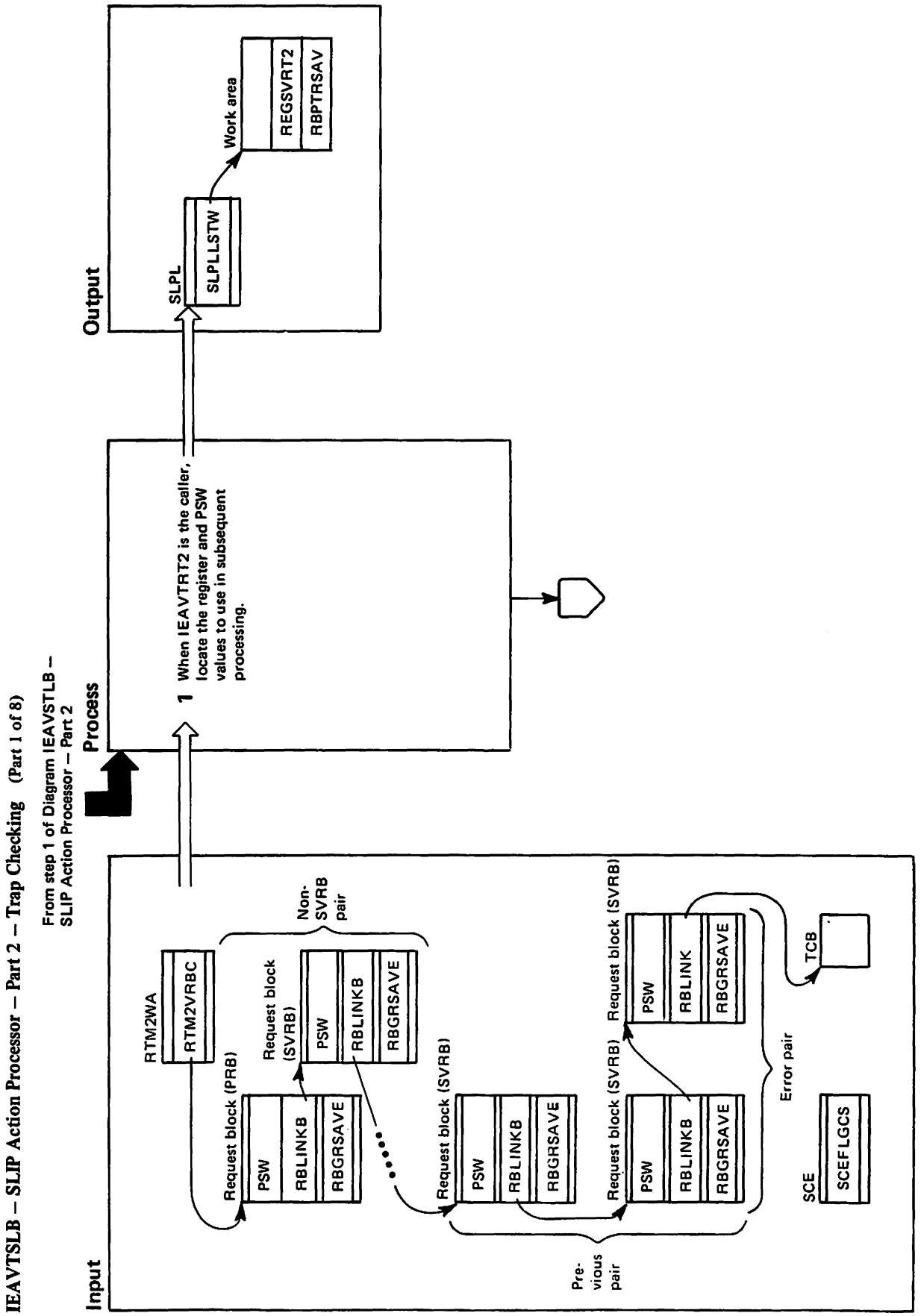


IEAVTSLB - SLIP Action Processor - Part 2 (Part 1 of 2)

IEAVTSLB – SLIP Action Processor – Part 2 (Part 2 of 2)

Extended Description	Module	Label
This module is called by IEAVTSLP. It routes control to the SLIP trap match routines (IEAVTSL1 or IEAVTSL2 depending on the parameters found in the SCVA) and processes the MATCHLIM and PRCNTLIM keywords. See diagram IEAVTSLB – SLIP Action Processor – Part 2 – Trap Checking for a description of IEAVTSLB's trap checking process.		
1 Starting with the last trap added to the chain, or the enabled non-IGNORE PER trap, IEAVTSLB compares the trap requirements (indicated in the SCE, SCVA pair) with the current system condition. When SLIP's caller is IEAVTRTS, IEAVTRT2, or IEAVTRTM, IEAVTSLB checks only enabled non-PER traps; when SLIP's caller is IEAVTPER, IEAVTSLB checks only enabled PER traps. IEAVTSLB never checks disabled traps. Diagram IEAVTSLB – SLIP Action Processor – Part 2 – Trap Checking describes the individual trap comparison process and, with Diagram IEAVTSL2 – SLIP Trap Matching Routine – Part 2 – ACTION Keyword Processing, the action taken when a match is found. After a trap is checked, processing continues at the next step.	IEAVTSLB	
2 If a no-match condition is found and other traps remain to be tested, IEAVTSLB decreases the SCE use count by one, locates the next trap to be checked, and returns to step 1 to repeat the comparison process. Non-PER trap checking stops when a match is found or all the traps have been examined. PER trap checking stops when a match is found or the enabled non-IGNORE PER trap has been checked. If the search for a matching trap is being terminated, IEAVTSLB decreases the use counts in all the remaining serialized traps.		SCEDECR

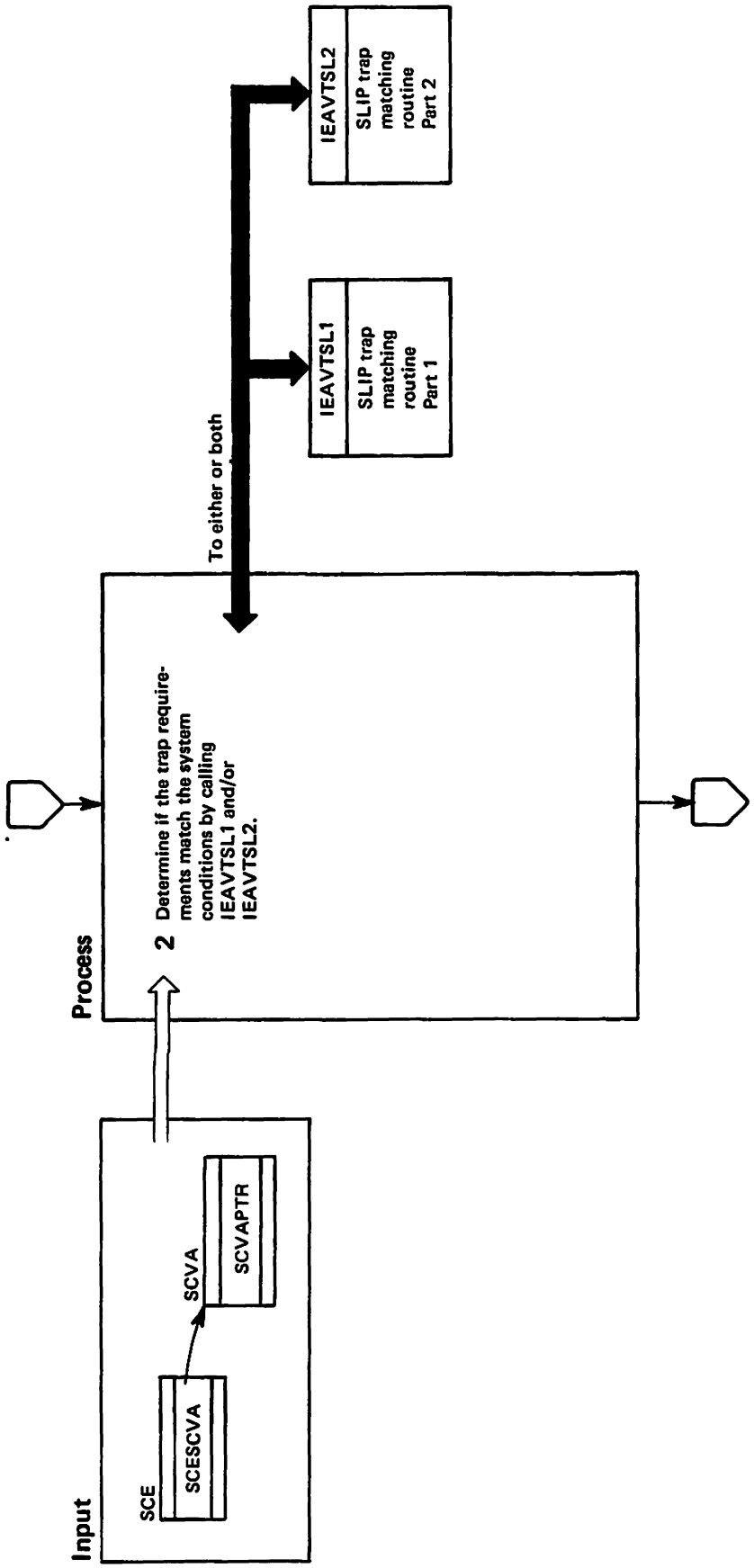
IEAVTSLB - SLIP ACTION PROCESSOR - PART 2 - TRAP CHECKING



IEAVTSLB – SLIP Action Processor – Part 2 – Trap Checking (Part 2 of 8)

Extended Description	Module	Label
This diagram describes the segment of IEAVTSLB that compares a trap's event qualifier keyword parameters with the current system status.	IEAVTSLB	
1 When the caller is IEAVTRT2, IEAVTSLB determines from which request block (RB) it is to obtain the register and PSW values necessary for later processing. The SLIP trap RBLEVEL keyword parameter (recorded in the SCEFLGCS field) specifies one of three pairs of request blocks: the error pair, the previous pair, or a non-SVRB pair. (The default value is the error pair.) IEAVTSLB uses the register values in one of the RBs and the corresponding PSW in the other. It saves a pointer to the RB containing the registers in the REGSVRT2 field, and a pointer to the RB containing the PSW in the RBTRSAV field. It uses the register values, for example, to resolve indirect addresses. The PVTMOD, ADDRESS, and/or LPANUC match subroutines use the PSW. When IEAVTRT2 is not the caller, this step is skipped.		

: 2 - Trap Checking (3 of



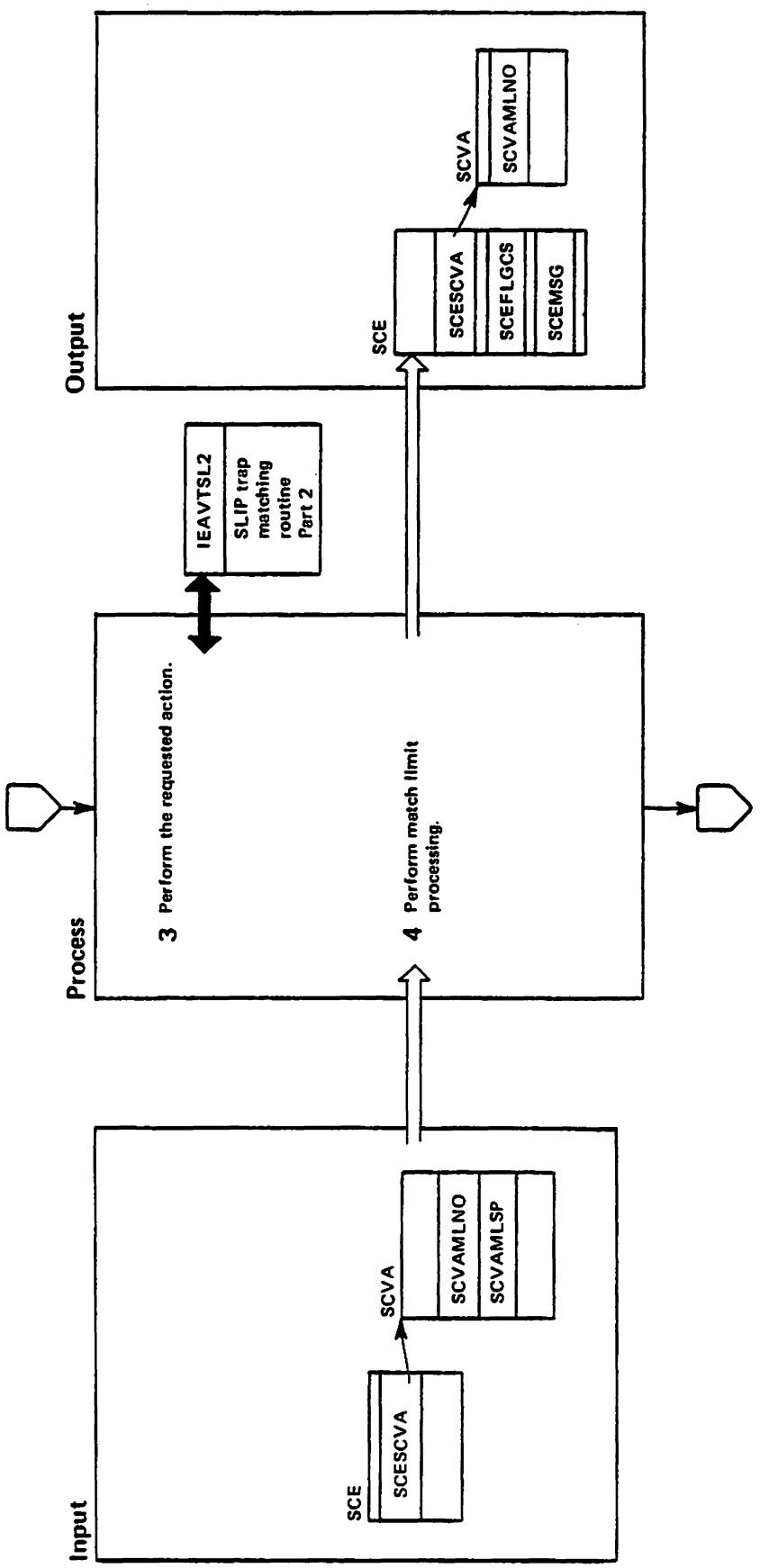
IEAVTSLB – SLIP Action Processor – Part 2 – Trap Checking (Part 3 of 8)

Extended Description	Module	Label
2 Each event qualifier keyword that can be specified on the SLIP command has a corresponding external subroutine (contained in modules IEAVTSL1 and IEAVTSL2) which determines whether the trap's keyword parameters match the current system conditions. An event qualifier keyword is a keyword with parameters describing conditions which must match current system conditions before action is taken. All keyword parameters are passed to IEAVTSLB as records in the SCVA. (Recall that each trap consists of an SCE and SCVA pair.) Beginning with the first SVCA record, IEAVTSLB calls the corresponding subroutine. The subroutine makes the comparison, sets a return code in register 15 indicating a match or no-match condition, and updates the SCVAPTR to point to the next record.	IEAVTSL1 IEAVTSL2	
If a match condition is found and keywords remain to be checked, IEAVTSLB goes to the next SCVA entry and calls its corresponding subroutine. IEAVTSLB repeats this process until a no-match condition is found, or all the keywords have been checked and their parameters found to match. If all the keyword parameters match, the subroutine corresponding to the ACTION keyword (EOL) receives control to take the action requested for the trap. Diagram IEAVTSL2 – SLIP Trap Matching Routine – Part 2 – ACTION Keyword Processing describes EOL processing.		
When a subroutine returns a no-match condition, IEAVTSLB stops match checking for the current trap, and continues processing at step 4.		

Extended Description	Module	Label
This chart shows in which module each match routine is located:		

For event qualifier keyword	*Go to Module*
ADDRESS	IEAVTSL1
ASID	IEAVTSL1
ASIDSA	IEAVTSL1
COMP	IEAVTSL1
DATA	IEAVTSL2
ERRTYP	IEAVTSL1
JOBNAME	IEAVTSL1
JSPGM	IEAVTSL1
LPAMOD	IEAVTSL1
MODE	IEAVTSL1
NUCMOD	IEAVTSL1
PVTMOD	IEAVTSL2
RANGE	IEAVTSL1
REASON	IEAVTSL1

IEAVTSLB only calls a match routine if its corresponding keyword is specified on the SLIP trap.

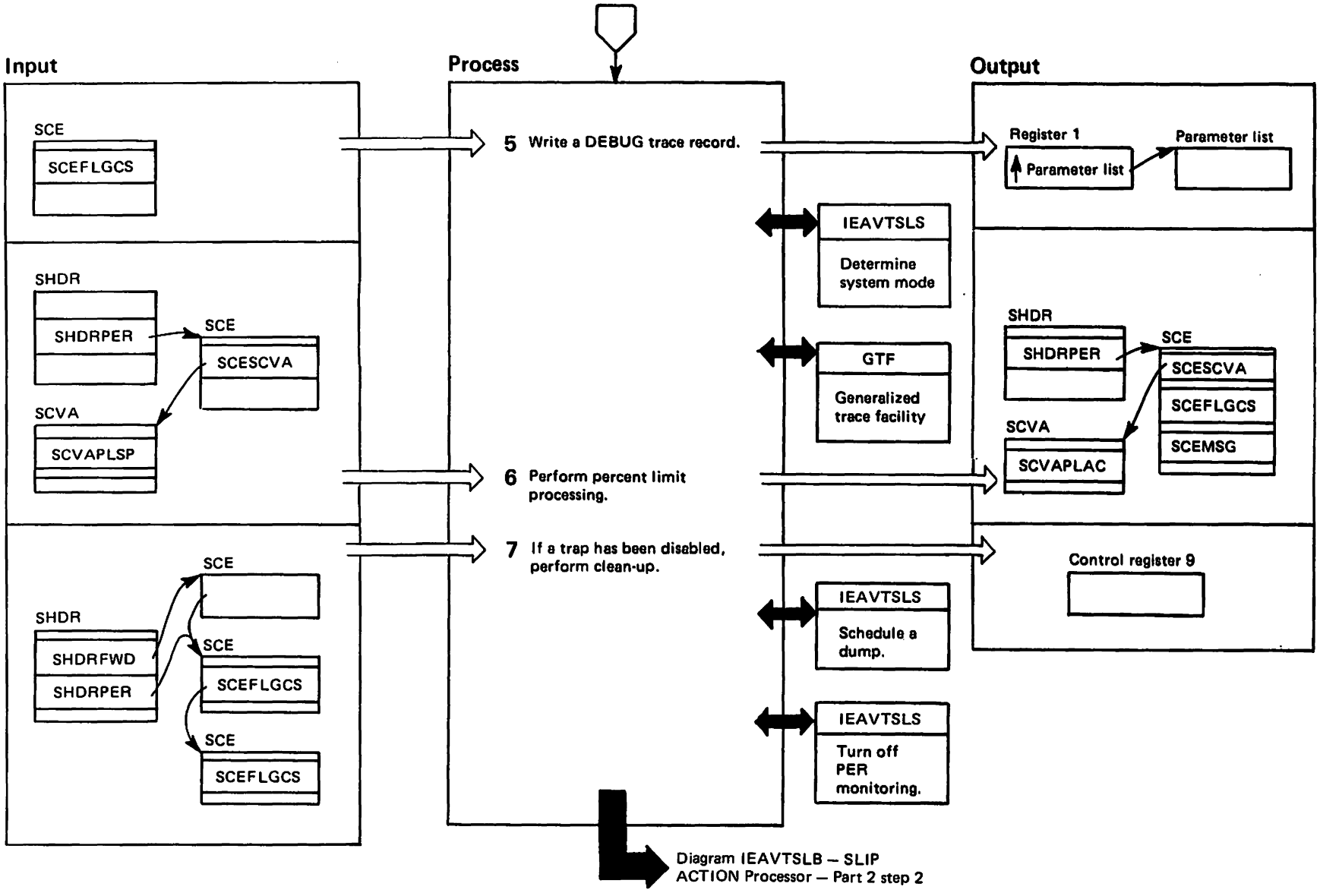


IEAVTSLB – SLIP Action Processor – Part 2 – Trap Checking (Part 5 of 8)

IEAVTSLB – SLIP Action Processor – Part 2 – Trap Checking (Part 6 of 8)

