LY28-1765-0
File No. S370-36

MVS/Extended Architecture System Logic Library: Supervisor Control

Program Product

MVS/System Product:

JES3 Version 2	5665-291
JES2 Version 2	5740-XC6

IBM

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 13, LY28-1254-2, 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 or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product in this publication is not intended to state or imply that only IBM's 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 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. Refer to MVS/XA System Initialization Logic for detailed information about the MVS control program prior to this point; refer to the MVS/XA Overview for general information about the MVS control program and the relationships among the components; and refer to the list of Corequisite Reading and Related Publications in the Master Preface to obtain the names of publications that describe some of the components not in the System Logic Library.

HOW THE LIBRARY IS ORGANIZED

MULTIPLE VOLUMES

The System Logic Library consists of multiple volumes. Volume 1 contains the master preface, the master table of contents, the master figure list, and the master index for the remaining volumes in the library. The last volume, or the module description volume, contains module descriptions for all of the modules in the components documented in the System Logic Library and an index. Each of the other volumes, or the component volumes, contains its own table of contents and index of the information in that particular volume. The component volumes describe the logic of the components in the MVS control program.

ORGANIZATION OF THE COMPONENTS

The component volumes are organized alphabetically by section name. Each section contains information about one or more of the components in the MVS control program. A section contains more than one component when the components are closely related, frequently referenced at the same time, and not so large that they require a volume of their own.

A three or four character mnemonic is associated with each section and is used in all figure, diagram, and page numbers in that section. For example, the mnemonic ASM is associated with the section "Auxiliary Storage Management."
All figures in this section are identified as Figure ASM-n, all diagrams as Diagram ASM-n, and all pages as ASM-n, where n represents the specific figure, diagram, or page number. Whenever possible, existing component acronyms are used as the mnemonic for a section. The mnemonics are in alphabetic order. The Table of Section Names in the Master Preface lists the section names, the components included in each section (if a section contains more than one component), the mnemonics for the sections, and the volume and order number for each volume.

HOW TO USE THE LIBRARY

To use this library efficiently, readers must be able to find the information that they need quickly; they must be aware of the types of information provided for each component; and they must know how to obtain additional information before referencing the System Logic Library. The following topics cover these points.

FINDING INFORMATION USING THE VOLUME TITLES

As readers become familiar with the section names, their mnemonics, and contents, they will be able to use the System Logic Library as they would an encyclopedia and go directly to the volume that they need. To help readers locate the correct volume, section mnemonics are included in the titles of the component volumes. If a volume contains one section, the mnemonic for that section is specified; if a volume contains more than one section, the mnemonics for the first and last section in the volume are specified.

The Table of Section Names in the Master Preface contains a list of section names and mnemonics. It provides a quick reference to the mnemonics and the components included in each section.

FINDING INFORMATION USING THE MASTER INDEX

Readers who are not sure which section contains the information they are looking for can locate information by using the master index in Volume 1. For the component volumes, the page number in an index entry consists of the mnemonic for the section, the volume number as a superscript on the mnemonic, and the page number; for the last volume (which contains the module descriptions), the page number consists of the volume number instead of a mnemonic and the page number. For example:

ASM-12   refers to the "Auxiliary Storage Management" section in volume 3, page 12.

The volume number and section mnemonic are not repeated for successive references to the same section in a single entry in the master index; for example, ASM-12, 17 refers to both pages ASM-12 and ASM-17. Familiarity with the library will aid in locating the exact volume in which a component is documented.

INFORMATION PROVIDED FOR EACH COMPONENT

The following information is provided for each 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 and logic diagrams. The automated module information is generated from the module prologue, block and line comments within the code, and the code itself. It consists of four parts: module description, module operation summary, diagnostic aids, and a logic diagram.
5. Module descriptions that describe the operation of the modules

Some components also include diagnostic techniques information, used for debugging the component, following the Introduction.

Items 1 through 4 are located in the component volumes; item 5 is located in the last volume.

FURTHER INFORMATION

For more information about the System Logic Library, including the order numbers of the library's publications, see the Master Preface.

CONTENTS

FIGURES

Summary of Amendments
for LY28-1765-0
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 MVS/XA System Logic Library Volume 13 , LY28-1254-2, 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:

- New module

  IEAVESLX

- Changed modules

  IEAVELK
  IEAVELCR
  IEAVEVRR
  IEAVESTU

- Minor technical and editorial changes throughout the publication.