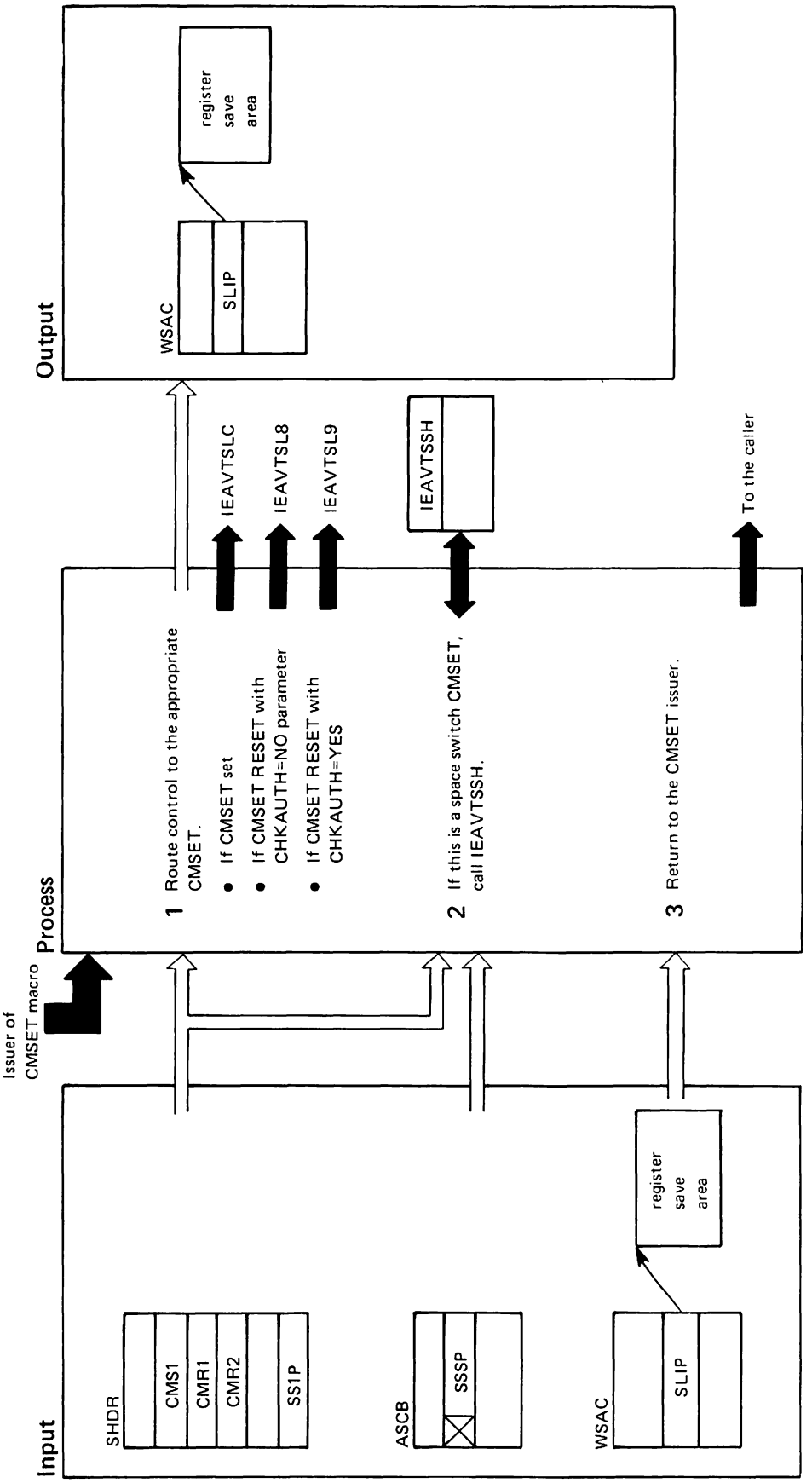
Extended Description	Module	Label
3 If all of the event qualifier keywords match, IEAVTSL2 receives control to take the action requested by the trap's ACTION parameters. This processing is described in the M.O. diagram IEAVTSL2 – SLIP Trap Matching Routine Part 2 – ACTION Processing.	IEAVTSL2	
4 Match limit processing determines whether the number of matches for a particular trap equals or exceeds the maximum number the trap allows. IEAVTSLB performs match limit processing when a match occurs for a trap that specifies or has a default MATCHLIM value. IEAVTSLB: • Adds one to the current number of matches for the trap (the SCVAMLN0 value). • Determines whether the current number of matches is greater than or equal to the match limit (the SCVAMLSP value). • If the match limit is reached, disables the trap by turning on the SCEDSABL bit in the SCEFLGCS field, and sets the SCEM411M bit, which causes message IEA411I to be issued.		

IEAVTSLB – SLIP Action Processor – Part 2 – Trap Checking (Part 7 of 8)



Extended Description	Module	Label
5 If the trap specifies the DEBUG option (SCEDEBUG=1), IEAVTSLB builds a parameter list for the generalized trace facility (GTF) and gives control to GTF by issuing an appropriate form of the HOOK macro. GTF writes a trace record containing information relevant to IEAVTSLB's processing for the trap.		
6 Percent limit processing determines whether the percentage of time spent processing PER interruptions exceeds the limit specified on the trap. When all of the following conditions are true, IEAVTSLB performs percent limit processing. • A PER interruption is being processed • An IGNORE PER trap match occurred or the enabled non-IGNORE PER trap has been examined • The trap user requests percent limit processing (the SCVAPLSP field in the enabled non-IGNORE trap does not equal 99) • A valid time-of-day (TOD) clock value was stored and a current TOD clock value can be obtained IEAVTSLB does not perform the limit check until approximately 33 seconds have elapsed since the first valid PER interruption occurred. This eliminates the possibility of a false high initial percentage. To determine whether the percent limit has been exceeded, IEAVTSLB: • Subtracts the value in the WAECKLIN field (the TOD clock value at time of entry) from the current time to get the accumulated time for the interruption • Adds the above value to the accumulated time spent processing all previous PER interruptions and stores the total in the SCVAPLAC field of the enabled non-IGNORE PER trap • Obtains the number of space switch interruptions since the last percent limit calculation (SCVAPLSC), multiplies that by the time it takes to handle one space switch interruption (SHDRSSTM), and adds the product to the SCVAPLAC. • Computes the total time elapsed since the first valid PER interruption and stores it in the WAETOTT field • Calculates the percentage of time spent processing PER and space switch interruptions • Compares the above percentage with the percent limit stored in the SCVAPLSP field of the enabled non-IGNORE PER trap If the limit is exceeded, IEAVTSLB disables the trap by turning on the SCEDSABL bit in the SCEFLGCS field. It sets the SCEM411P bit in the non-IGNORE trap to cause message IEA411I to be issued in subsequent processing. Processing continues at step 7.		
7 If a trap has been disabled during match limit or percent limit processing, IEAVTSLB performs cleanup processing. If the non-IGNORE PER trap was disabled, IEAVTSLB: • Calls IEAVTSLS to schedule IEAVTGLB to run as an SRB. IEAVTGLB deactivates PER in the system. • If the TRDUMP or STDUMP option is coded on the trap, calls IEAVTSLS to schedule an SVC dump. • Adjusts the return code to request that the PSW PER bit be turned off in the resume PSW for the interrupted process. • Enters zeros in control register 9 of the current processor to help remove the effects of having PER on in the system. If a non-PER trap was disabled and the TRDUMP option is coded on the trap, IEAVTSLB calls IEAVTSLS to schedule a dump. If an IGNORE type PER trap is disabled, IEAVTSLP performs no cleanup processing.	IEAVTSLS IEAVTSLS	

IEAVTSLC - SLIP/CMSET INTERCEPT INTERFACE ROUTINE

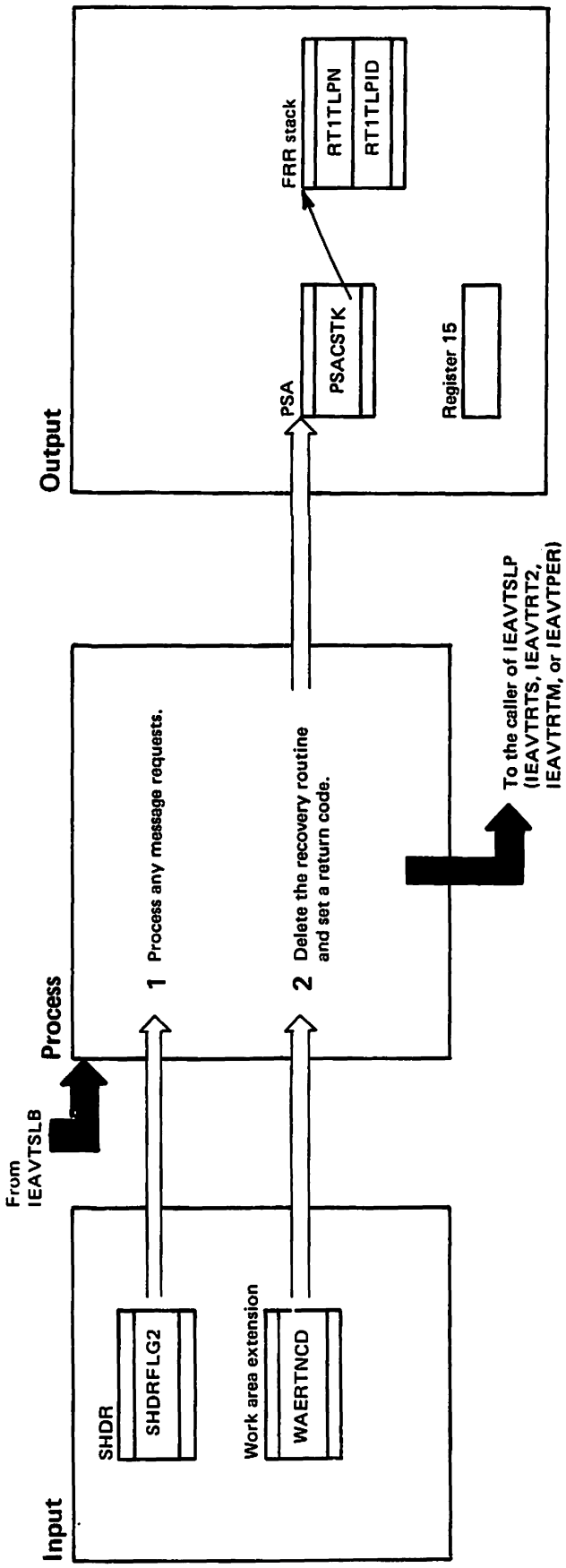


IEAVTSLC – SLIP/CMSET Intercept Interface Routine (Part 1 of 2)

IEAVTSLC – SLIP/CMSET Intercept Interface Routine (Part 2 of 2)

Extended Description	Module	Label
IEAVTSLC intercepts calls to change a unit of work's addressing environment. IEAVTSLC has separate entry points for the macros it intercepts. IEAVTSLC – intercepts CMSET SET macros. IEAVTSL8 – intercepts CMSET RESET macros when the parameter CHKAUTH=NO is coded. IEAVTSL9 – intercepts CMSET RESET macros when the parameter CHKAUTH=YES is coded.		
During SLIP initialization, the addresses of the CMSET routines were saved in the SLIP header (SHDR) and replaced by the corresponding entry point addresses of this module. IEAVTSLC and IEAVTSSH assure the integrity of the SLIP trap in a cross memory environment no matter where a unit of work is operating.		
1 When a caller issues a CMSET SET, CMSET RESET with CHKAUTH=YES, or CMSET RESET with CHKAUTH=NO macro, IEAVTSLC passes control to the appropriate macro. (Recall that the entry points in system routines for these macros were saved in the SLIP header when SLIP was initialized.) The CMSET SET and CMSET RESET macros adjust the addressing environment so that the caller can issue a PC or PT instruction. After the macro is executed, control returns to IEAVTSLC.		IEAVTSLC IEAVTSL8 IEAVTSL9
2 If the current PASID is different from the PASID prior to the CMSET and if the space switch flag (ASCBSSSP) in either the current or former ASCB is set. IEAVTSLC calls IEAVTSSH to perform a space switch.	IEAVTSSH	
3 IEAVTSLC returns control to the issuer of CMSET.		

IEAVTSLE - SLIP ACTION PROCESSOR - PART 3



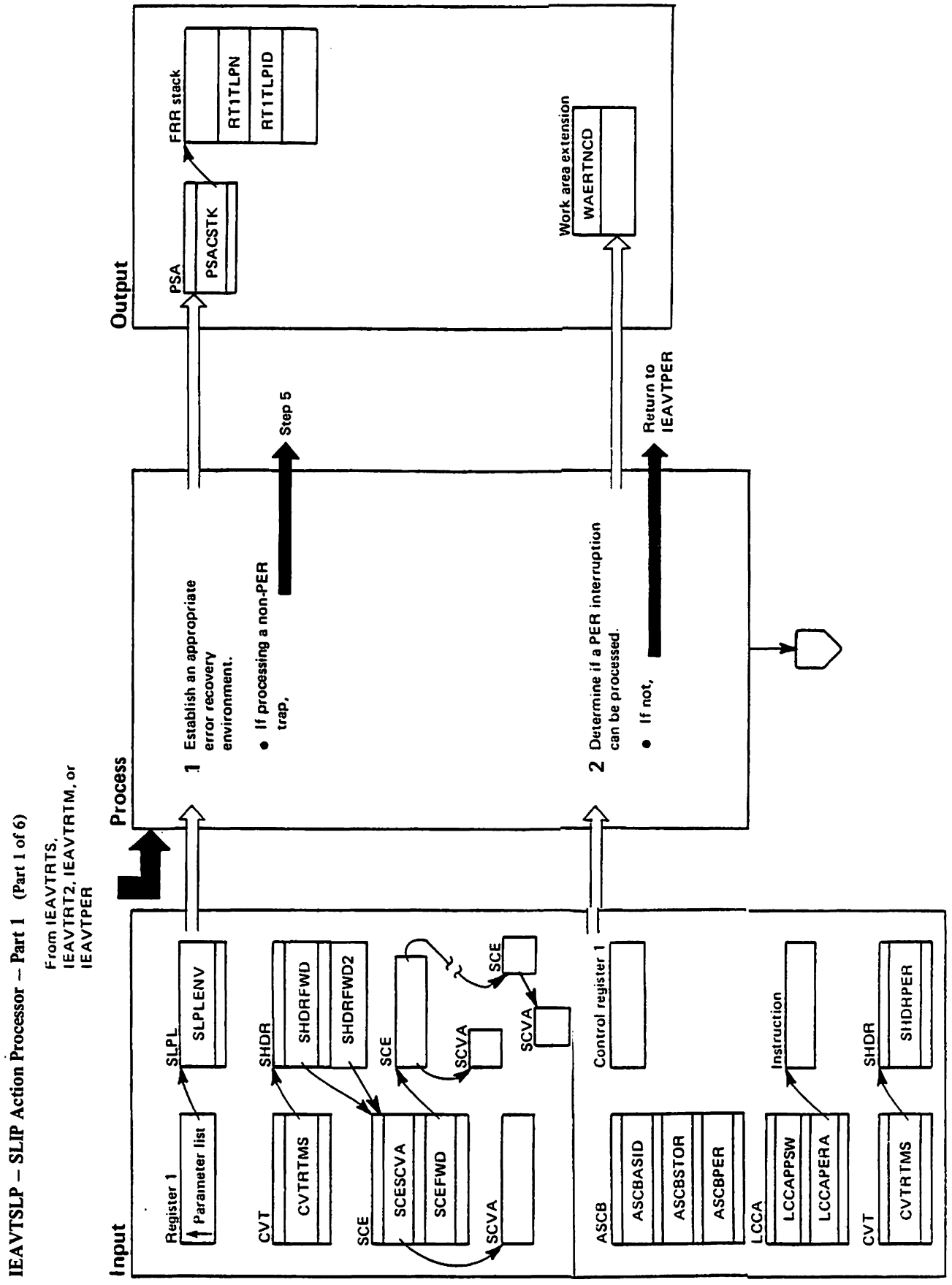
IEAVTSLE – SLIP Action Processor – Part 3 (Part 1 of 2)

IEAVTSLE – SLIP Action Processor – Part 3 (Part 2 of 2)

Extended Description	Module	Label
IEAVTSLE initiates message processing requested by IEAVTSLP when a trap has been automatically disabled. IEAVTSLE also removes the recovery environment established by IEAVTSLP and returns to the caller of IEAVTSLP.		
1 If the SHDRPSTM bit in the SHDRFLG2 field equals one, IEAVTSLP processing has requested that a message be issued. IEAVTSLE schedules IEECB915 to run as an SRB in the master address space. IEECB915 posts the SLIP command processor communication routine (IEECB905), which performs the processing necessary to issue messages requested by IEAVTSLP.	IEAVTSLE	AGAIN10
2 IEAVTSLE deletes the recovery routine established by IEAVTSLP. If a PER interruption is being processed, IEAVTSLE copies the WAERTNCD field into register 15. Possible return codes and their meanings are:		SLIPRT

Return code	*Meaning*
0	Return to the interrupted program.
4	Force recovery for the interrupted process.
8	Turn off the PER bit in the resume PSW and return to the interrupted program.
12	Turn off the PER bit in the resume PSW and force recovery for the interrupted process.

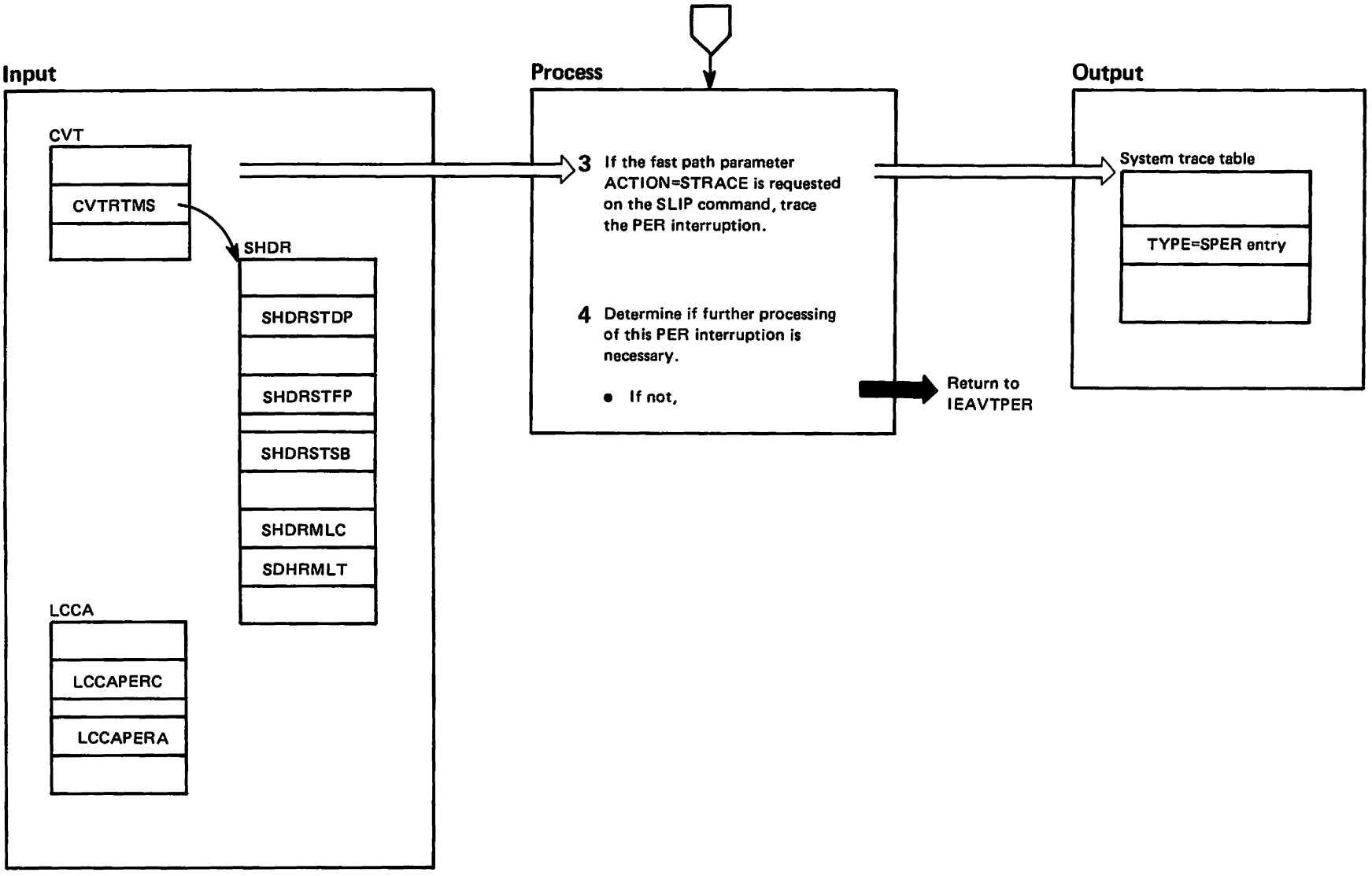
IEAVTSLP - SLIP ACTION PROCESSOR - PART 1



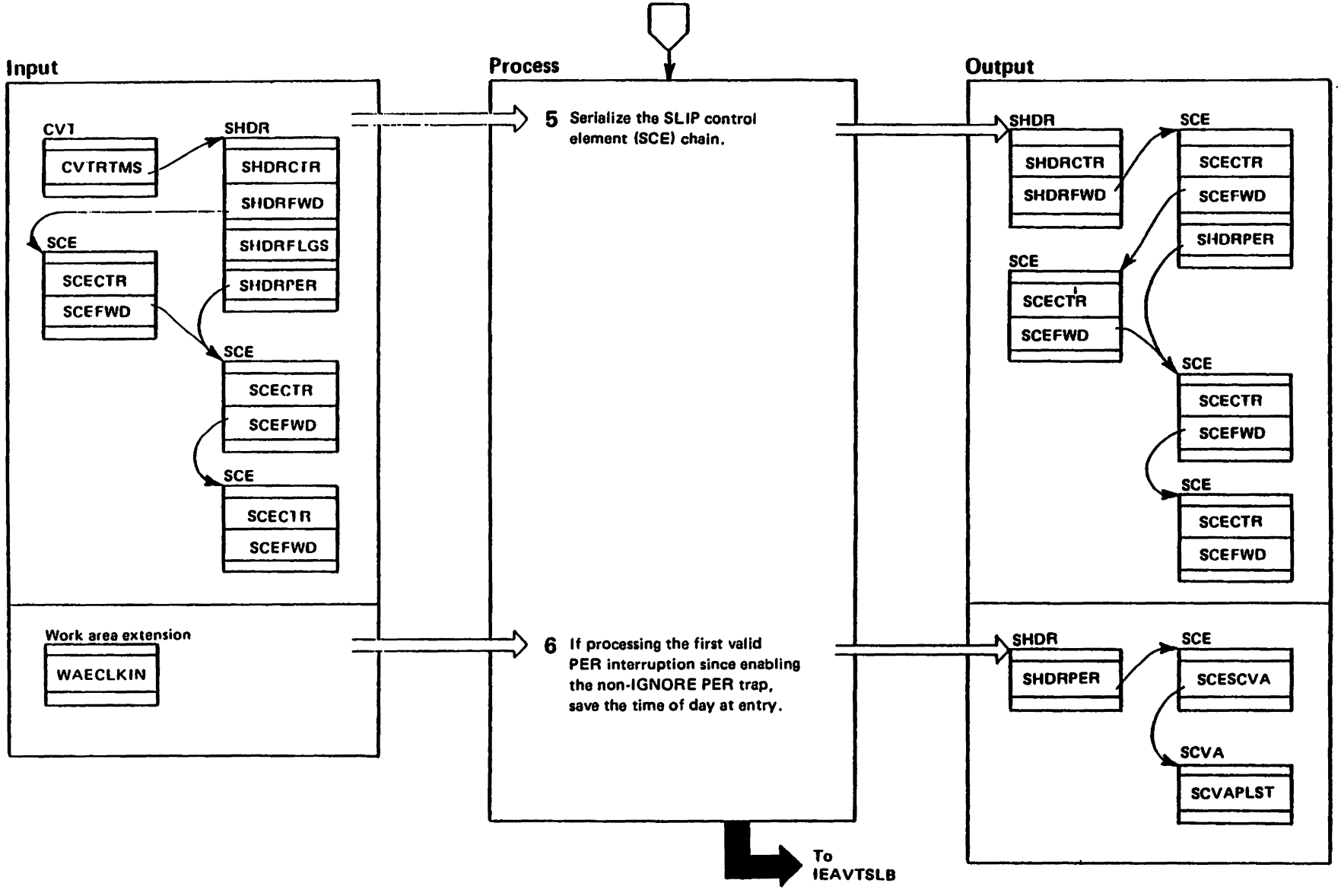
IEAVTSLP – SLIP Action Processor – Part 1 (Part 2 of 6)

Extended Description	Module	Label
When the SLIP facility is active, IEAVTSLP receives control from either a recovery termination management routine (IEAVTRTM, IEAVTRT2, or IEAVTRTS) as part of its normal error processing or from the program check FLIH/SLIP interface routine (IEAVTPER) after a PER interruption occurs. IEAVTSLP determines whether there is a SLIP trap with requirements that match the current system conditions.		
There are two types of SLIP traps, PER and non-PER: • PER traps allow the programmer to define the PER event that is to trigger trap processing. Only one non-IGNORE PER trap (any PER trap that does not specify the parameter ACTION=IGNORE) can be enabled at a time. • Non-PER traps allow the programmer to design error recognition traps for errors that are handled by recovery termination management (RTM).		
Each trap is represented by a pair of control blocks, the SLIP control element (SCE) and the SLIP control element variable area (SCVA). The SCEs are chained together, with the newest trap being added at the end of the chain. The SHDRFWD field points to the oldest PER trap element. The SHDRFWD2 field points to the oldest non-PER trap element.		
When searching for a match, IEAVTSLP begins with the newest trap (last on the chain). If the caller is IEAVTRTS, IEAVTRT2, or IEAVTRTM, it checks only non-PER traps until a match is found or all the traps have been checked. When the caller is IEAVTPER, IEAVTSLP checks only PER traps, and stops when a match is found or the enabled non-IGNORE PER trap has been checked. When a match is found, IEAVTSLP performs the SLIP action specified by the trap's ACTION keyword parameter.	IEAVTSLP	
At various points in its processing, IEAVTSLP must identify the caller. If unable to do so, it issues an ABEND with a system completion code of X'06C'.		
1 IEAVTSLP refers to the SLPLENV field to identify the caller, then establishes error recovery to suit that environment.	IEAVTSLP	
For IEAVTRTS: IEAVTSLP enters a logical phase number in the RT1TLPN field to allow FRR recovery and sets the logical phase recovery routine ID to the RTS FRR (RT1TLPID=1). It substitutes the FRR stack chosen by RTM for the current stack (PSACSTK=SLPLRTSF) and adds an FRR. Processing continues at step 5.		
For IEAVTRT2: IEAVTSLP obtains the local lock to allow setting an FRR and adds an FRR to the stack. It sets a section flag and changes an error recovery flag to allow section recovery. Part of the code to set up the error recovery environment is located in IEAVTRT2. (See the RTM2 diagrams for diagrams and extended descriptions of IEAVTRT2.)		
For IEAVTRTM: IEAVTSLP determines if there is enough room in the FRR stack for the maximum number of FRRs that might be needed for recovery in this environment. If not enough space exists, IEAVTSLP returns to IEAVTRTM. Otherwise, it adds an FRR to the current stack. Processing continues at the next step.	IEAVTSLP	
For IEAVTPER: IEAVTSLP adds an FRR to the current (PCFLIH) stack.		
2 If one of the following is true, a PER interruption cannot be processed:		
a. DAT is off (the DAT bit in the old PSW is off).		ERROR2
b. The PER interruption is redundant (one that will be reported again).		
c. No enabled non-IGNORE PER trap exists (SHDRPER=0).		ERROR3
If condition a or b exists, IEAVTSLP returns to IEAVTPER with a return code of zero. When condition c is true, IEAVTSLP returns to IEAVTPER with a return code of eight. In either case, the FRR is deleted before returning. If none of the conditions are true, normal processing continues at the next step.		

IEAVTSLP – SLIP Action Processor – Part 1 (Part 3 of 6)

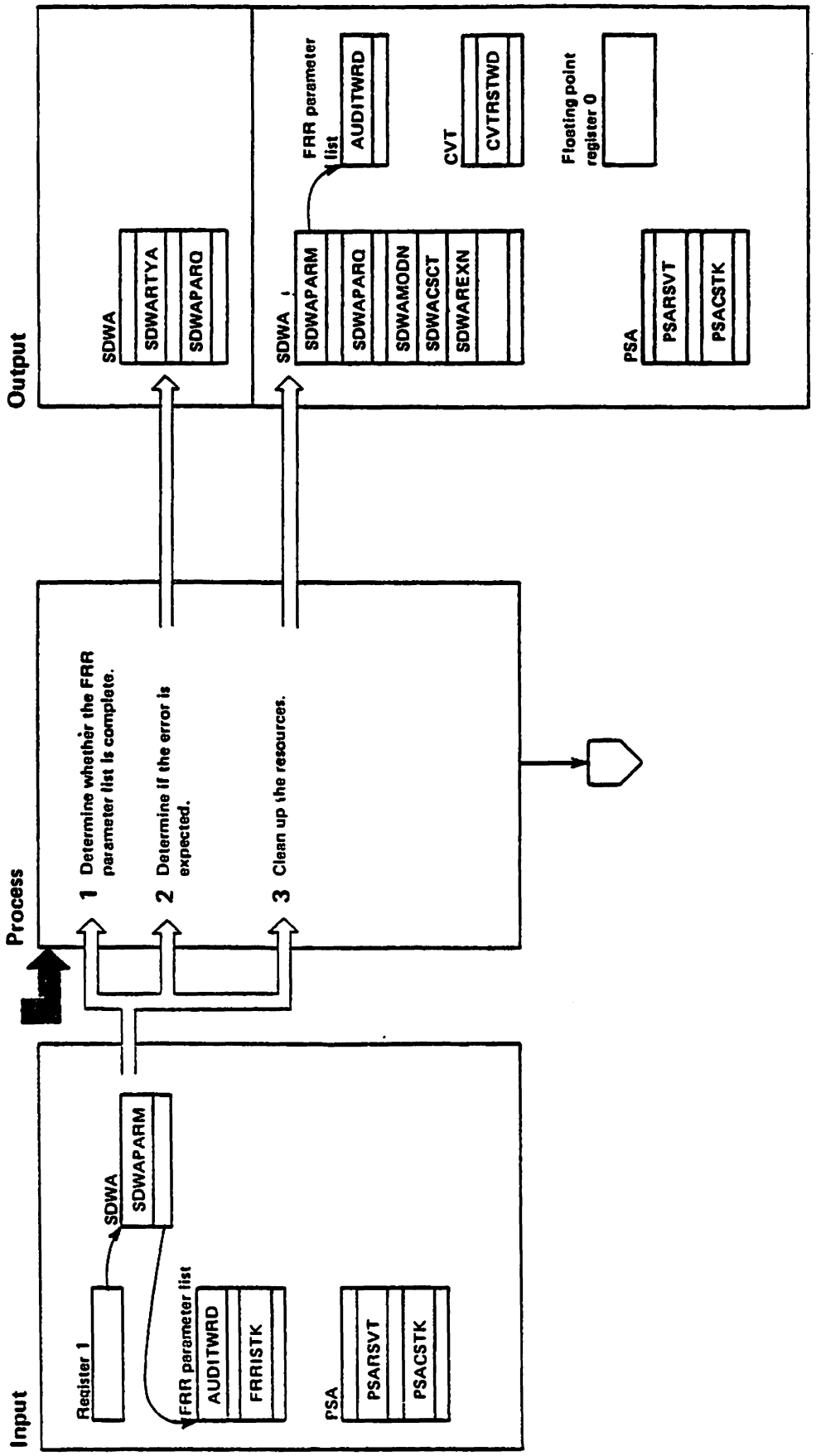


Extended Description	Module	Label
3 If the fast path parameter ACTION=STRACE is re-requested (the SHDRSTFP bit is on) and if the SLIP event that requested the fast path ACTION=STRACE is still enabled (the SHDRSTDP bit is off), IEAVTSLP issues the PTRACE macro to trace the PER interruption. The MATCHLIM keyword processing provides no filters. Every PER interrupt results in a match condition. IEAVTSLP increments the match counter (SHDRMLC) via a compare and swap instruction.		FASTRACE
ACTION=STRACE processing generates an SPER entry in the system trace table. The SPER entry contains the current program interrupt trace data, the PER code, the interrupt address from the PSA, and the SLIP-PER trap-ID.		STRACE
If the match counter equals the match limit (SHDRMLT), IEAVTSLP disables the fast path ACTION=STRACE function by setting the SHDRSTDP bit on, using a compare and swap instruction.		
4 Processing of the PER interruption continues if one of the following situations occurs: • Fast STRACE has not been requested (the SHDRSTFP bit is off). This PER interruption will go through SLIP's complete filter and limit processing. • The PER interruption was traced but it caused a match limit condition to occur (SHDRMLC reached SHDRMLT). The PER interruption is passed to IEAVTSLB for cleanup processing that will deactivate PER interruptions within the system.		
IEAVTSLP deletes the FRR and returns to IEAVTPER with a return code of zero if one of the following occurred: • Fast STRACE had been requested but is now disabled (the SHDRSTDP bit is on). • The PER interruption was traced and it did not cause a match limit condition. • The PER interruption was not recognizable.		



Extended Description	Module	Label
5 IEAVTSLP serializes the SCE chain to prevent the SLIP command processor (IEECB905) from deleting a trap while this module is examining it. To do so in most cases, IEAVTSLP adds one to the SCE use count in the SHDR control block and one to the use counts in all but the last SCE on the chain. The one exception to this is when a PER interruption is being processed and there are no enabled traps that specify the IGNORE parameter and have the same PER event type as the enabled non-IGNORE PER trap (the SHDRPERI bit in the SHDRFLGS field equals 0). In this case, IEAVTSLP serializes only the SCEs up to the enabled non-IGNORE PER trap (pointed to by the SHDRPER field).		SCEINCR
6 If this is not the first valid PER interruption since enabling the non-IGNORE PER trap, IEAVTSLP skips this step. A valid PER interruption is one for which none of the conditions in step 2 are true. IEAVTSLP enters the time-of-day clock value from the WAECLKIN field in the SCVAPLST field of the enabled non-IGNORE PER trap for subsequent use in percent limit processing.		AGAIN4

IEAVTSLR - SLIP PROCESSOR RECOVERY ROUTINE



IEAVTSLR – SLIP Processor Recovery Routine (Part 1 of 6)

Extended Description **Module** **Label**

This module provides recovery for the SLIP modules (IEAVTSLP, IEAVTSLB, IEAVTSLE, IEAVTSL1, IEAVTSL2, and IEAVTSLS) and the address convert routine (IEAVTADR). If a recoverable error occurs, it attempts a retry at an appropriate point in SLIP processing or IEAVTADR. If an unrecoverable error occurs, IEAVTSLR requests percolation. Input to IEAVTSLR is the address of the FRR parameter list containing footprints (in AUDITWRD) set during SLIP processing.

1 IEAVTSLR determines whether the error occurred before IEAVTSLP finished entering the basic recovery information in the FRR parameter list. If so (the ABPLIST bit in AUDITWRD equals zero), the error is unrecoverable and IEAVTSLR returns to IEAVTRTS to percolate. If the information has been saved (ABPLIST=1), IEAVTSLR continues processing.

2 Certain errors that occur while IEAVTSLP, IEAVTSL2 or IEAVTADR is processing are anticipated; these include:

- A page fault that occurs while IEAVTSLP is examining or retrieving the instruction that caused a PER interruption.
- A page fault that occurs while IEAVTSL2 is retrieving user-defined data.
- A page fault that occurs while IEAVTADR is processing.
- A page fault that occurs while IEAVTSL1 is examining the instruction that caused a PER interrupt in the ASIDSA subroutine.
- A CMSET SSARTO macro fails in the DATA subroutine of IEAVTSL2.
- A CMSET SSARTO macro fails in the ADRCMSET subroutine of IEAVTSLS.

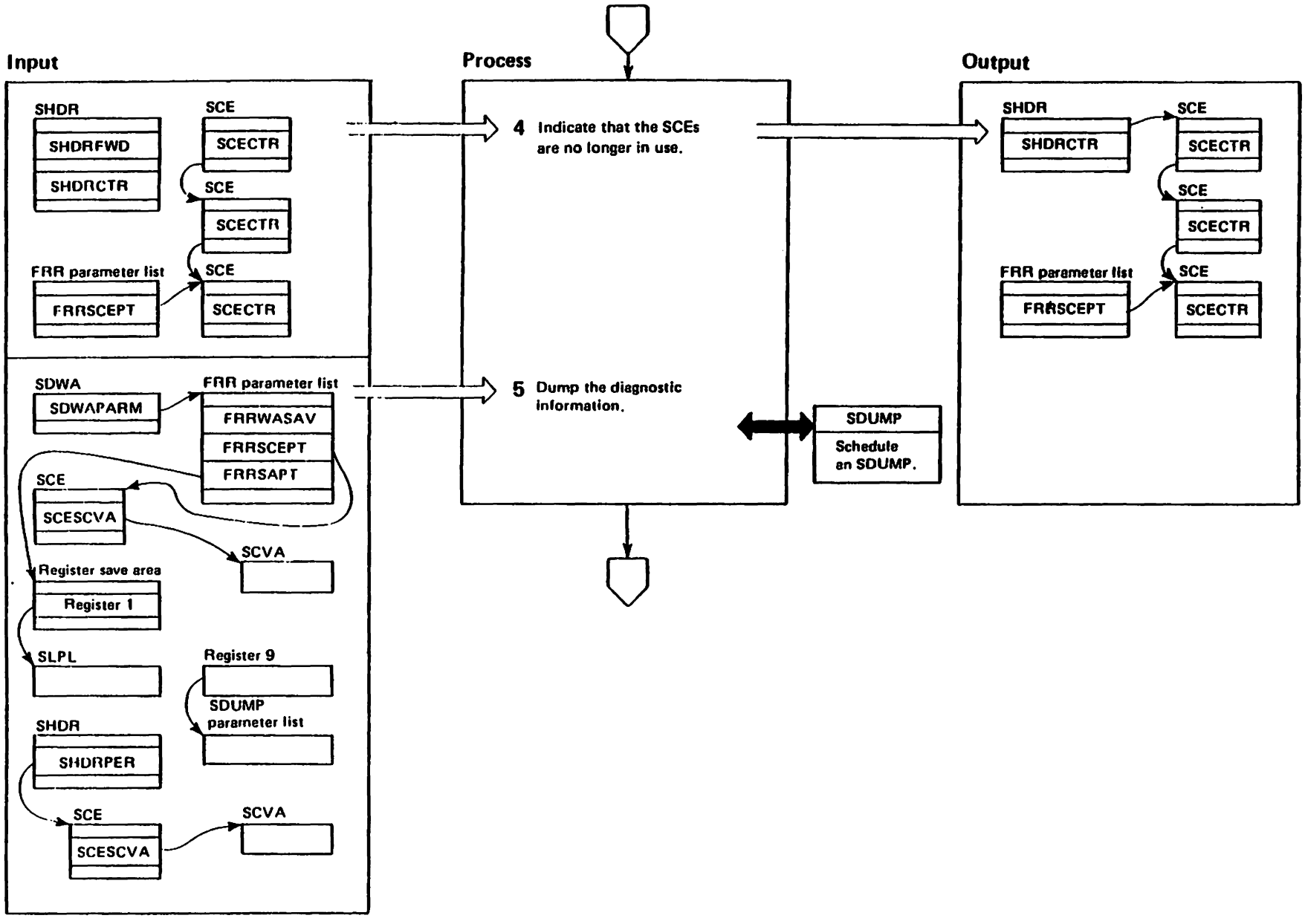
Extended Description **Module** **Label**

Each of these errors has a footprint bit in the AUDITWRD field of the FRR parameter list. If any bit is one and a retry is possible (the SDWACLUP and SDWARPIV bits in the SDWAERRD field equal zero), IEAVTSLR retries at an appropriate point in IEAVTSLP, IEAVTSL1, IEAVTSL2, IEAVTSLS, or IEAVTADR. These errors are not recorded on the SYS1.LOGREC data set. IEAVTSLR requests a retry and returns to IEAVTRTS.

3 For errors that are not anticipated, IEAVTSLR:

- Enters the module, CSECT, and FRR names in the SDWA for subsequent recording.
- Releases the restart lockword (if held), and restores floating point register zero (if it was in use when the error occurred.)
- If SLIP obtained the local lock, this module updates the SDWA to cause RTM to free the lock if percolation occurs, but not if a retry is attempted.
- If IEAVTRTS called IEAVTSLP, this module adjusts the FRR stacks as required.

IEAVTSLR – SLIP Processor Recovery Routine (Part 3 of 6)



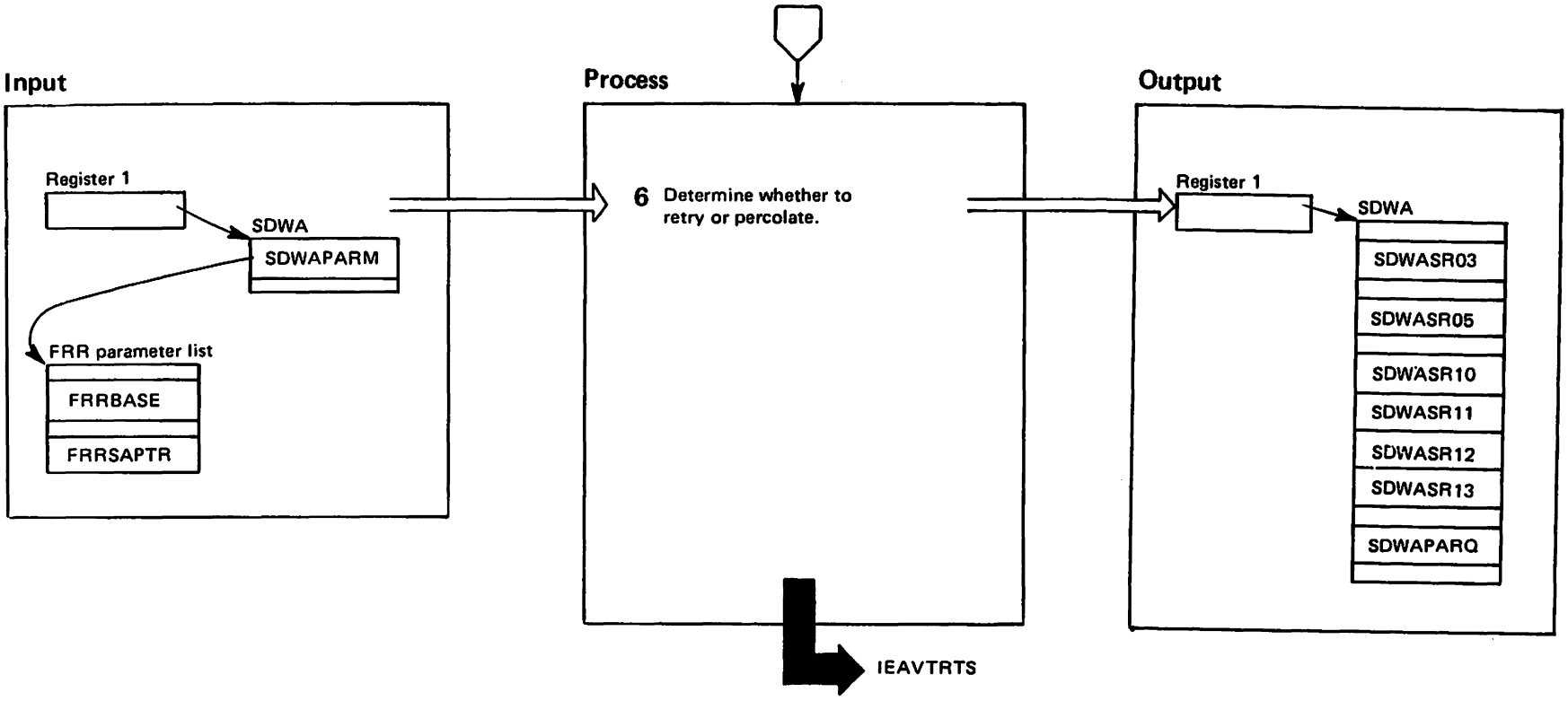
Extended Description	Module	Label

4 To indicate the SCE chain is no longer serialized, IEAVTSLR subtracts one from the use counts in the SCE pointed to by the FRRSCEPT field, and in each of the preceding SCEs on the chain.