SUPERVISOR CONTROL

INTRODUCTION

Supervisor control includes the following services:

- The service manager, which schedules requests.
- The dispatcher, which dispatches work. The dispatcher and memory switch are described in the DISP section of the System Logic Library.
- The various interruption handlers, which route control to appropriate routines for given interruptions.
- Interprocessor communications (IPC), which senses or changes the hardware status of another processor.
- The exit effectors, which provide a mechanism for scheduling asynchronous exits.
- The lock managers, which permit serialization of system resources.
- Spin loop timeout, which prevents inter-processor deadlock situations from occurring.
- Intersect serialization function, which serializes the dispatching queues.
- Validity checking routines, which validate queues, control blocks and addresses.
- Supervisor control recovery routines, which provide functional recovery for supervisor control.

SUPERVISOR CONTROL SERVICES

The following topics give a brief overview of the supervisor control services.

SERVICE MANAGEMENT

In order to facilitate multiprocessing, MVS uses a category of facilities, called service management, to schedule system services. Service management consists of:

- The SCHEDULE macro instruction, which allows new service requests to be entered into the queue of dispatchable work with a minimal amount of overhead.
- A control block, supplied to SCHEDULE as input and called a service request block (SRB), which represents a service request. The SRB contains information needed to dispatch the routine.
- The PURGEDQ macro instruction, which allows service requests to be terminated.

Figure 1 on page SUP-4 shows the basic pointer structure utilized by the service management facilities. This structure incorporates two levels of system priority, global and local. SRBs queued off the SVT are global SRBs; SRBs queued off the ASCB are local SRBs.

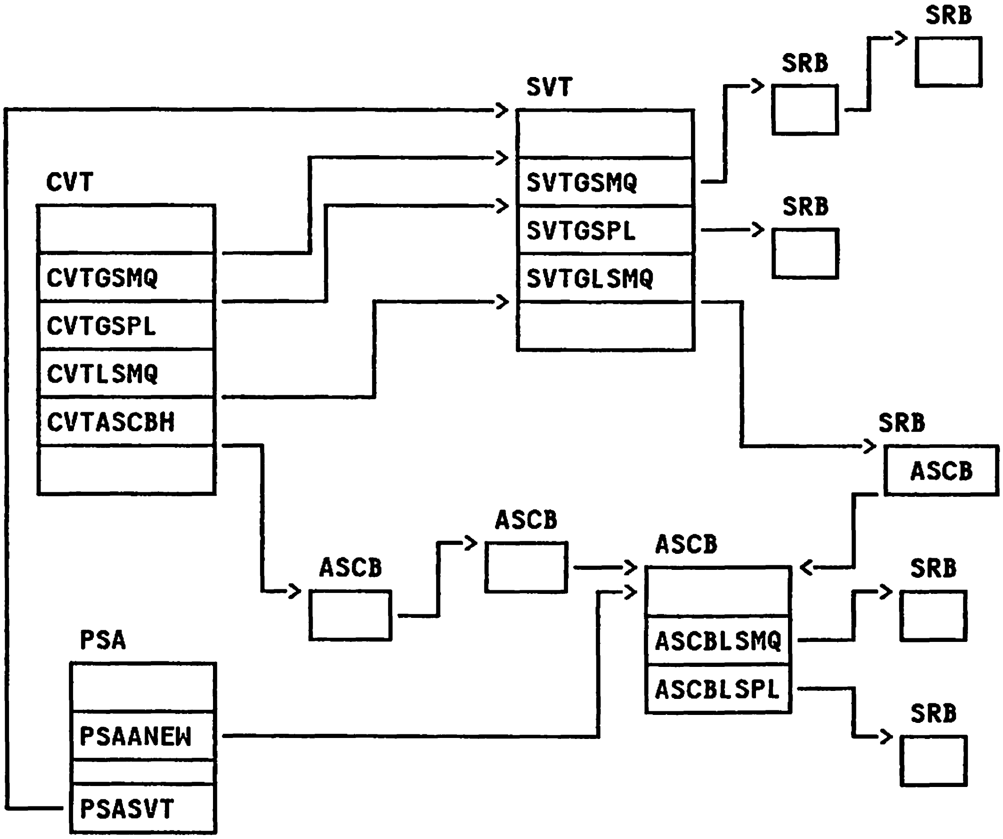


Figure 1. SRB Scheduling Pointer Structure

At the global level, there are two queues, the global service management queue (GSMQ) and the global service priority list (GSPL). Likewise, at the local level there are two queues, the LSMQ and the LSPL. In addition, there is a single SMQ at the local level (SVTLSMQ) that is maintained for compatibility reasons. The SMQs are used as staging queues, and the SPLs are used as dispatching queues.

Service requests scheduled at the global level are placed on the appropriate global queue and have a priority higher than any address space, regardless of the actual address space in which they will be dispatched. Service requests scheduled at the local level will be placed on the single local SMQ (SVTLSMQ) or the appropriate local queue in the address space in which the SRB will be dispatched. Any SRBs scheduled to the single local SMQ will eventually be moved to the appropriate local queue in the address space in which it will be dispatched.

Service requests scheduled at a local level have a priority equal to the address space in which they will be dispatched, but higher than any task within that address space.

These scheduled routines have the following characteristics:

- They receive control in supervisor state.
- They may execute enabled for interruptions, but will not lose control to higher priority work unless they are suspended for a page fault, a lock, or a page fix.
- They may free the SRB control block once they get control.
- They may not issue SVCs.

- They may execute in any designated address space.

To use the service management facility, the user must:

1. Construct the SRB.

2. Schedule it, using the SCHEDULE macro, to the appropriate global or local queue.

The user will then continue to execute until he is interrupted, causing an entry into the dispatcher.

INTERRUPTION HANDLERS

The interruption handlers route control to the appropriate routines after machine interruptions occur. Any interruption causes processor control to be taken from the executing program and given to an interruption handling routine.

Any interruption causes the current PSW to be saved as the old PSW, and the new PSW to be loaded. This new PSW passes control to the appropriate interruption-handling routine.

The interruption handlers process:

- SVC interruptions, which occur when an SVC instruction is executed. The SVC FLIH (first level interruption handler) determines which SVC routine the requester wants and passes control to it.

- I/O interruptions, which occur when a channel or device signals a change of status. For example, an I/O operation terminates, an error occurs, or a device becomes ready. The I/O FLIH branches to the I/O supervisor, which performs the I/O services and handles I/O errors.

- External interruptions, which occur for:

  - Timer interruptions (for processor timer expiration, clock comparator interruption, or clock synchronization failure; see the TIME section of the System Logic Library).

  - Hitting interrupt key (when the operator presses interrupt key on the console).

  - External calls (when remote pendable signal routine signals another processor).

  - Emergency signals (when machine check handler or remote immediate signal routine signals another processor).

  - Service signal interruptions (resulting from the Service Processor Call SVC 122).

  - Malfunction alerts (caused by machine failure of another processor).

  The external FLIH determines the cause of the interruption and branches to the external service routine.

- Restart interruptions, which occur when the operator initiates restart on the system operator's console, or when a system program issues a SIGP (signal processor) instruction with restart order code. The restart FLIH routes control to RTM (recovery termination management).

- Program interruptions, which may be caused by program errors (invalid operation, protection exception); page faults or segment faults (caused by referencing a page not in main storage); event monitoring (caused by a monitor call instruction (MC) or a program event recording (PER) interruption). The program FLIH determines the cause of the interruption, and does one or more of the following:

- Calls real storage management on paging exceptions to determine if this is a valid page fault, and if so, to initiate processing to bring the page into real storage.
- Calls the generalized trace facility (GTF) for tracking.
- Calls RTM if the program exception appears to be a program error.
- Calls the vector SLIH to build the environment required for using the vector feature.

INTERPROCESSOR COMMUNICATIONS (IPC)

Interprocessor communications (IPC) include the signal service routines, plus the external call and emergency signal SLIHs (second level interruption handlers). The main purpose of IPC consists of sensing or changing the hardware status of another processor or causing special routines to be invoked on another processor.

The signal service routines perform two different types of signal services — direct and remote. The direct signal service (IEAVEDR), invoked via the DSGNL macro, uses the signal routine, IEAVESGP, to issue the signal to modify, sense or alter the physical state of a specific processor. The remote signal services, (IEAVERI or IEAVERP), invoked via the RISGNL or RPSGNL macros, use the signal routine, IEAVESGP, to issue the emergency signal or the external call signal to route control to a routine on a specific processor.