5 If a lower level recovery routine has not already taken a dump (SDWAEAS=0), IEAVTSLR builds an SDUMP parameter list and requests a full dump of the diagnostic information concerning this error. It also requests a summary dump containing information relevant to an error in SLIP, normally including:

- The FRR parameter list for IEAVTSLR.
- The SCE and SCVA being processed at the time of the error.
- The SLIP parameter list and associated work areas.
- If a PER interruption was being processed at the time of the error, the SCE and SCVA representing the enabled non-IGNORE PER trap.

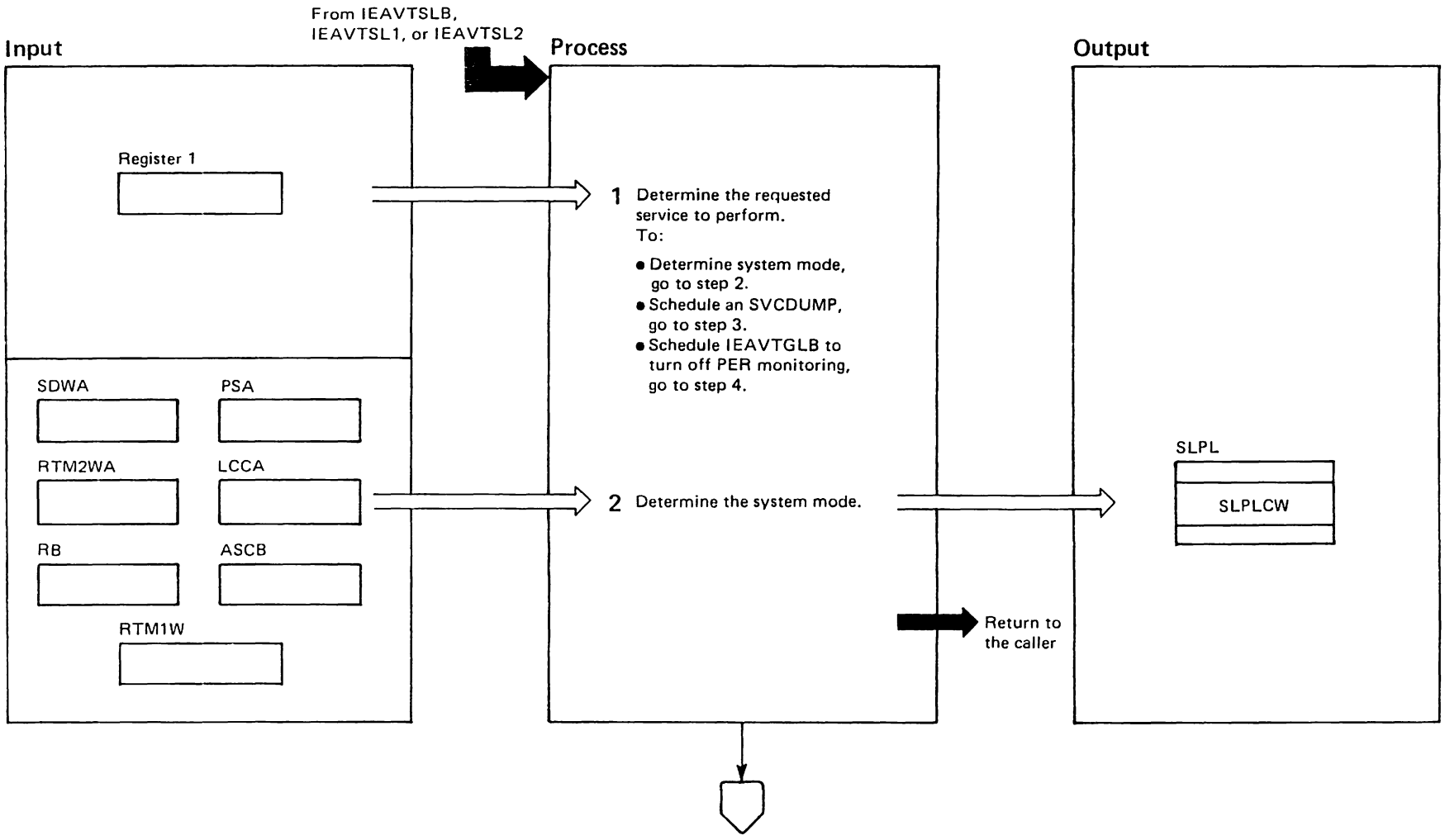
Additional system information is also available in the summary and full dumps.



IEAVTSLR – SLIP Processor Recovery Routine (Part 6 of 6)

Extended Description	Module	Label
6 If a retry is possible (the SDWACLUP bit in the SDWAERRD field equals 0), IEAVTSLR refreshes the registers used in the IEAVTSLE exit code and requests a retry at a point near the end of normal SLIP processing. If a retry is not possible (SDWACLUP=1), IEAVTSLR requests percolation.		

IEAVTSLS – SLIP Processor Service Routine (Part 1 of 4)

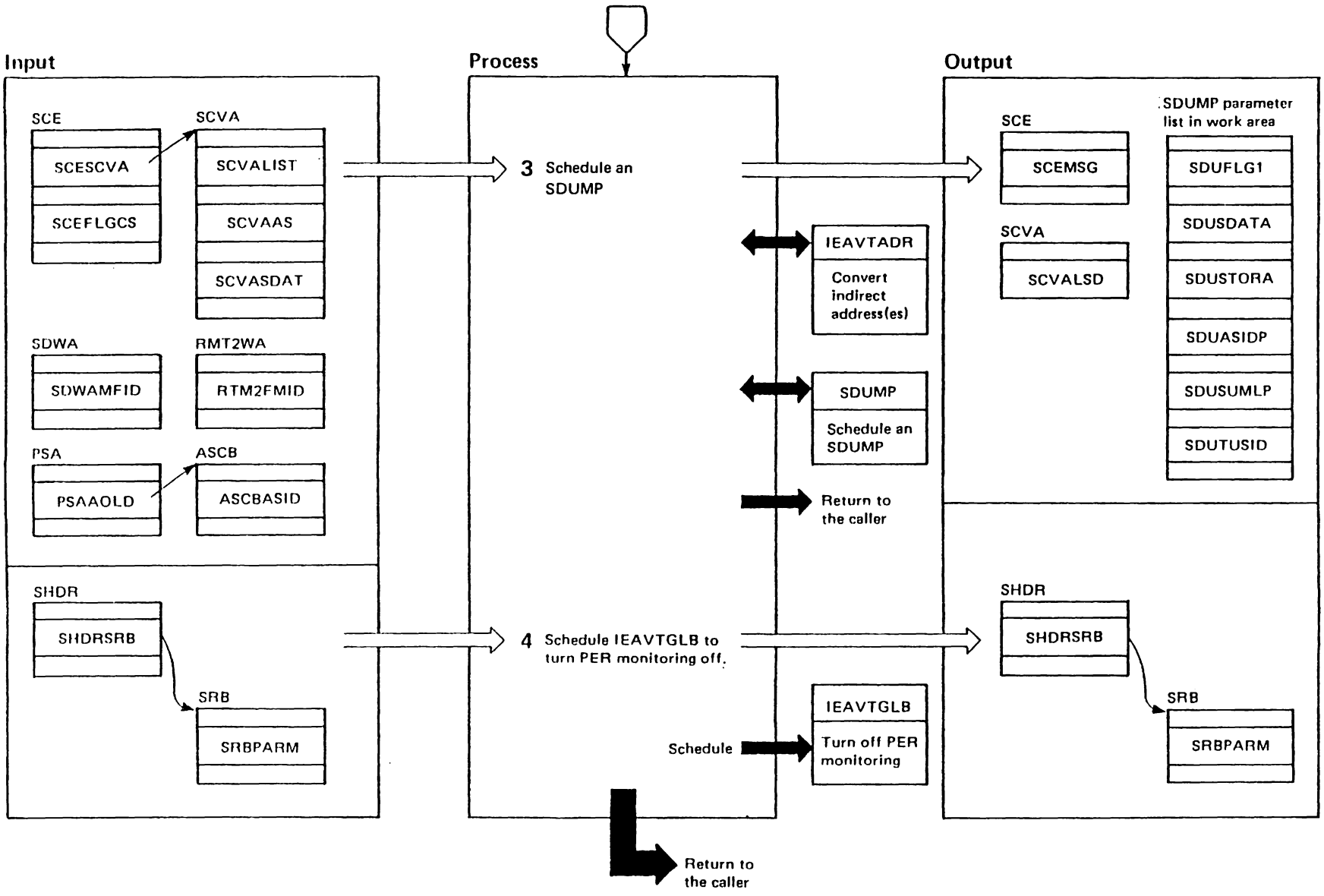


IEAVTSLS – SLIP Processor Service Routine (Part 2 of 4)

Extended Description	Module	Label
1 IEAVTSLS uses the contents of register 1 to determine the requested service and branches to the appropriate routine.	IEAVTSLS	

Register 1	*Service*
1	Determine system mode
2	Schedule an SVCDUMP
3	Schedule IEAVTGLB to turn off PER monitoring

2 The DETRMODE subroutine refers to the various system control blocks shown in the input to determine the current mode of the system. (Which control blocks are examined depends on SLIP's caller.)		DETRMODE



IEAVTSLS – SLIP Processor Service Routine (Part 4 of 4)

Extended Description	Module	Label
3 To schedule an SDUMP, IEAVTSLS:	IEAVTSLS	DUMPIT
• Builds an SDUMP parameter list in the work area pointed to by the SLPLLSTW field, using the default SDUMP parameters.		
• Calls the internal subroutine CHNGDUMP, which modifies the SDUMP parameter list to correspond with the type of dump requested by the SLIP user in the SDATA, ASIDLST, LIST and/or SUMLIST keyword parameters.		CHNGDUMP
The LIST and SUMLIST keywords can specify address space qualifiers with the addresses supplied. CHNGDUMP calls the internal subroutine ADRCMSET to convert any symbolic ASIDs (CU=current, P=primary, S=secondary, H=home) into explicit ASIDs for the SDUMP parameter list and to issue the CMSET SSARTO macro to these address spaces.		
The LIST and SUMLIST keywords require that indirect address(es) be converted to direct address(es). CHNGDUMP calls IEAVTADR to do this conversion. If a dump is in progress (RTCTSDPL≠0) or the internal SLIP resource(s) are not available (SCVALSIB≠0), an SDUMP cannot be scheduled. Instead, CHNGDUMP returns to DUMPIT with an indicator that no dump can be taken. DUMPIT adds one to the IEA412I message counter (SCEM412). The SLIP command processor communication routine (IEECB905) is posted later in IEAVTSLE to initiate message processing. IEECB905 issues message IEA412I, informing the SLIP user that no dump was scheduled. (See the diagram and extended description of IEECB905 in the section, "Command Processing.")	IEAVTADR IEAVTSLS	DUMPIT
If the ASIDLST keyword is specified, the ASIDLIST subroutine processes the ASIDLST records by building a parameter list of ASIDs for SDUMP. It is necessary for IEAVTSLS to identify SLIP's caller (IEAVTRTM, IEAVTRTS, IEAVTRT2, or IEAVTPER) so that IEAVTSLS and SDUMP will use the proper cross memory environment (the PSW S-bit, PASID, and SASID). Using an input list of ASIDs, some of which might be symbolic, IEAVTSLS builds the SDUMP parameter list. In the input list, each ASID is identified by a code number (current=0, PASID=1,		ASIDLIST

Extended Description	Module	Label
SASID=2, home=3, explicit ASID=4, and LLOC=5). If the input ASID is a symbolic ASID, IEAVTSLS translates it into an explicit ASID before copying it into the SDUMP parameter list.		
After CHNGDUMP successfully completes, DUMPIT refers to the SDUMP parameter list (SDUASIDP) to determine if the ASIDLST keyword was used to specify a list of ASIDs. If so, SDUMP is requested to dump the corresponding address space(s). If the ASIDLST keyword was not specified, the address space to be dumped depends on the caller:		DUMPIT

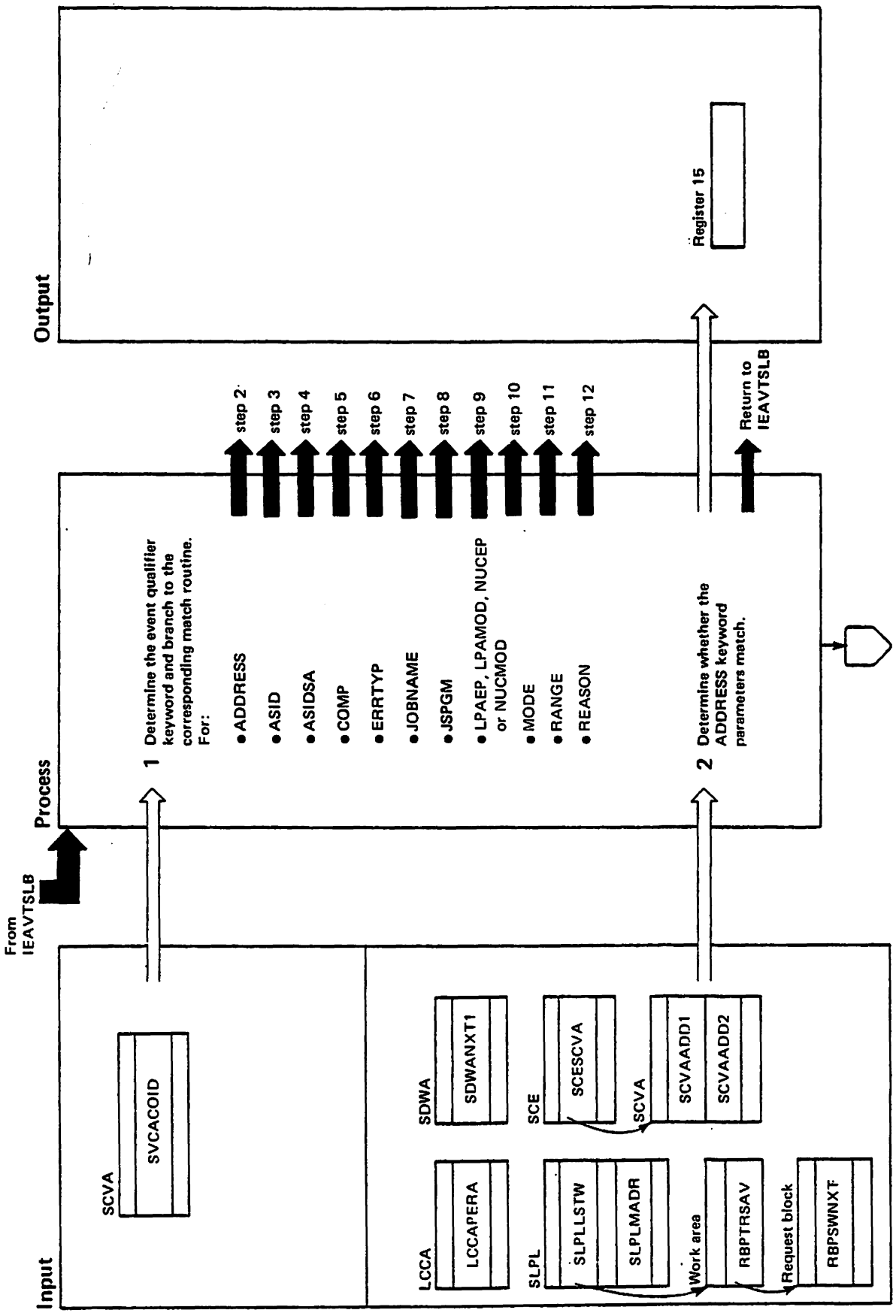
Caller	*Address space to be dumped*
IEAVTPER	ASCBASID, the current address space
IEAVTRTS	SDWAFMID, the failing address space, or ASCBASID
IEAVTRT2	RTM2FMID, the failing address space, or ASCBASID
IEAVTRTM	The master address space (ASID=1)

DUMPIT issues the appropriate form of the executable SDUMP macro. The SDUMP return code is examined to determine the results of the dump request.

If a dump was not scheduled, the action taken depends on the type of dump requested. If a dump of a failing address space was requested, DUMPIT attempts to schedule a dump in the home address space instead. If the failing dump request (either the first or second) was for the current address space or for a dump of the address space(s) specified on the trap, DUMPIT adds one to the SCEM412 message counter so the SLIP command processor communication routine (IEECB905) will issue message IEE412I.

4 To schedule IEAVTGLB to turn PER monitoring off, IEAVTSLS:		SCHEDGLB
• Sets the IEAVTSLP function request bit (in SRBPARM) to one in the SLIP SRB.		AGAIN19
• Schedules IEAVTGLB as a global SRB if the SRB is available (that is, IEAVTGLB is not already scheduled).		
• Sets the return code in the SLIP workarea to 8, which causes the interrupted process to be resumed with PER interruptions disabled.		

IEAVTSL1 - SLIP TRAP MATCHING ROUTINE PART 1



IEAVTSL1 – SLIP Trap Matching Routine Part 1 (Part 1 of 10)

IEAVTSL1 – SLIP Trap Matching Routine Part 1 (Part 2 of 10)

Extended Description

IEAVTSL1 receives control from IEAVTSLB to determine whether the trap conditions specified on the SLIP command match the current system conditions. Each SLIP event qualifier keyword has a corresponding subroutine in IEAVTSL1 or IEAVTSL2, which determines whether the event qualifier keywords.) IEAVTSL1 processes the ADDRESS, ASID, ASIDSA, COMP, ERRTYP, JOBNAME, JSPGM, LPAEP, LPAMOD, MODE, NUCEP, NUCMOD, RANGE, and REASON event qualifier keywords. IEAVTSL1 passes keyword parameters in the SCVA. IEAVTSL1 compares the event specified in the keyword with the current system condition, and sets a return code in register 15 to indicate a match (0) or no-match (4) condition.

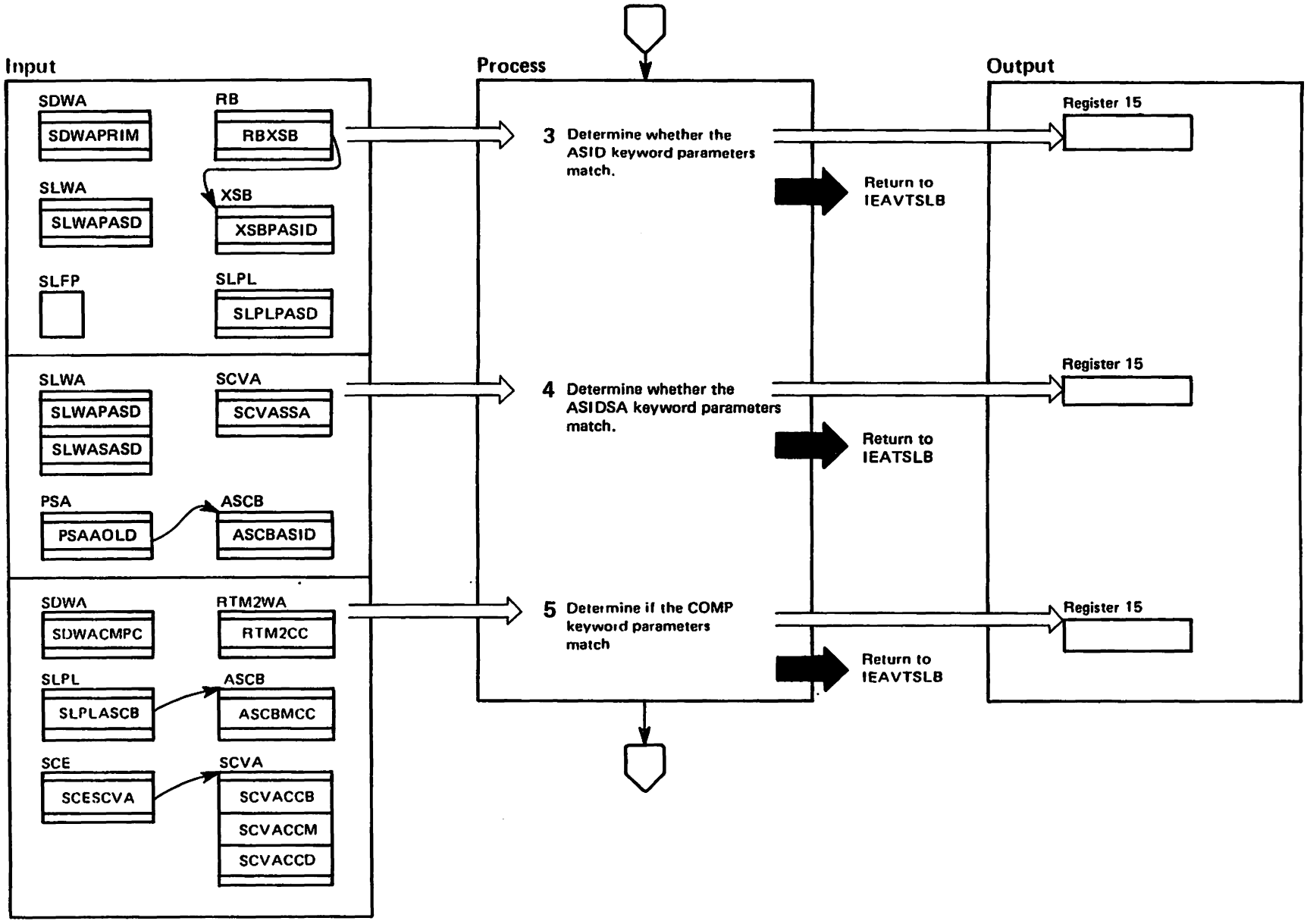
Module **Label**

1 IEAVTSL1 determines which SLIP event qualifier keyword was specified and branches to the corresponding match routine.

2 ADDRESS locates the address to be compared and determines if it falls within the address range specified on the trap (SCVAADD1 and SCVAADD2). The address used in the comparison depends on the caller. IEAVTSL1 ADDRESS

Caller	*Address field to be compared*
IEAVTRTS	SDWANXT1, the address taken from the PSW at the time of the error
IEAVTRT2	RBPSWNXT in the request block selected by the RBLEVEL processing. (See step 1 of the IEAVTSLB – SLIP Action Processor – Part 2 diagram.)
IEAVTRTM	SLPLMADR, the address of the CALLRTM BALR instruction that requested abnormal address space termination
IEAVTPER	LCCAPERA, the address of the instruction causing the PER interruption

IEAVTSL1 – SLIP Trap Matching Routine Part 1 (Part 3 of 10)



Extended Description	Module	Label
3 The ASID subroutine locates the ASID to be compared, then determines if it matches any specified on the trap (SCVAASD). The ASID used in the comparison depends on the caller:	IEAVTSL1	ASID

Caller	*ASID to be compared*
IEAVTRTS	SDWAPRIM (the PASID of the failing address space)
IEAVTRT2	XSBPASID (the PASID in the XSB of the RB located from the RBLEVEL keyword)
IEAVTRTM	SLPLPASD (the PASID of the address space being terminated)
IEAVTPER	SLWAPASD (the PASID of the address space at a time of the PER interruption)

If the ASIDs match, IEAVTSL1 returns a match condition (return code=0).