Direct Signal Services

The direct signal function is invoked via the DSGNL macro. The direct service is defined for those control program functions that require the modification or sensing of the physical state of one of the configured processors.

The direct signal function consists of the following:

- IEAVEDR executes on the sending processor to validate the DSGNL request and to set up the interface to IEAVESGP so that the SIGP instruction can be issued.
- IEAVESGP (signal routine) executing on the sending processor issues the SIGP for one of the following order codes:
  - Sense
  - Start
  - Stop
  - Restart
  - Stop and Store Status
  - Initial CPU Reset
  - CPU Reset
  - Set Prefix Register
  - Store Status at Address
- The specified receiving processor's physical state is sensed or altered.

Remote Signal Services

There are two types of remote signal services:

- Remote immediate signal — invoked via the RISGNL macro.
- Remote pendable signal — invoked via the RPSGNL macro.

The remote immediate signal function consists of the following:

- IEAVERI, executing on the sending processor, sets up the interface to the receiving routine.
- IEAVESGP (signal routine), executing on the sending processor, issues the emergency signal SIGP instruction.
- IEAVEES (the emergency signal SLIH) receives control from the external FLIH on the receiving processor and routes control to the receiving routine.

The remote pendable signal function consists of the following:

- IEAVERP (executing on the sending processor) indicates to the receiving processor what functions to perform.
- IEAVESGP (signal routine) also executing on the sending processor, issues the external call SIGP.
- IEAVEXS (the external call SLIH) receives control from the external FLIH on the receiving processor and routes control to the receiving routine.

EXIT EFFECTORS (SCHEDULING EXIT ROUTINES)

A service exists whereby a program may request that a user-defined exit routine execute asynchronously. System routines use this service to handle asynchronous events such as end-of-task condition, expiration of a timer interval, or special I/O handling (for example, tape label checking or I/O error checking).

The scheduling of user exit routines, called asynchronous exit routines, is handled by three supervisor routines: the stage 1 exit effector, the stage 2 exit effector, and the stage 3 exit effector.

In order to schedule a routine to execute asynchronously under a specific task, an interrupt request block, IRB, must be placed on that task's RB chain. The following describes the control flow for that mechanism.

1. The user must first create and format the IRB via the CIRB macro instruction. CIRB invokes the stage 1 exit effector, which obtains storage from LSQA and formats the IRB.
2. The user must set up the interface (to the stage 2 exit effector), which is in one of the following forms:

   a. Interrupt queue element (IQE). This contains the TCB and IRB addresses.

   b. Request queue element (RQE). This is exclusively a data management interface, allowing asynchronous exits to be scheduled from I/O appendages. The RQE contains the TCB and IRB addresses.

   c. Service request block (SRB). This is used by only IOS when scheduling a non-resident error recovery procedure. In each address space there is a predetermined task designated as the error task. (Its address is contained in ASXBETSK). Each address space also has a pre-formatted system IRB (SIRB). An SRB passed to stage 2 exit effector represents a request to schedule the

SIRB to the error task. The SIRB always gives control to the IOS error recovery procedure loader.

The user branch-enters the stage 2 exit effector with either the address of an IQE, RQE, or SRB. Stage 2 queues the request off of the ASXB for the current address space and returns to the caller.

3. When the dispatcher checks an address space for available work, it determines if there are queued requests. If so, it invokes the stage 3 exit effector.

4. Stage 3 processes the queued requests. Stage 3 dequeues the requests (IQE, RQE, or SRB) from the asynchronous exit queue and places the associated IRB on the indicated task's RB chain.

   When the dispatcher dispatches that task, with the IRB highest on the RB chain, the asynchronous exit will get control.

See Figure 2 on page SUP-9 for an illustration of exit effector data structure.