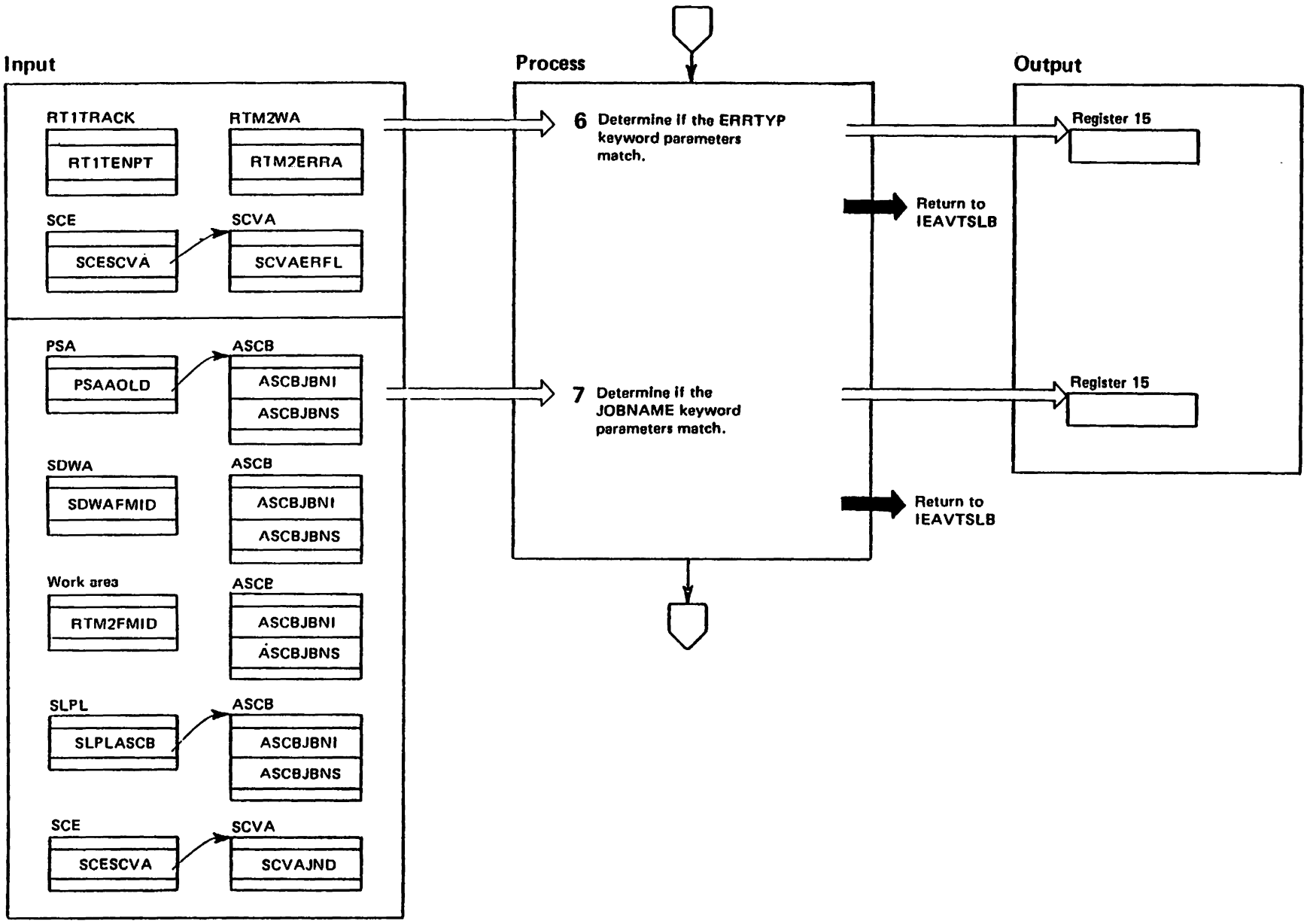
4 The ASIDSA subroutine determines in what address space the storage alteration (SA) occurred, then determines whether this address space matches any of the address spaces specified with the ASIDSA keyword. If a match is found, IEAVTSL1 returns a match condition to the caller (IEAVTSLB). — IEAVTSL1 ASIDSA

If a page fault occurs while the ASIDSA subroutine is checking the instruction that caused the PER interrupt, SLIP's FRR (IEAVTSLR) receives control. IEAVTSLR has RTM reenter the ASIDSA subroutine at SL1SFAIL to return a no match condition to the caller.

5 The COMP subroutine determines which completion code to compare, then checks whether it matches the trap's description of allowable completion code(s) (the SCVACOMP field). The completion code compared depends on the caller: — IEAVTSL1 COMP

Caller	*Completion code field*
IEAVTRTS	SDWACMPC
IEAVTRT2	RTM2CC
IEAVTRTM	ASCBMCC in the ASCB pointed to by the SLPLASCB field (the ASCB being terminated)
IEAVTPER	Not applicable

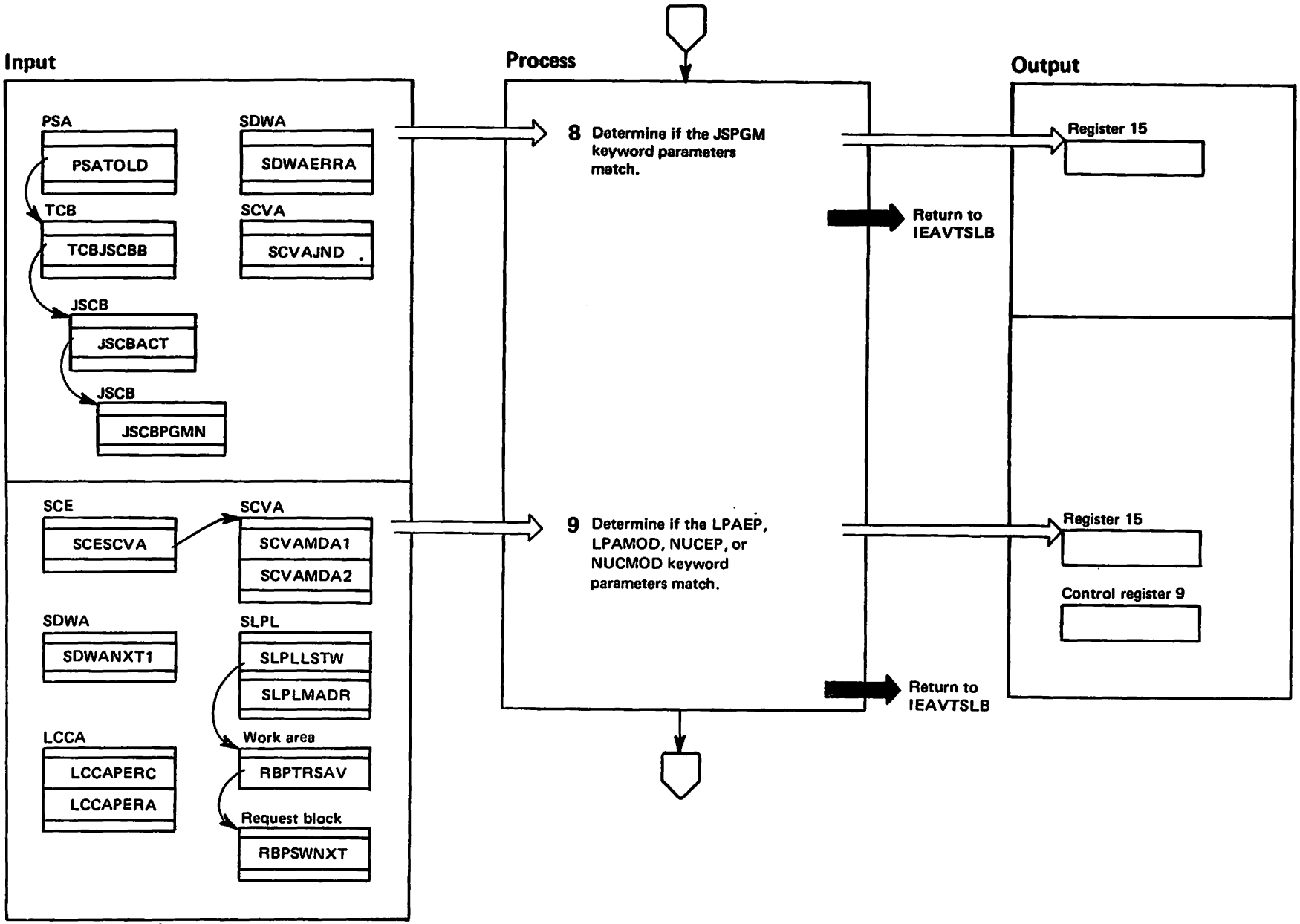
IEAVTSL1 – SLIP Trap Matching Routine Part 1 (Part 5 of 10)



Extended Description	Module	Label
6 The field that the ERRTYP subroutine compares with the error type(s) specified in the trap (SCVAERFL field) depends on the caller. When entry is from IEAVTRTS, the ERRTYP subroutine uses the error that caused entry to RTM (RT1TENPT). For a match condition to exist: • The RT1TENPT field must match one of the error conditions specified in the trap, or • RT1TENPT must indicate an SVC error occurred, an SVC 13 (ABEND) must have been issued, and the trap must specify ABEND as an error type. If SLIP's caller is IEAVTRT2, the ERRTYP subroutine compares the error that caused entry to RTM (RTM2ERRA). A match results if RTM2ERRA matches any error type indicated in the SCVAERFL field, except abnormal address space termination. However, if the RTM2ERRA field indicates an abnormal address space termination error, and the trap specified ABEND as an error type, the result is also a match. If the caller is IEAVTRTM, the ERRTYP subroutine indicates a match only if the trap specified abnormal address space termination as the error type. Since a PER interruption is not an error condition, the ERRTYP subroutine is never called when processing a PER interruption.	IEAVTSL1	ERRTYP

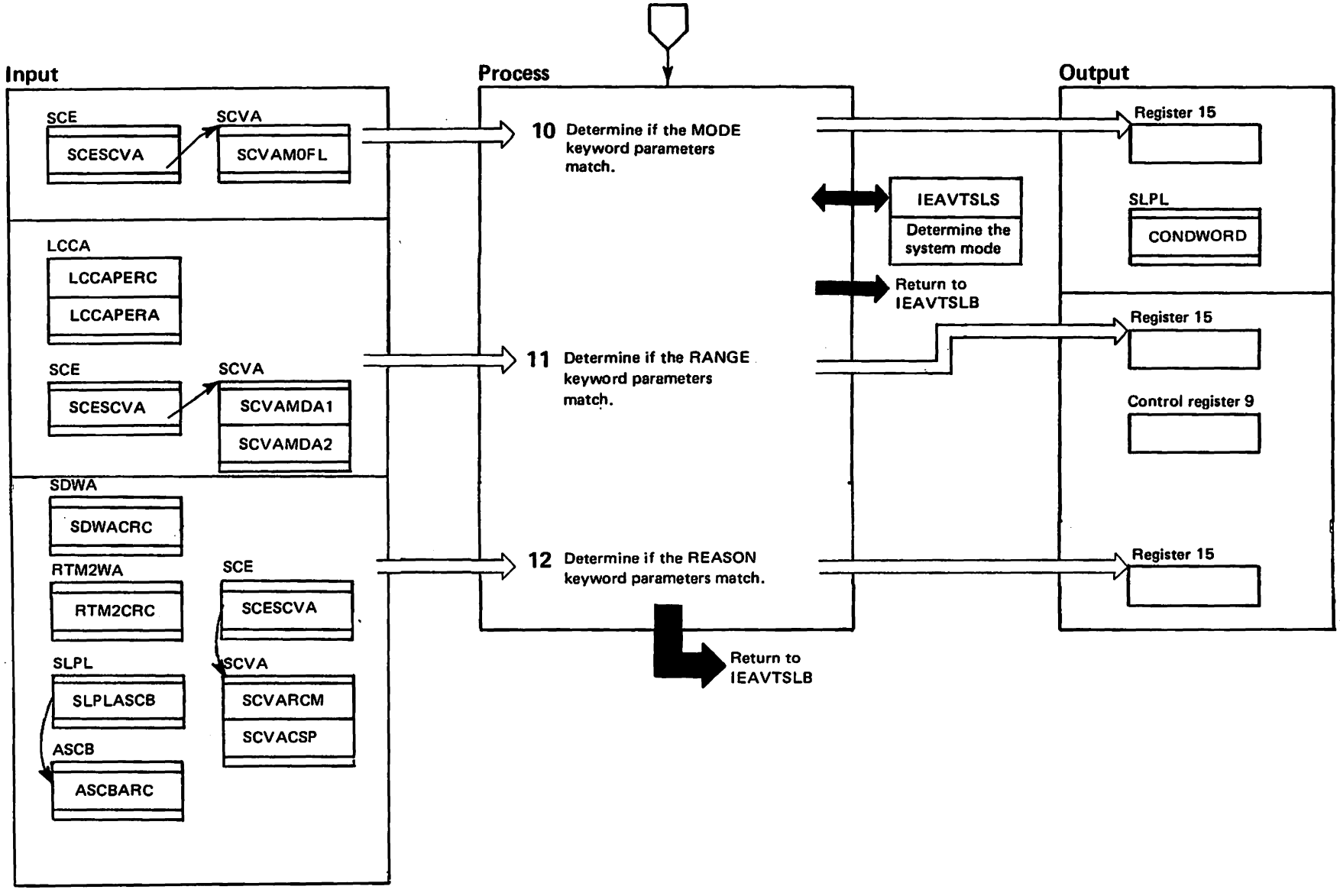
Extended Description	Module	Label
7 The JOBNAME subroutine checks for a match between the jobname pointed to by the ASCBJBNI or ASCBJBNS fields in various ASCBs and the jobname specified on the SLIP trap (the SCVAJND field). The environment determines which address space(s) is examined for a jobname match: *Caller* — *ASCBs used in the comparison* IEAVTRTS — If a failing address space has been identified and is not current (SDWAFMID≠0 and SDWAFMID≠ ASCBASID), its jobname(s) and the jobnames(s) from the current address space. Otherwise, IEAVTSL1 uses only the jobname(s) from the current address space. IEAVTRT2 — If a failing address space has been identified and is not current (RTM2FMID≠0 and RTM2FMID≠ASCBASID), its jobname(s) and the jobname(s) from the current address space. Otherwise, IEAVTSL1 uses only the jobname(s) from the current address space. IEAVTRTM — The jobname(s) from the address space pointed to by the SLPLASCB field (the address space being terminated). IEAVTPER — The jobname(s) from the current address space.	IEAVTSL1	JOBNAME

IEAVTSL1 – SLIP Trap Matching Routine Part 1 (Part 7 of 10)



Extended Description	Module	Label
8 If SLIP was called by IEAVTRTS, IEAVTRT2, or IEAVTPER, the JSPGM subroutine determines if the jobstep program name (JSCBPGMN) matches the name specified in the trap (SCVAJND). A no-match condition results when: • JSCBPGMN≠SCVAJND. • IEAVTRTM called SLIP. IEAVTSL1 cannot obtain a jobstep program name from the terminating address space. • IEAVTRTS called SLIP while processing a DAT, error (SDWATEXC=1). IEAVTSL1 cannot obtain a jobstep name when this type of error occurs. • IEAVTSL1 cannot test a jobstep name because either the PSATOLD or the TCBJSCBB field equals zero.	IEAVTSL1	JSPGM
9 The main function of the LPANUC subroutine is to determine whether a given address (the load module address, the entry point address, or an alternate entry point address, i.e. alias) falls within the address range specified in the SLIP trap (SCVAMDA1 and SCVAMDA2). The address used in comparison depends on the caller: *Caller* — *Address* IEAVTRTS — SDWANXT1, the address from the PSW at the time of the error IEAVTRT2 — RBPSWNXT field in the request block selected by the RBLEVEL processing done in step 1 IEAVTRTM — SLPLMADR, the address of the CALLRTM BALR instruction IEAVTPER — LCCAPERA, the address of the instruction causing the PER interruption When processing a non-PER trap, a PER trap for storage alteration (SA) or instruction fetch (IF) events, or a PER trap with the IGNORE action specified, the LPANUC subroutine returns after determining if a match condition exists. If an enabled non-IGNORE PER trap for a successful branch (SB) event is being examined, the LPANUC subroutine does further checking to determine if additional processing is necessary to limit SB monitoring to the range specified. To do this, the LPANUC subroutine manipulates the type of PER monitoring performed when: • A match condition was found, the trap is for SB events, and an IF interruption occurred. This situation indicates that the specified range has been entered. The LPANUC subroutine turns off IF PER monitoring and turns on SB PER monitoring by manipulating the SB and IF bits in control register 9. • A no-match condition was found and an SB interruption occurred. Because the instruction was executed outside the specified range, the LPANUC subroutine switches PER monitoring back to IF mode.	IEAVTSL1	LPANUC

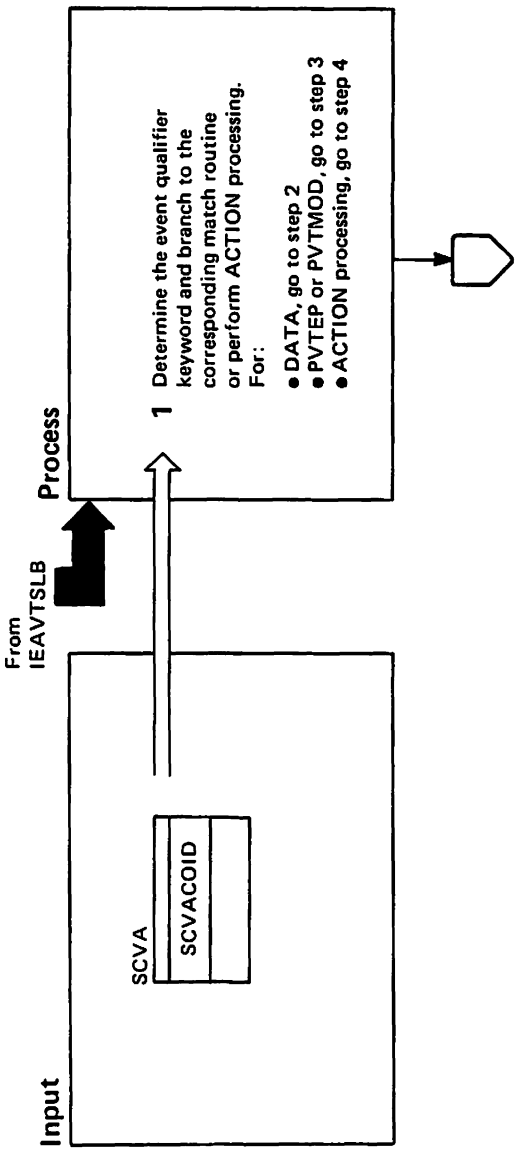
IEAVTSL1 – SLIP Trap Matching Routine Part 1 (Part 9 of 10)



Extended Description	Module	Label
10 The MODE subroutine calls IEAVTSLS to determine the current mode of the system. (Which control blocks are examined depends on IEAVTSLP's caller.) MODE then compares the current mode with the mode specified in the trap (SCVAMOFL).	IEAVTSL1	MODE
11 The RANGE subroutine is called only when processing a PER interruption. RANGE determines if the address of the instruction that caused the PER interruption (LCCAPERA) falls within the address range specified on the SLIP trap (SCVAMDA1 and SCVAMDA2). If so, the keyword matches. When processing a non-PER trap, a PER trap for storage alteration (SA) or instruction fetch (IF) events, or a PER trap with the IGNORE action specified, the RANGE subroutine returns after determining if a match condition exists. If an enabled non-IGNORE PER trap for a successful branch (SB) event is being examined, the RANGE subroutine does further checking to determine if additional processing is necessary to limit SB monitoring to the range specified by the RANGE subroutine parameters. To do this, the RANGE subroutine manipulates the type of PER monitoring performed when: • A match condition was found, the trap is for SB events, and an IF interruption occurred. This situation indicates that the specified range has been entered. The RANGE subroutine turns off IF PER monitoring and turns on SB PER monitoring by manipulating the SB and IF bits in control register 9. • A no-match condition was found and an SB interruption occurred. Because the instruction was executed outside the specified range, this subroutine switches PER monitoring back to IF mode.		RANGE

Extended Description	Module	Label
12 The REASON keyword allows a reason code to be specified on non-PER traps. The REASON subroutine determines whether there is a reason code and, if so, checks whether it matches the trap's description of allowable reason code(s) by examining the SCVAREAS field. The reason code compared depends on the caller: *Caller* — *Reason code field* IEAVTRTS — SDWACRC IEAVTRT2 — RTM2CRC IEAVTRTM — ASCBARC *Note:* The REASON keyword cannot be specified without the COMP keyword. If it is or if the REASON keyword is specified on a PER trap or if the REASON keyword syntax is invalid, IEECB909 issues a message.		REASON

IEAVTSL2 - SLIP TRAP MATCHING ROUTINE PART 2



IEAVTSL2 – SLIP Trap Matching Routine Part 2 (Part 1 of 6)

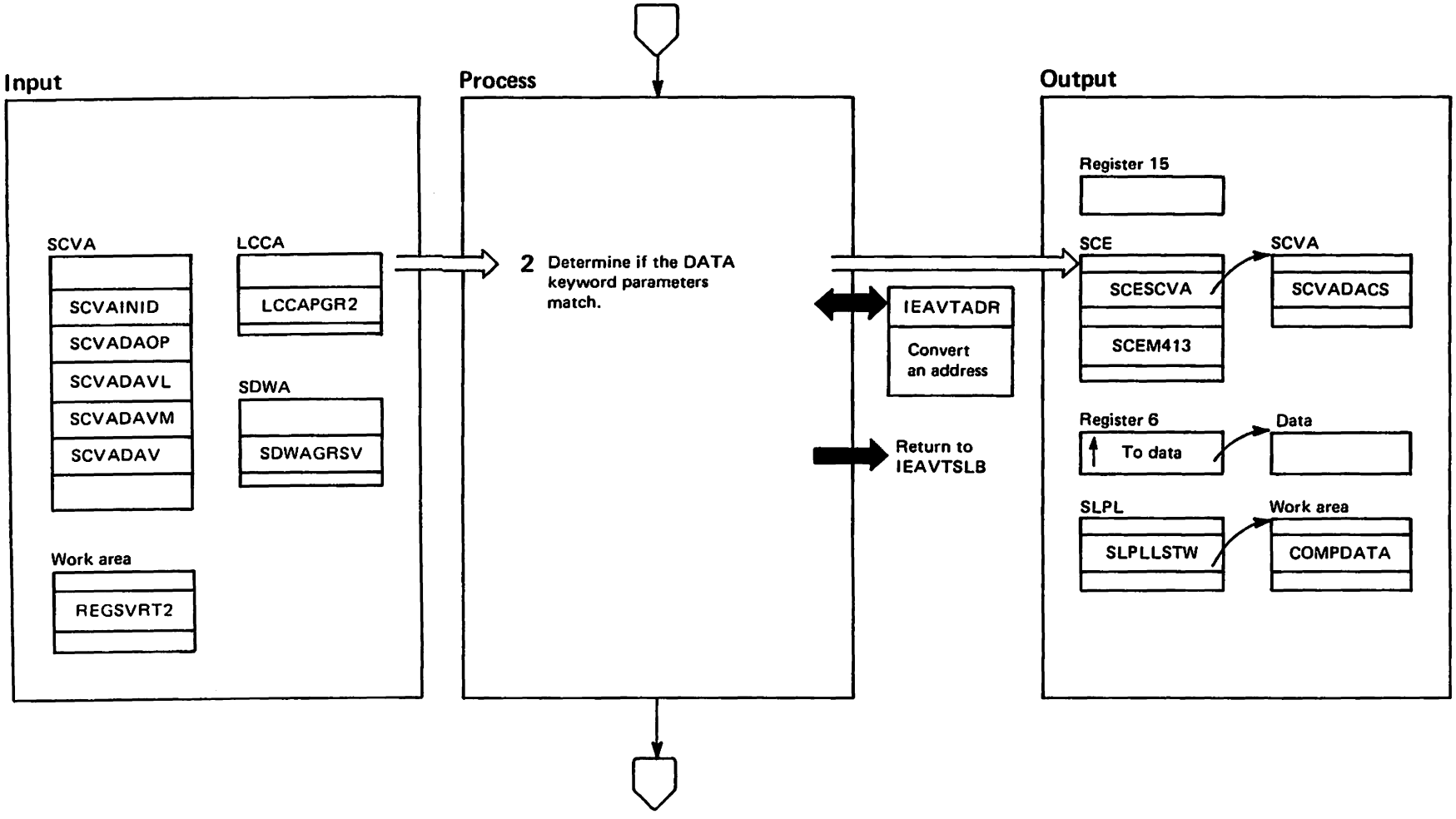
IEAVTSL2 – SLIP Trap Matching Routine Part 2 (Part 2 of 6)

Extended Description **Module** **Label**

IEAVTSL2 receives control from IEAVTSLB to determine whether the trap conditions specified on the SLIP command match the current system conditions. Each SLIP event qualifier keyword (see *System Commands* for a description of event qualifier keywords) has a corresponding subroutine in IEAVTSL1 or IEAVTSL2, which determines whether the event described by the keyword matches the current system condition. IEAVTSL2 processes the DATA, PVTEP, and PVTMOD event qualifier keywords. Keyword parameters are passed in the SCVA. IEAVTSL2 compares the event specified in the keyword with the current system condition, sets a return code in register 15 to indicate a match (0) or no-match (4) condition, and updates the SCVAPTR to point to the next record. IEAVTSL2 also processes the action requested by the trap's ACTION parameters. This processing is described in diagram IEAVTSL2 – SLIP Trap Matching Routine – Part 2 – ACTION Keyword Processing.

1 IEAVTSL2 determines which SLIP event qualifier keyword was specified and branches to the corresponding match routine.

IEAVTSL2 – SLIP Trap Matching Routine Part 2 (Part 3 of 6)



IEAVTSL2 – SLIP Trap Matching Routine Part 2 (Part 4 of 6)

Extended Description | **Module** | **Label**

2 The DATA subroutine determines whether the value in a given data area or register compares successfully with the data on the SLIP trap. The process is described in five steps, a - e. — IEAVTSL2 — DATA

a. The DATA subroutine refers to the SCVAINID field to determine the location of the data to be compared with the trap data. Possibilities are a storage location referred to by either a direct virtual address, an indirect address, or a register. For a storage comparision, IEAVTSL2 issues a CMSET SSARTO macro to the address space associated with the ASID qualifier of the DATA keyword. If IEAVTSL2 is already running in this address space, it does not issue the CMSET SSARTO macro. If the comparison involves a storage location referenced by a direct address, the DATA subroutine loads the address into register 6. If an indirect address is used, DATA calls IEAVTADR to convert it to a direct address. If unsucessful, IEAVTADR sets a nonzero return code. DATA considers this a no-match condition and continues processing at step d. If the indirect address is successfully converted, DATA places the converted address in register 6. If register contents are to be compared, DATA locates the requested register in the appropriate register save area: — IEAVTADR

IEAVTSLP caller	*Register save area used*
IEAVTPER	LCCAPGR2 field
IEAVTRTM	A register save area pointed to by the previous IEAVTRTM FRR parameter list
IEAVTRTS	SDWAGRSV field
IEAVTRT2	A register save area pointed to by the REGSVRT2 field in the request block determined by RBLEVEL processing (See step 1 of the IEAVTSLB – SLIP Action Processor – Part 2 diagram.)

If the registers are unavailable (a possibility only when IEAVTRTS or IEAVTRT2 is SLIP's caller), the DATA subroutine treats this as a no-match condition and continues processing at step d. Otherwise, it loads the pointer to the saved register into register 6. At this point, register 6 contains the address of the data to be compared.

b. If the data to be compared is a storage location and SLIP's caller is IEAVTRTS, IEAVTRTM, or IEAVTPER, the DATA subroutine uses an LRA (load real address) instruction to determine if the requested storage location is available. This is done to avoid page fault processing if the data is paged out. If the LRA instruction indicates the data is not available, the DATA subroutine indicates a no-match condition and continues processing at step d. Otherwise, IEAVTSL2 considers the data available and match processing continued.

When IEAVTRT2 is SLIP's caller, IEAVTSL2:

- Establishes the address space associated with the ASID qualifier of the DATA keyword as the primary and secondary address space by issuing a CMSET SET macro.
- Enables the PSW for I/O and external interrupts so that page faults can be tolerated.
- Copies the requested data into a page-fixed SLIP work area.
- Issues a CMSET SET Macro to the home address space, followed by a CMSET SSARTO macro to establish the address space associated with the ASID qualifier of the DATA keyword as the secondary address space.

c. The DATA subroutine copies the requested data into a page-fixed SLIP workarea. If the data is unavailable even though the LRA instruction test was successful (step b), processing continues at step d. If a data-to-data comparison is requested, steps a-c are repeated to fetch the second comparand. If a data-to-value comparison is requested, the DATA subroutine uses the value specified in the trap (the SCVADAV field) as the other comparand. The comparison is done using the comparison operator requested in the trap (the SVCADAOP field). If a match is found, the DATA subroutine continues processing at step e. If the data does not match, the DATA subroutine indicates a nomatch condition and continues processing at step e.

d. If the data is unavailable (it is paged out, the registers are not available, or IEAVTADR was unable to convert the indirect address), the DATA subroutine:

- Adds one to the data unavailable counter (SCVADACS), indicating the data for comparison could not be retrieved.
- Sets a message indicator bit (SCEM413) and the SHDRPSTM bit to 1, causing the IEA4131 message to be sent to the SLIP user. If the trap is a PER type, DATA sets these bits the first time it finds data unavailable.
- Indicates a no-match condition.

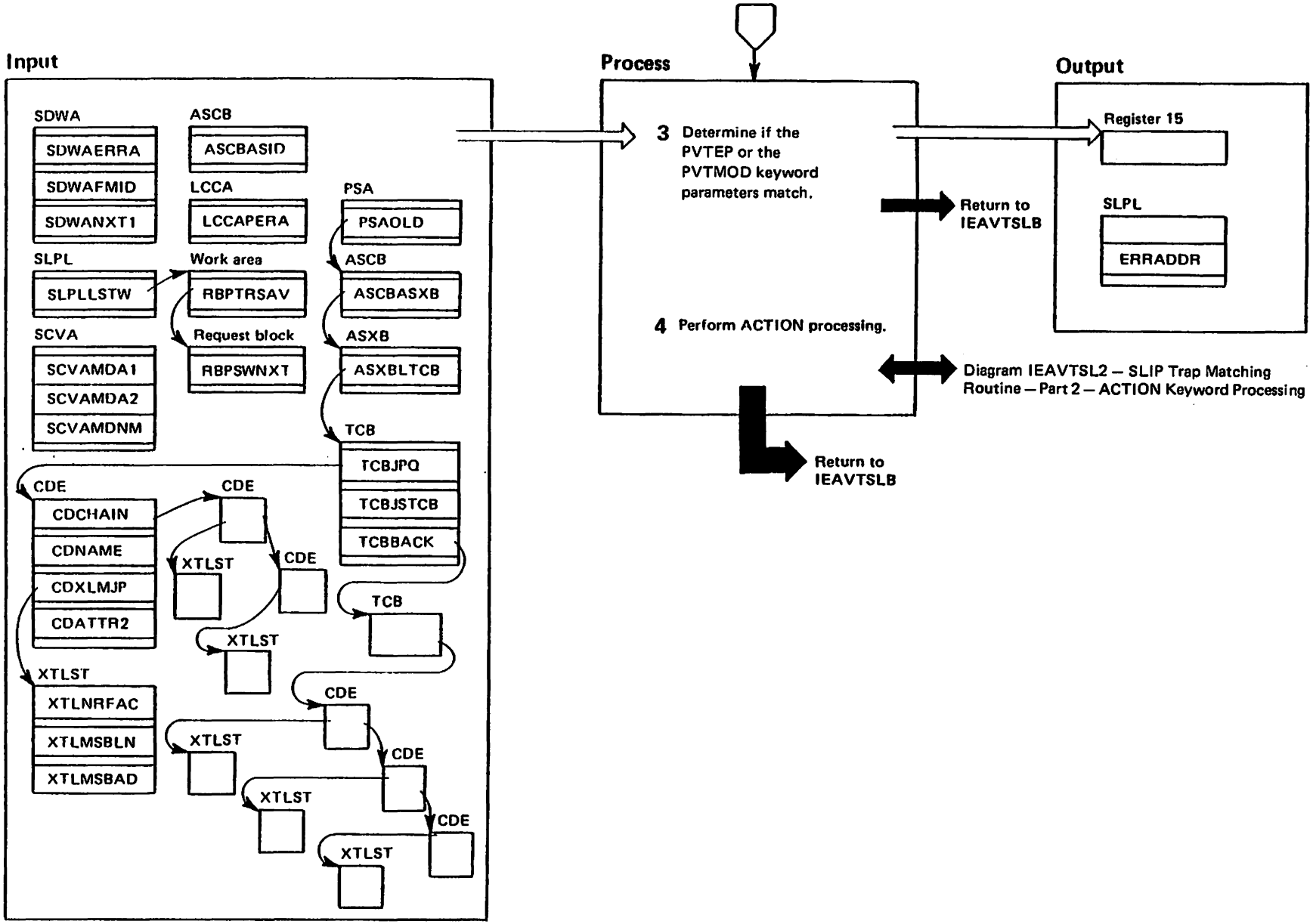
e. If no other DATA parameters have been specified, IEAVTSL2 performs the following:

- Issues a CMSET SET macro to restore the cross memory environment to home.
- Indicates either a match or no match condition in the return code.
- Returns to the caller.

If other DATA parameters have been specified, the processing flow depends on the next logical operator and the present comparison conditon.

Logical Operator	*Condition*	*Processing Flow*
AND	Match	Returns to step a
AND	No Match	Skips all subsequent DATA parameters within the current parentheses level and then returns to step e.
OR	Match	Skips all subsequent DATA parameters within the current parentheses level and then returns to step e
OR	No Match	Returns to step a

IEAVTSL2 – SLIP Trap Matching Routine Part 2 (Part 5 of 6)



IEAVTSL2 – SLIP Trap Matching Routine Part 2 (Part 6 of 6)

Extended Description	Module	Label
3 The PVTMOD subroutine cannot perform match processing when: • IEAVTRTM called SLIP. In this case, the failing address space's control blocks are not available for searching. • IEAVTRTS called SLIP and the failing address space is not current (SDWAFMID ≠ ASCBASID) or a DAT error is being processed. In either of these situations, the private area control blocks are not available for searching. If SLIP's caller is IEAVTRTS or IEAVTPER and a CML lock is not held, PVTMOD attempts to obtain the LOCAL lock. (If SLIP's caller is IEAVTRT2, IEAVTSLP obtained the LOCAL lock when establishing the recovery environment in diagram IEAVTSLP – SLIP Action Processor – Part 1.) If unsuccessful, PVTMOD indicates a no-match condition and returns. PVTMOD determines which address to use in the comparsion and places it in the ERRADDR field. The address used depends on the caller:	IEAVTSL2	PVTMOD

Caller	*Address*
IEAVTRTS	SDWANXT1, the address from the PSW at the time of the error
IEAVTRT2	RBPSWNXT field of the request block selected by the RBLEVEL processing. (See step 1 of the SLIP Action Processor – Part 2 (IEAVTSLB) diagram.)
IEAVTPER	LCCAPERA, the address of the instruction that caused the PER interruption

The PVTMOD subroutine performs the following processing:

- For a storage alteration event or when called by IEAVTRTS or IEAVTRT2 –

  Starting with the last TCB on the TCB chain, the PVTMOD subroutine searches the corresponding job step CDE chain for a module whose name matches the name specified in the trap, and whose beginning and ending addresses (as modified by any offsets specified in the trap) span the address in the ERRADDR field. The search continues through the CDE and TCB chains until a match is found or the relevent chains have all been searched. IEAVTSL2 indicates the results of the search in register 15. If PVTMOD obtained the local lock, PVTMOD releases it.

- For instruction fetch or successful branch monitoring –

  The PVTMOD subroutine determines whether the address in the LCCAPERA falls within the address range specified in the SLIP trap (SCVAMDA1 and SCVAMDA2).

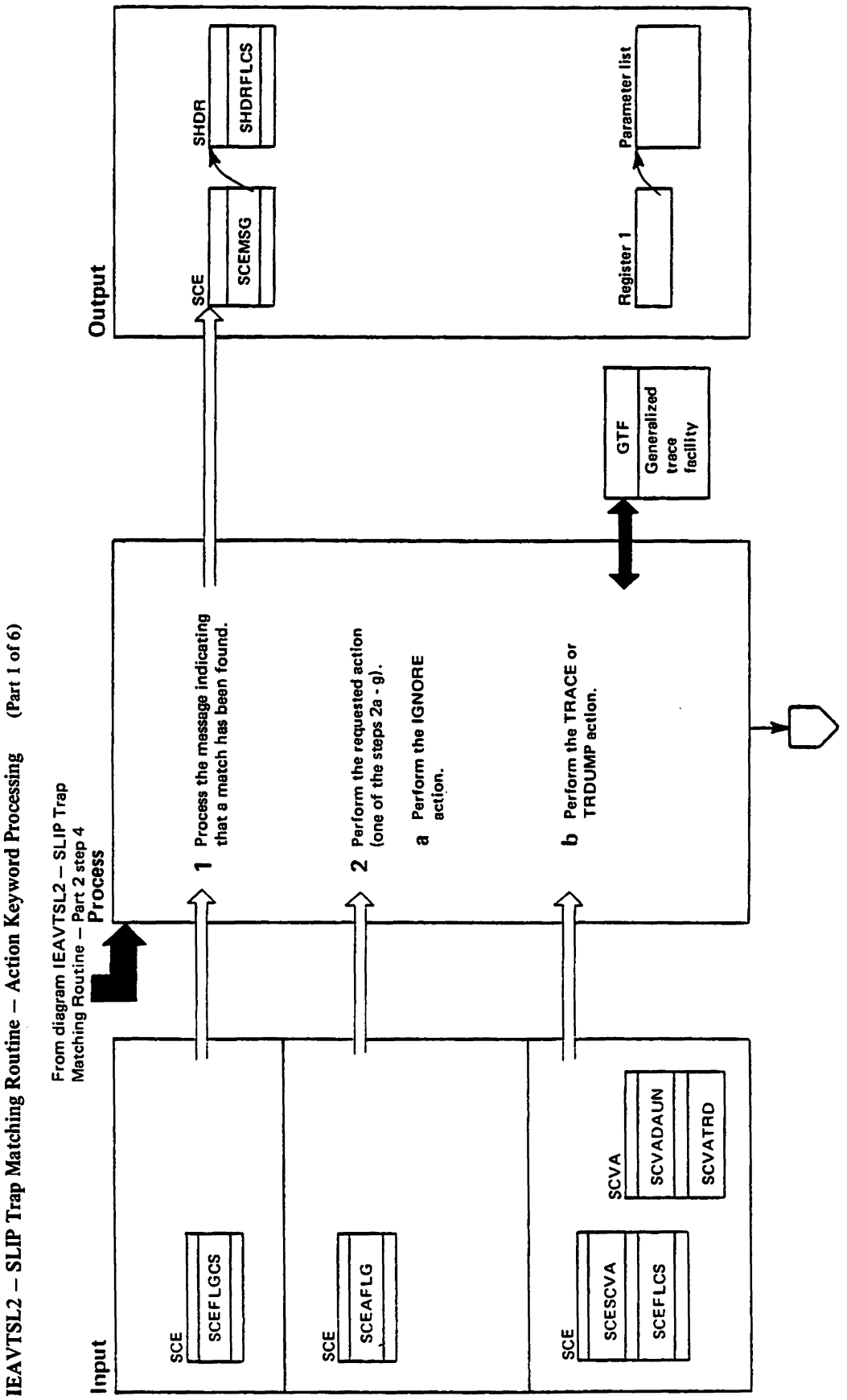
- For an enabled non-IGNRORE PER trap that has had a successful branch (SB) event –

  The PVTMOD subroutine determines if additional processing is necessary to limit SB monitoring to the range specified. To do this, the PVTMOD subroutine manipulates the type of PER monitoring performed when:

  - A match condition was found, the trap is for SB events, and an IF interruption occurred. This situation indicates that the specified range has been entered. The PVTMOD subroutine turns off IF PER monitoring by manipulating the SB and IF bits in control register 9.
  - A no-match condition was found and an SB interruption occurred. Because the instruction was executed outside the specified range, the PVTMOD subroutine switches PER monitoring back to IF mode.