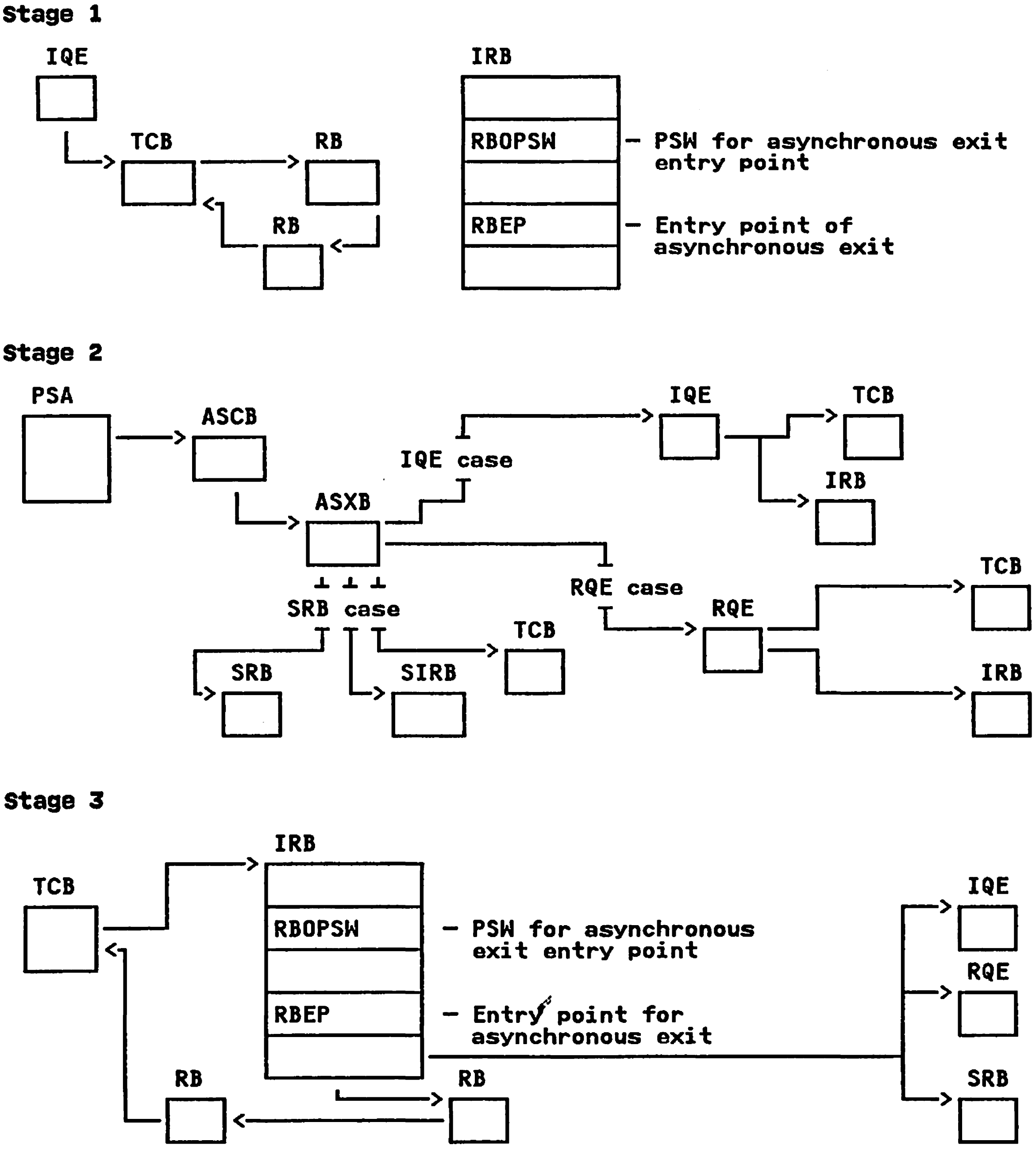

Stage 1
IQE
TCB
RB
RB
IRB
RBOPSW
RBEP
- PSW for asynchronous exit entry point
- Entry point of asynchronous exit
Stage 2
PSA
ASCB
ASXB
IQE case
IQE
TCB
IRB
RQE case
RQE
TCB
IRB
SRB case
SRB
SIRB
TCB
Stage 3
TCB
IRB
RBOPSW
RBEP
- PSW for asynchronous exit entry point
- Entry point for asynchronous exit
RB
RB
IQE
RQE
SRB

Figure 2. Asynchronous Exit Effector Data Structure

LOCK MANAGEMENT

In a multiprocessor system, some method of serialization must be used to prevent interference between processors competing for a resource. MVS/XA uses locking to serialize resources.

There are two types of locks; spin and suspend.

Spin Locks

An unconditional request for a spin lock causes a disabled loop on the processor until the lock becomes available if it cannot be immediately obtained. The owner of a spin lock must run disabled for I/O and external interrupts and cannot take a page fault.

A list of global locks in hierarchical order, highest first, are listed below.