4 When keyword match processing is completed, IEAVTSL2 branches to the EOL subroutine to perform the action requested by the trap's ACTION parameters. This processing is described in IEAVTSL2 – SLIP Trap Matching Routine Part 2. | | EOL |

IEAVTSL2 - SLIP TRAP MATCHING ROUTINE - ACTION KEYWORD PROCESSING



IEAVTSL2 – SLIP Trap Matching Routine – Action Keyword Processing (Part 2 of 6)

Extended Description	Module	Label
If all the event qualifier keywords specified on the trap have been compared and found to match the current system conditions, IEAVTSLB calls the EOL subroutine in IEAVTSL2 to take the action specified on the SLIP trap.	IEAVTSL2	EOL

1 When processing a non-IGNORE non-PER trap, or the first match for a non-IGNORE PER trap (SCEMATCH=0), EOL:

- Adds one to the IEA992I message counter (SCEM992).
- Turns on the SHDRPSTM bit, which causes the SLIP command processor communication routine (IEECB905) to be posted. IEECB905 issues message IEA992I, informing the SLIP user that a match has occurred. (See the diagram and extended description of IEECB905.)

2 To determine which action to take, IEAVTSL2 refers to bits in the SCEAFLG field.

Bit setting	*Action parameter*	*Go to step*
SCEIGNOR=1	IGNORE	a
SCETRACE=1 or SCETRDMP=1	TRACE or TRDUMP	b
SCENODMP=1	NODUMP	c
SCENOSVD=1 or SCENOSYA=1 or SCENOSYM=1 or SCENOSYU=1	NOSVCD or NOSYSA or NOSYSM or NOSYSU	d
SCENOSUP=1	NOSUP	e
SCEWAIT=1	WAIT	f
SCESVCD=1	SVCD	g
SCESTRCE = 1 or SCESTDMP = 1	STRACE or SDUMP	h
SCERECRD=1	RECORD	i

After the requested action has been taken, processing continues at step 3.

Extended Description	Module	Label

a IEAVTSL2 does nothing when the IGNORE action is specified. Processing continues at step 3.

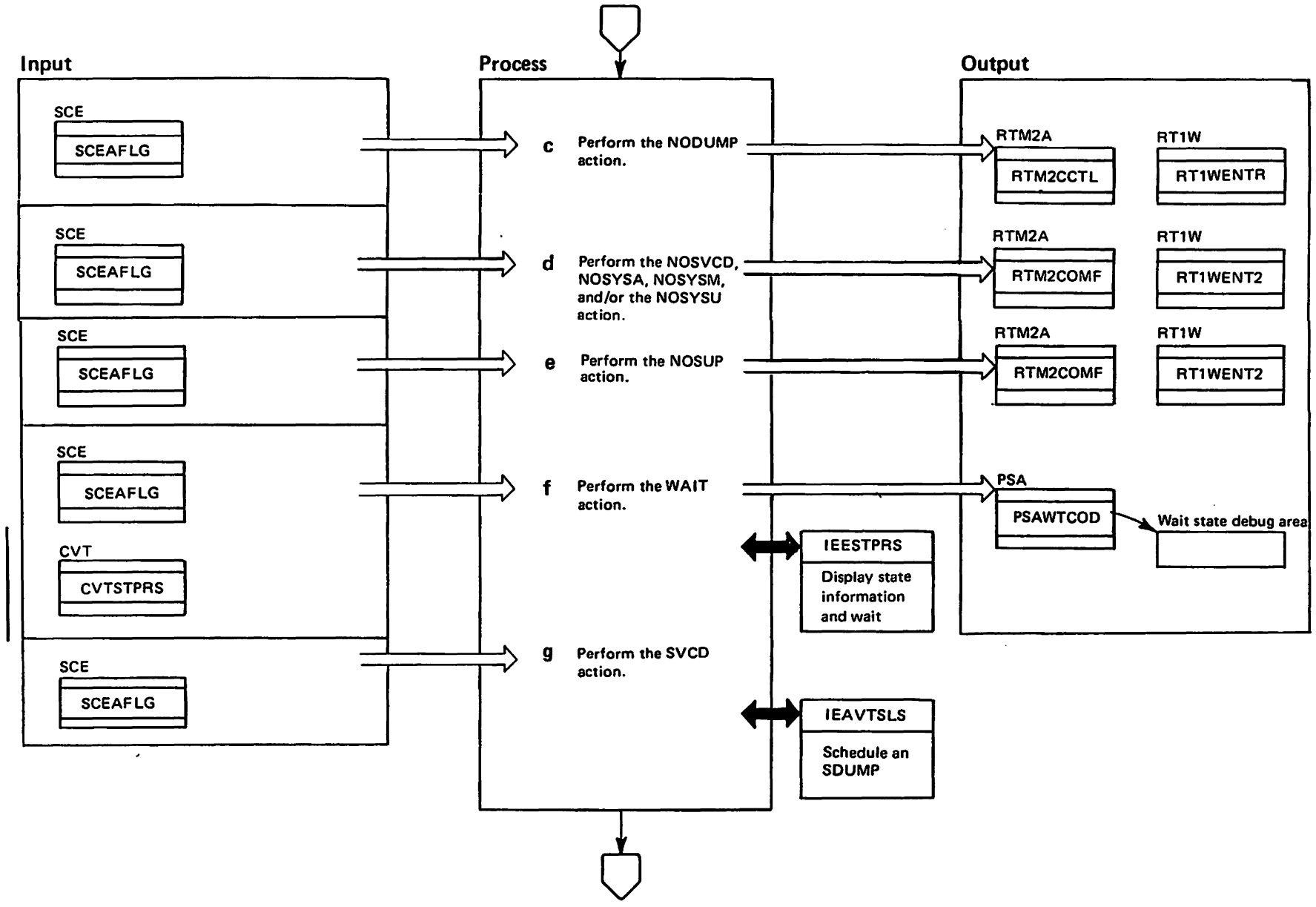
b The EOL subroutine requests that a GTF SLIP trace record be built by:

- Initializing a parameter list for GTF.
- Issuing the appropriate form of the HOOK macro, which gives GTF control to build the trace record. The EOL subroutine refers to the SCVATRD field to determine which type of record to request (SLIP standard, SLIP standard plus user-defined data, or SLIP user-defined data).

If IEAVTRT2 is SLIP's caller, the EOL subroutine issues a TYPE=P HOOK macro, which results in a monitor call (MC) instruction being executed. For all other callers, a TYPE=BPN2 HOOK macro is used, which results in a branch entry to GTF.

Processing continues at step 3.

IEAVTSL2 – SLIP Trap Matching Routine – Action Keyword Processing (Part 3 of 6)



Extended Description	Module	Label

c The EOL subroutine takes the following action, depending on SLIP's caller:

Caller	*EOL action*
IEAVTRTS	Sets the RT1NODMP bit to 1 to suppress dumps.
IEAVTRT2	Sets the RT2NODMP bit to 1 to suppress dumps.
IEAVTRTM	Nothing; there are no dumps to suppress.
IEAVTPER	Not applicable; the NODUMP option cannot be coded on PER traps.

Processing continues at step 3.

d The EOL subroutine takes the following action, depending on SLIP's caller:

Caller	*EOL action*
IEAVTRTS	Sets the appropriate bit(s) to 1 to suppress dumps (RT1WNOSV, RT1WNOSA, RT1WNOSM, RT1WNOSU).
IEAVTRT2	Sets the appropriate bit(s) to 1 to suppress dumps (RTM2NOSV, RTM2NOSA, RTM2NOSM, RTM2NOSU).
IEAVTRTM	Nothing; there are no dumps to suppress.
IEAVTPER	Not applicable; the NOSVCD, NOSYSA, NOSYSM, and NOSYSU options cannot be coded on PER traps.

Processing continues at step 3

e The EOL subroutine takes the following action, depending on SLIP's caller:

Caller	*EOL action*
IEAVTRTS	Sets the RT1WNOSP bit to 1 to indicate to dumping services that dump requests should not be suppressed.
IEAVTRT2	Sets the RTM2NOSP bit to 1 to indicate to dumping services that dump requests should not be suppressed.
IEAVTRTM	Nothing; there are no dumps.
IEAVTPER	Not applicable; the NOSUP option cannot be coded on PER traps.

Processing continues at step 3.

Extended Description	Module	Label

f Processing a wait option (SCEWAIT=1) requires that the processor be running disabled. When SLIP's caller is IEAVTRT2 (the only enabled caller), the EOL subroutine disables the current processor by using a STNSM instruction. This code saves the system mask from the current PSW and replaces it with a mask disabled for external and I/O interruptions. The EOL subroutine uses a compare and swap (CS) instruction to obtain the restart resource. If the attempt is unsuccessful, EOL takes no wait action and processing continues at step 3.

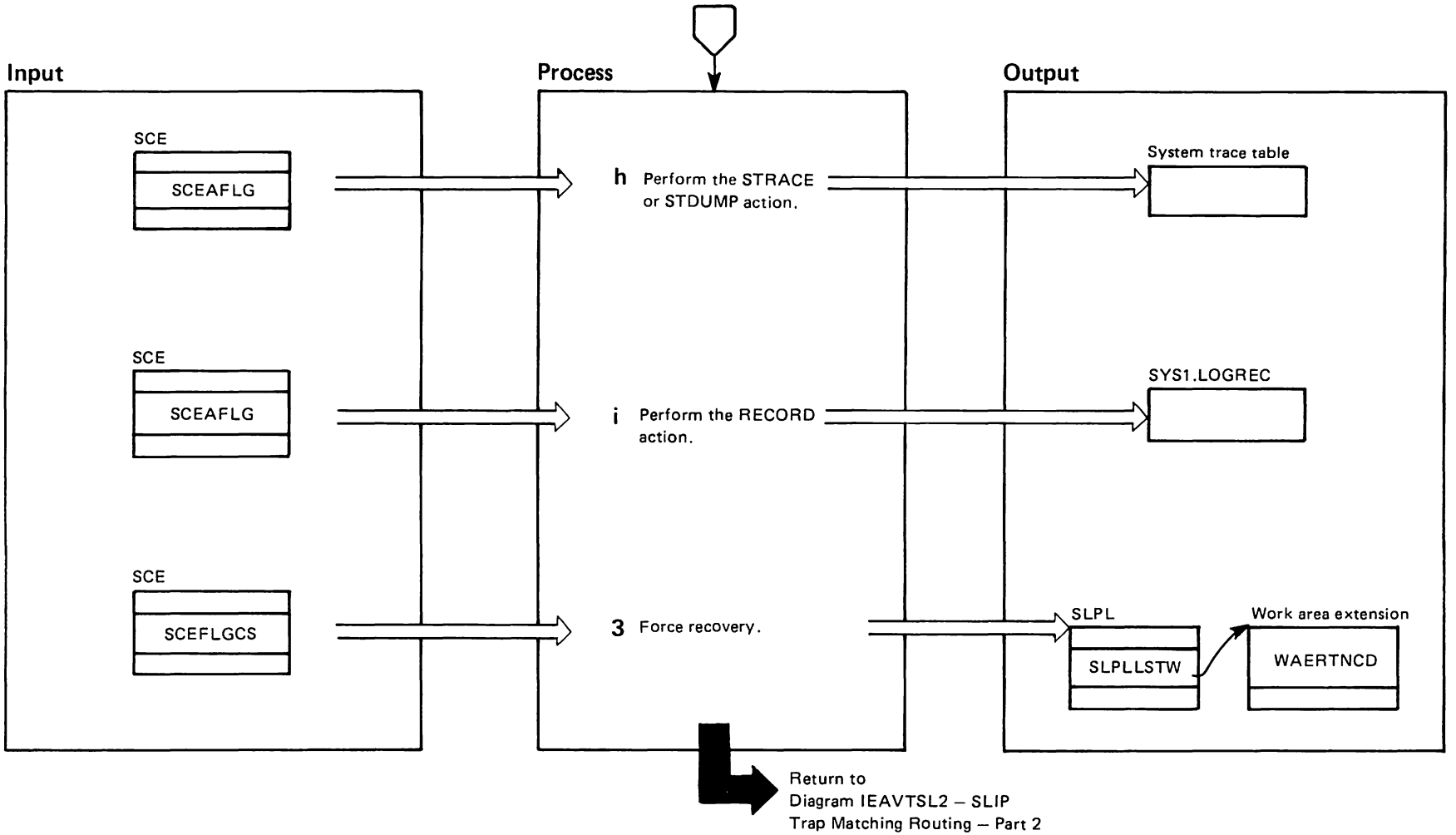
If the restart resource is obtained, the EOL subroutine:

- Builds a debugging area pointed to by the PSAWTCOD field.
- Sets up the stop/restart parameters.
- Calls the stop/restart routine (IEESTPRS) to display (using the disabled console communications facility) a message showing the registers, PSW, and cross memory information. The system waits for a response before continuing. If the system is restarted, EOL frees the restart resource and, if IEAVTRT2 is the IEAVTSLP caller, restores the system mask. — IEESTPRS

Processing continues at step 3.

g The EOL subroutine calls IEAVTSLS to request an SDUMP. — IEAVTSL2 IEAVTSLS — EOL

IEAVTSL2 – SLIP Trap Matching Routine – Action Keyword Processing (Part 5 of 6)



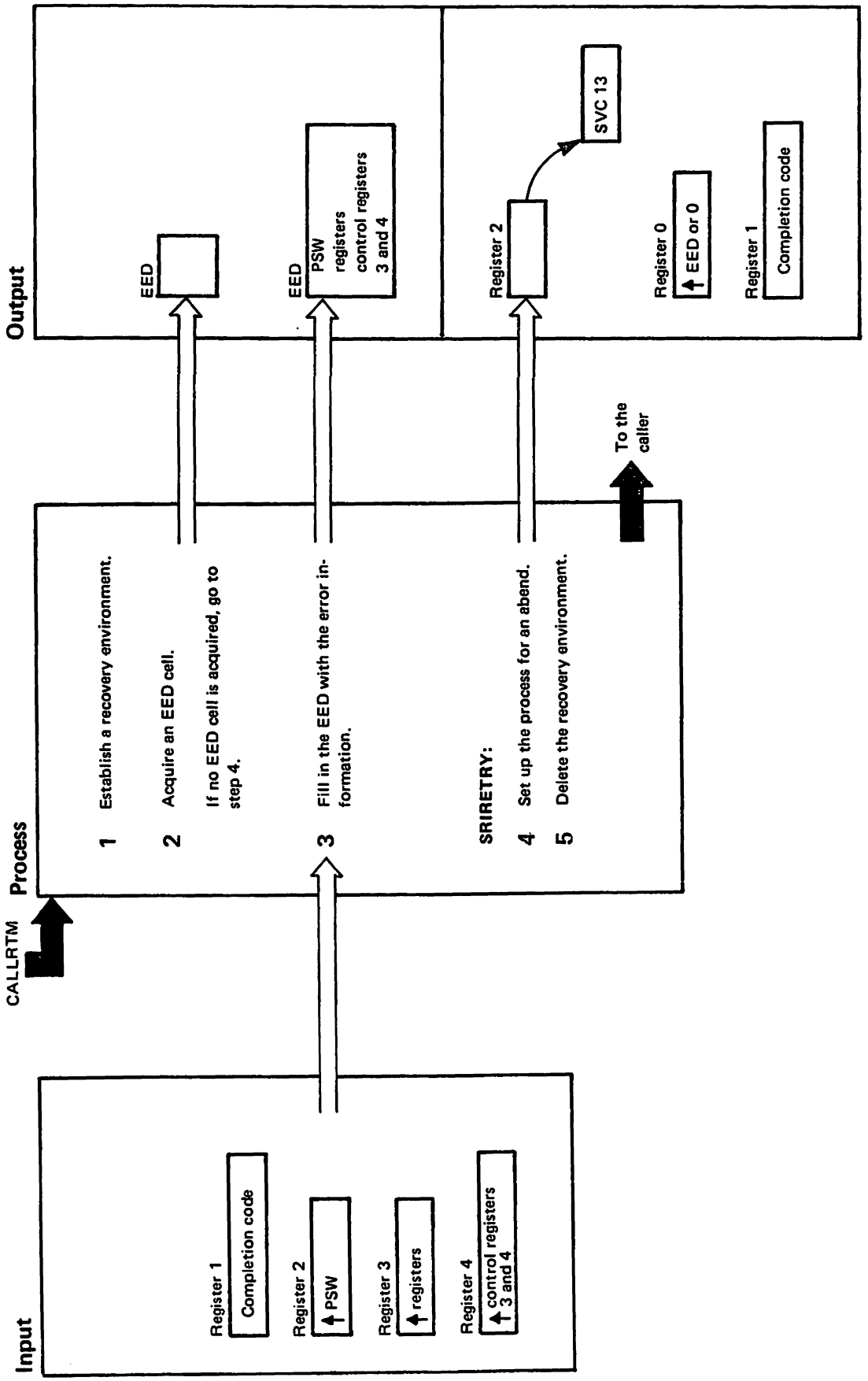
IEAVTSL2 – SLIP Trap Matching Routine – Action Keyword Processing (Part 6 of 6)

Extended Description	Module	Label
h The EOL subroutine builds the parameters required by the PTRACE macro for TYPE=SPER events. It then issues the PTRACE macro which gives control to the System Trace Formatter Routine (ITRF0009) to build the SPER entry in the Trace Table.		
i The EOL subroutine takes the following action, depending on SLIP's caller:		

Caller	*EOL action*
IEAVTRTS	Sets the RT1WRCRD bit to 1 to indicate to RTM1 that recording should be done for all FRRs and ESTAEs for this error.
IEAVTRT2	Sets the RTM2RCRD bit to 1 to indicate to RTM2 that recording should be done for all ESTAEs for this error.
IEAVTRTM	Nothing; there are no ESTAEs or FRRs entered in this path.
IEAVTPER	Not applicable; the RECORD option cannot be coded on PER traps.

3 If a PER interruption is being processed and the trap specified the RECOVERY action parameter, the EOL subroutine adjusts the return code to indicate that recovery processing for the interrupted program is required. IEAVTSLE passes the return code to IEAVTPER, which passes it to IEAVEPC. IEAVEPC forces a recovery of the interrupted program. (See the diagram and extended description of IEAVEPC.)	IEAVTSL2	EOL

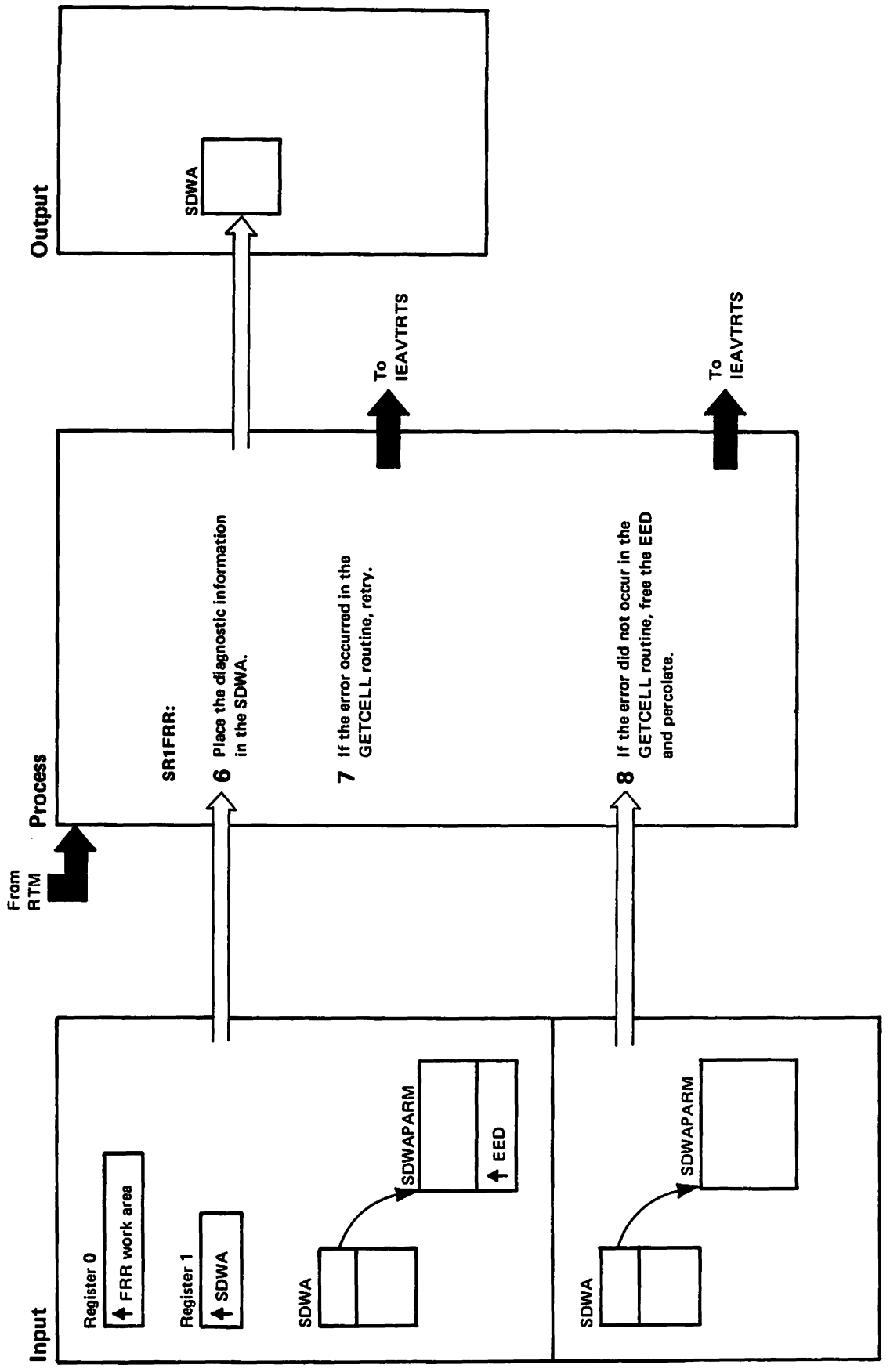
IEAVTSR1 - ITERM PROCESSOR



IEAVTSR1 – ITERM Processor (Part 1 of 4)

IEAVTSR1 – ITERM Processor (Part 2 of 4)

Extended Description	Module	Label
ITERM is used to terminate a process (an SRB routine or a task) that has been interrupted. IEAVTSR1 processes a service mode call to RTM by setting up the interrupted process for an abend.	IEAVTSR1	
1 IEAVTSR1 establishes its own FRR (SR1FRR) for recovery.		
2 IEAVTSR1 attempts to obtain an EED (extended error descriptor) cell for a savearea. If no EED cell is acquired, ITERM continues processing and keeps no record of error information.		
3 IEAVTSR1 saves registers 0-15, the PSW and control registers 3 and 4 in the EED. IEAVTSR1 eventually presents this error information to the recovery routines when the process (an SRB routine or a task) is redispatched.		
4 IEAVTSR1 places the address of an SVC 13 in the old PSW. When the process is redispatched, an ABEND is issued. Register 0 either points to the EED containing the saved error information or a zero if no EED was obtained.		SR1RETRY
5 IEAVTSR1 deletes the FRR and returns control to the caller.		