Lock Name	Description
RSMGL	Real storage management global lock - serializes RSM global resources.
VSMFIX	Virtual storage management fixed subpools lock - serializes VSM global queues.
ASM	Auxiliary storage management lock - serializes ASM resources on an address space level.
ASMGL	Auxiliary storage management global lock - serializes ASM resources on a global level.
RSMST	Real storage management steal lock - serializes RSM control blocks on an address space level when it is not known which address space locks are currently held.
RSMCM	Real storage management common lock - serializes RSM common area resources (such as page table entries).
RSMXM	Real storage management cross memory lock - serializes RSM control blocks on an address space level when serialization is needed to a second address space.
RSMAD	Real storage management address space lock - serializes RSM control blocks on an address space level.
RSM	Real storage management lock (shared/exclusive) - serializes RSM functions and resources on a global level.
VSMPAG	Virtual storage management pageable subpools lock - serializes the VSM work area for VSM pageable subpools.
DISP	Global dispatcher lock - serializes the ASVT and the ASCB dispatching queue.
SALLOC	Space allocation lock - serializes receiving routines that enable a processor for an emergency signal or malfunction alert.
IOSYNCH	I/O supervisor synchronization lock - serializes, using a table of lockwords, IOS resources.

Lock Name	Description
IOSUCB	I/O supervisor unit control block lock - serializes access and updates to the UCBs. There is one IOSUCB lock per UCB.
SRM	System resources management lock - serializes SRM control blocks and associated data.
TRACE	Trace lock (shared/exclusive) - serializes the reading (shared) and writing (exclusive) of the system trace buffer.
CPU	Processor lock - provides system-recognized (legal) disablement. Note that the CPU lock has no hierarchy in respect to the other spin type locks. However, once obtained, no suspend locks can be obtained.

For a more detailed explanation of spin lock managers, refer to MVS/XA Diagnostic Techniques.

Suspend Locks

A request for a suspend lock suspends the requestor (if the lock cannot be obtained immediately) to allow that processor to process other work. The owner of a suspend lock can run enabled for I/O or external interruptions and can take a page fault. The local and cross memory services locks are suspend type locks; all others are spin locks.

A description of the global and local suspend locks and their hierarchy are listed below.

Lock Name	Type	Description
CMSSMF	Global	System management facilities cross memory services lock - serializes SMF functions and control blocks.[1]
CMSEQDQ	Global	ENQ/DEQ cross memory services lock - serializes ENQ/DEQ functions and control blocks.[1]
CMS	Global	General cross memory services lock - serializes on more than one address space where this serialization is not provided by one or more of the other global locks. The CMS lock provides global serialization when enablement is required.[1]
CML	Local	Local storage lock - serializes functions and storage within an address space other than the home address space. There is one CML lock per address space.[2]
LOCAL	Local	Local storage lock - serializes functions and storage within a local address space. There is one LOCAL lock per address space.[2]

[1] The cross memory services locks (CMSSMF, CMSEQDQ, and CMS) are equal to each other in the hierarchy.

[2] The CML and LOCAL locks are equal to each other in the hierarchy.

For a more detailed explanation of suspend lock managers, refer to MVS/XA Diagnostic Techniques.

SPIN LOOP TIMEOUT

A spin loop is a situation in which one processor in a multiprocessor environment is unable to communicate with another processor or requires a resource currently held by another processor. The processor that has attempted communication (Px) is the "detecting" or "spinning" processor. The processor that has failed to respond (Py) is the "disabled" or "failing" processor.

The "detecting" processor attempts communication with the "disabled" processor for a period of time that is determined by an excessive spin loop factor located in that processor's PCCA (PCCAXSLF). This factor, computed by SRM during NIP processing, is the approximate number of instructions that the processor will execute in 40 seconds. The spinning program on processor Px calculates its 40 second spin loop by dividing the spin loop factor by the number of instructions in the loop.

During its loop, the spinning processor periodically "opens a window" for a malfunction alert (MFA) or emergency signal (EMS), which would indicate that processor Py has malfunctioned. If a malfunction occurs, alternate CPU recovery (ACR) is invoked on processor Px to take Py offline and free any resources that it holds. Px can then exit from its spin loop and continue processing. If processor Px completes its spin loop without the desired response from processor Py, a spin loop timeout condition exists. The spinning routine (on Px) invokes the excessive spin notification routine (IEEVEXSN) to inform the system operator. The operator has the option of initiating an ACR to remove processor Py from the complex or instructing processor Px to begin spinning again for another forty seconds. This occurs repeatedly until the processor Py releases the required resources or is removed from the complex. If the identified processor (Py) is removed from the complex via alternate CPU recovery (ACR), all global resources held by the identified processor (Py) are released so that the spinning processor (Px) can continue.

INTERSECT SERIALIZATION

The intersect function is used by any routine that alters the dispatching queues. This mechanism indicates to the dispatcher that it should not begin processing until the intersecting function has completed. In the same manner, a routine cannot intersect until the dispatcher has completed. The two levels of intersect are:

- Global — used by any routine that modifies the ASCB queue or the dispatchability of an ASCB. The dispatcher lock must be held before requesting the global intersect.

- Local — used by any routine that modifies the TCB queue or the dispatchability of a TCB. The local lock must be held before requesting the global intersect.

VALIDITY CHECKING

The validity check routine determines whether the storage protect key for a specified address or address range matches the task's assigned protect key.

SUPERVISOR CONTROL RECOVERY

Supervisor control recovery routines can receive control by one of three mechanisms:

- Direct interface with RTM
- Normal SETFRR/ESTAE mechanism
- Supervisor control FRR stack mechanism

Direct Interface With RTM

There are a number of routines (IEAVELCR, IEAVELKR, IEAVESLR, IEAVEVRR) called on SLIH mode entry to RTM to validate certain basic system information.

Normal SETFRR/ESTAE Mechanism

A number of supervisor control functions use the standard SETFRR/ESTAE mechanism to control the recovery environment.

Supervisor Control FRR Stack Mechanism

In order to bypass SETFRR processing on high-performance paths, a multiple FRR stack mechanism is used to provide recovery for some supervisor control routines.

There is a pointer in the PSA to the FRR stack that this processor is using currently. When an error occurs, RTM will route control only to FRRs on that stack. (See the RTM section of the System Logic Library for a description of routing to FRRs.)

For each processor there are eight FRR stacks — a normal stack and seven super stacks, which are used to provide recovery for supervisor control functions. The current stack pointer will always point to one of the stacks.

If the dispatcher or any of the interruption handlers receives control, rather than issuing a SETFRR to establish recovery, it will flip the current stack pointer to point to the appropriate super FRR stack.

If a routine called by a supervisor control function issues a SETFRR, the FRR entry will appear on the current stack. If an error occurs while a super stack is current, then RTM will first route control to all the FRRs on that stack and will then route control to the super FRR routine (IEAVESPR).

See Figure 3 on page SUP-14 for an illustration of the supervisor control recovery data structure.