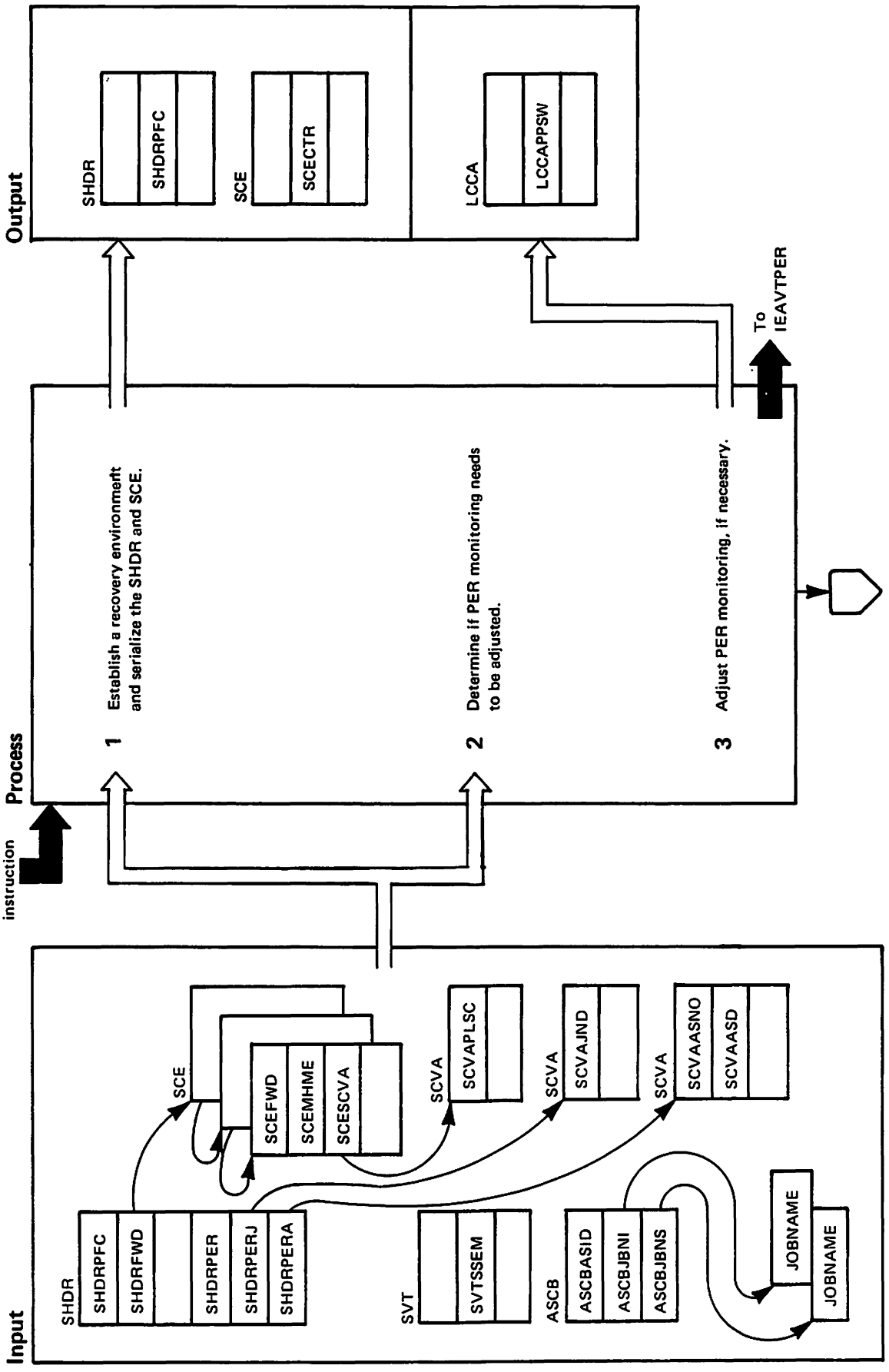


IEAVTSR1 – ITERM Processor (Part 3 of 4)

IEAVTSR1 – ITERM Processor (Part 4 of 4)

Extended Description	Module	Label
6 ITERM's FRR places the following diagnostic information in the SDWA: component ID, component description, date of the assembly, version of the module, the FRR label, the module name, and the CSECT name.		SR1FRR
7 If the error occurred while ITERM was acquiring an EED cell, the FRR requests a retry at entry point SR1RETRY and has RTM record the error.		
8 If the error did not occur while ITERM was acquiring an EED cell, no retry is attempted. The FRR frees the EED cell, requests error recording, and percolates.		

IEAVTSSH - SLIP SPACE SWITCH HANDLER

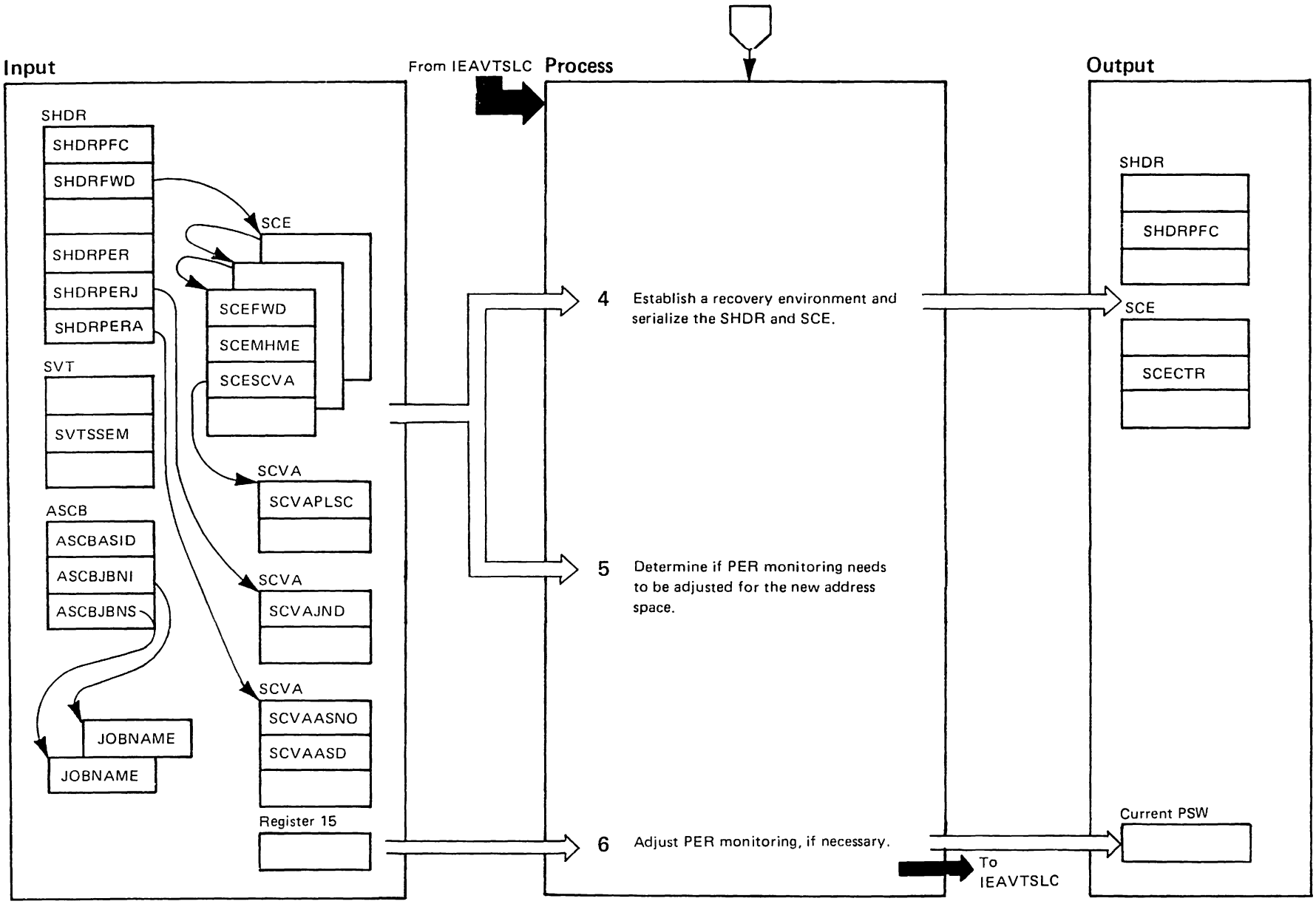


IEAVTSSH – SLIP Space Switch Handler (Part 1 of 4)

IEAVTSSH – SLIP Space Switch Handler (Part 2 of 4)

Extended Description	Module	Label
IEAVTSSH intercepts calls to change a unit of work's addressing environment and adjusts the PER bit in the PSW according to the parameters coded on the SLIP command. IEAVTSSH has separate entry points for the instructions/macros it intercepts. IEAVTSSH – intercepts PT and PC instructions. (Described in steps 1-3.) IEAVTSS1 – intercepts CMSET SET macros, CMSET RESET macros when the parameter CHKAUTH=YES is coded, CMSET RESET macros when the parameter CHKAUTH=NO is coded. (Described in steps 4-6.)		
During SLIP initialization, the addresses of the CMSET routines were saved in the SLIP header (SHDR) and replaced by the corresponding entry point addresses of IEAVTSLC. (IEAVTSLC executes the requested CMSET before calling IEAVTSSH.) After a CMSET SET or CMSET RESET macro changes the addressing environment for a unit of work or before a PC or PT instruction completes a space switch operation, IEAVTSSH checks and adjusts PER monitoring before returning control to IEAVTSLC or to the issuer of the PC, or PT instruction. IEAVTSSH assures the integrity of the SLIP trap in a cross memory environment no matter where a unit of work is operating.		
1 IEAVTSSH establishes an FRR (See Recovery Processing at the end of this diagram) and serializes the SLIP header (SHDR) and SCE chain. Serialization is done so that this SLIP trap cannot be deleted by another routine until IEAVTSSH finishes checking and adjusting PER monitoring.		COMMON
2 COMMON uses the parameters from the SLIP command to determine if the PER bit is to be left in its current state, turned on, or turned off. COMMON places a return code in register 15 stating which is to be done.		
3 IEAVTSSH uses the return code from COMMON to set the PER bit in the PSW of the interrupted unit of work (LCCAPPSW). IEAVTSSH either changes PER monitoring to on or off, or leaves PER monitoring in its present state, depending on the return code. IEAVTSSH returns control to IEAVTPER to finish processing the space switch interrupt.		

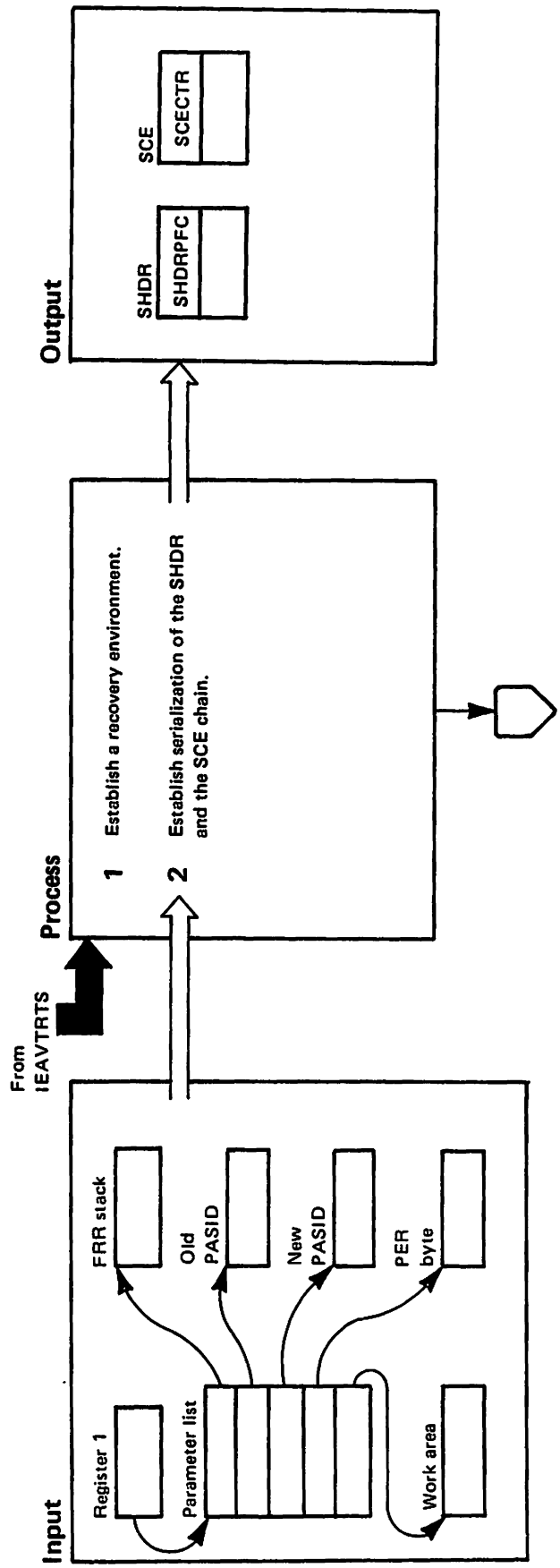
IEAVTSSH – SLIP Space Switch Handler (Part 3 of 4)



IEAVTSSH – SLIP Space Switch Handler (Part 4 of 4)

Extended Description	Module	Label
4 IEAVTSSH uses the COMMON subroutine to establish an FRR (See Recovery Processing at the end of this diagram) and to serialize the SLIP header and SCE chain.		COMMON
5 COMMON uses the parameters from the SLIP command to determine if the PER bit is to be left in its current state, turned on, or turned off. COMMON places a return code in register 15 stating which action is to be taken.		
6 IEAVTSSH uses the return code from COMMON to set the PER bit in the current PSW. Control returns to IEAVTSLC.		
Recovery Processing		
If an error occurs while IEAVTSSH is executing, RECOVERY receives control. The FRR releases the serialization of the SLIP header and SCE chain if serialization was completed. If a retry is allowed, the FRR requests a retry from RTM to the caller of IEAVTSSH. If a retry is not allowed, the FRR requests that RTM continue with termination.		RECOVERY

IEAVTSSX - SPACE SWITCH EXTENSION



IEAVTSSX – Space Switch Extension (Part 1 of 4)

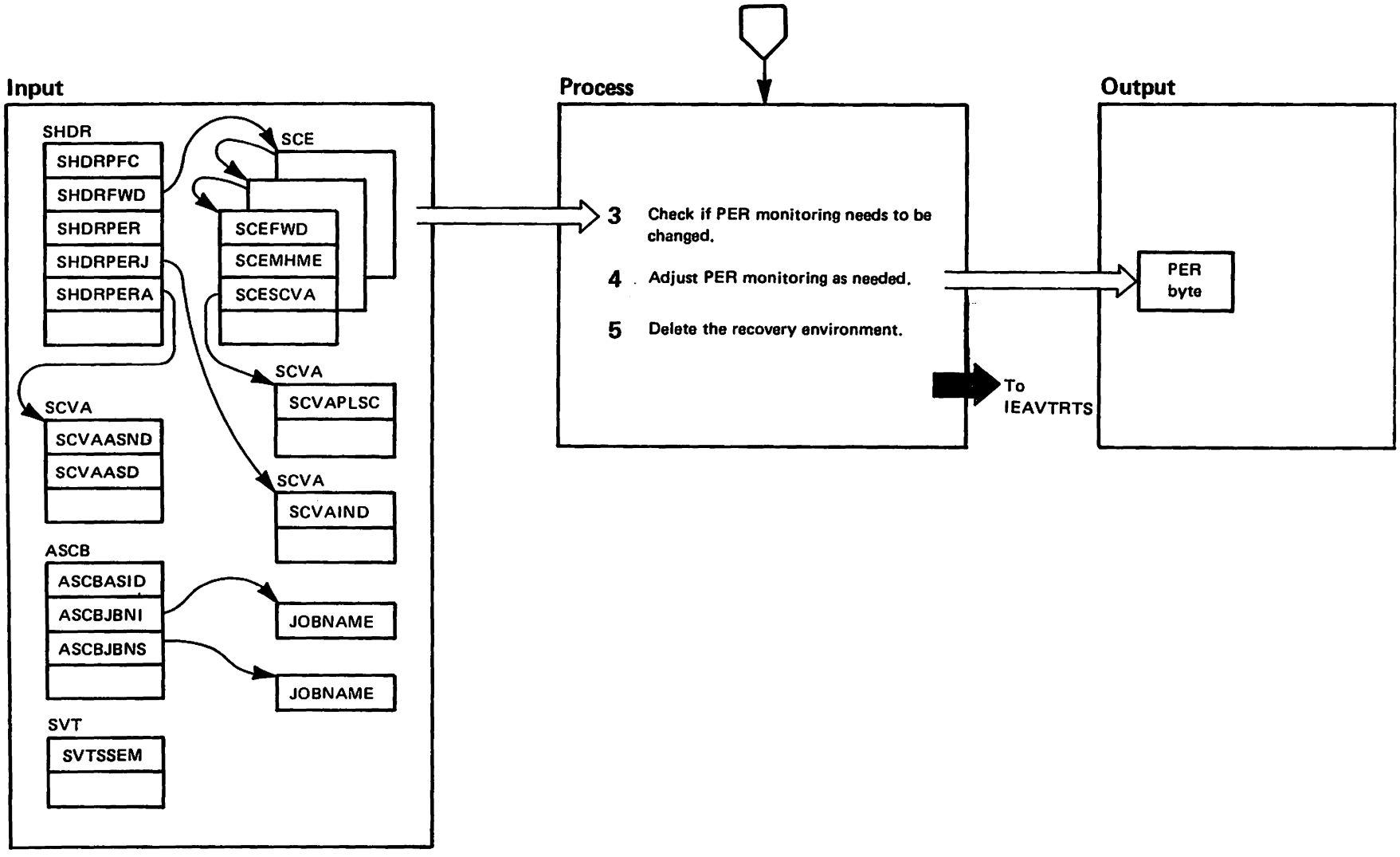
IEAVTSSX – Space Switch Extension (Part 2 of 4)

Extended Description | **Module** | **Label**

Before invoking an FRR, RTM1 (IEAVTRTS) calls IEAVTSSX to set the PER bit in the PSW that is used to give the FRR control. IEAVTSSX performs the same function as IEAVTSSH (turn the PER bit on or off as requested), but is tailored to meet the needs of RTM1.

1 IEAVTSSX issues a SETFRR to establish its own FRR. (See Recovery Processing at the end of this diagram.)

2 IEAVTSSX serializes the SLIP header (SHDR) and SCE chain to prevent alteration of the SLIP environment by another caller.



Extended Description **Module** **Label**

3 IEAVTSSX checks the old and new PASIDs and the parameters on the SLIP command to determine if the PER bit is to be turned on or off.

4 IEAVTSSX sets the PER bit in the input mask of the PER byte.

5 IEAVTSSX releases the serialization of the SCE chain and the SHDR, and deletes the FRR.

Recovery Processing

If an error occurs while IEAVTSSX is executing, RECOVERY receives control. In case of an error, the FRR releases the serialization of the SCE chain and the SHDR if serialization had been completed.

If a retry is allowed, the FRR is deleted and control is given to IEAVTRTS for eventual return to IEAVTSSX.

If no retry is allowed, the FRR gives control to IEAVTRTM to continue with termination.

INDEX

J

K

L

M

N

P

R

S

T

W

X

MVS/Extended Architecture
System Logic Library:
Recovery Termination
Management

READER'S
COMMENT
FORM

LY28-1735-0

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Note: ***Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.***

Possible topics for comment are:

Clarity Accuracy Completeness Organization Coding Retrieval Legibility

If you wish a reply, give your name, company, mailing address, and date:

_____

_____

_____

_____

What is your occupation? _____

How do you use this publication? _____

Number of latest Newsletter associated with this publication: _____

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.)

LY28-1735-0

S370-36

Reader's Comment Form

Cut or Fold Along Line

Fold and tape | Please Do Not Staple | Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Department D58, Building 921-2
PO Box 390
Poughkeepsie, New York 12602

Fold and tape | Please Do Not Staple | Fold and tape

Printed in U.S.A.

IBM®

MVS/Extended Architecture System Logic Library: Recovery Termination Management

LY28-1735-0 S370-36

Printed in U.S.A.