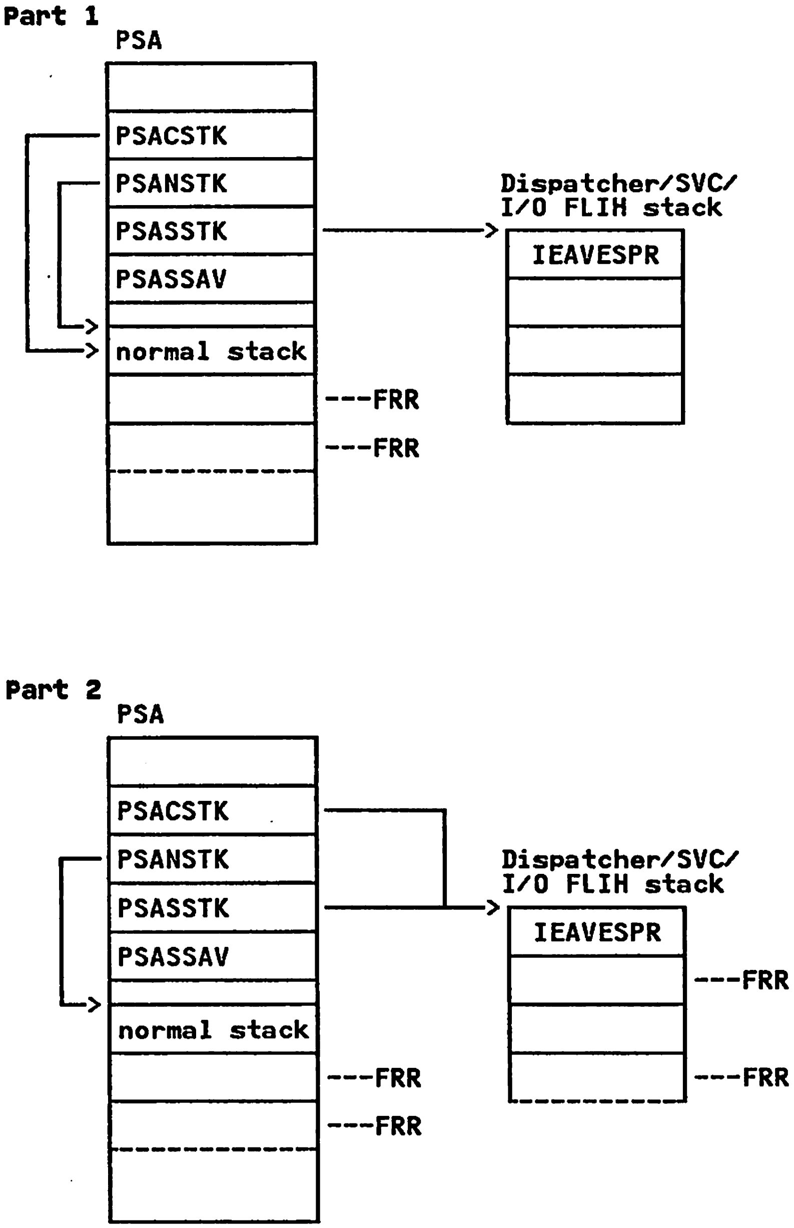


Figure 3. Supervisor Control Recovery Data Structure

MODULE NAMING CONVENTIONS

Each supervisor control module name consists of at least seven characters. The first four characters are 'IEAV'. The remaining characters are an abbreviation of the module description.

ADDRESSING AND RESIDENCY MODES

The addressing and residency mode of the supervisor control modules vary. The modules with their respective addressing and residency modes are listed below.

Module	RMODE	AMODE
IEAVEADV	24	ANY
IEAVEBBR	ANY	31
IEAVCKRS	ANY	31
IEAVCRVF	ANY	31
IEAVECVB	24	ANY
IEAVECMS	24	ANY
IEAVEDR	ANY	31
IEAVEEE0	ANY	31
IEAVEEE2	24	ANY
IEAVEES	ANY	31
IEAVEEXP	24	ANY
IEAVEEXT	ANY	31
IEAVEF00	24	ANY
IEAVEINT	24	ANY
IEAVEIO	24	31
IEAVEJST	ANY	31
IEAVELCR	ANY	31
IEAVELK	24	ANY
IEAVELKR	ANY	31
IEAVEPC0	ANY	31
IEAVEPC	ANY	31
IEAVEPDR	ANY	31
IEAVEPD0	24	31
IEAVEQV0	24	ANY
IEAVERES	ANY	31
IEAVEREX	ANY	31

Module	RMODE	AMODE
IEAVERI	ANY	31
IEAVERP	ANY	31
IEAVESCR	ANY	31
IEAVESC0	24	ANY
IEAVESGP	ANY	31
IEAVESLK	24	ANY
IEAVESLR	ANY	31
IEAVESLX	ANY	31
IEAVESPR	ANY	31
IEAVESRT	ANY	31
IEAVESTU	ANY	31
IEAVESVC	24	ANY
IEAVETCL	ANY	31
IEAVEVAL	24	ANY
IEAVEVS	ANY	31
IEAVESAR	ANY	31
IEAVEVRR	ANY	31
IEAVEXS	ANY	31
IEAVFRLK	ANY	31
IEAVLKRM	ANY	31
IEAVELRM	ANY	31
IEAVSSAF	ANY	31
IEAVMVC0	ANY	31
IEAVSPDM	ANY	31
IEAVVMCH	ANY	31
IEAVVSR	ANY	31
IEAVVSRB	ANY	31

For additional information on addressing and residency mode, see System Programming Library: 31-Bit Addressing.

This section contains problem analysis for the following:

- SRB/SSRB pool manager
- STOP/RESET services
- SUSPEND/RESUME/TCTL services

PROBLEM ANALYSIS FOR THE SRB/SSRB POOL MANAGER

The SRB/SSRB pool manager, IEAVESPM, obtains and frees SRBs from the SRB pool and SSRBs (with their associated XSBs) from the SSRB pool. System routines (in key 0, supervisor state) issue the GETSRB, FREESRB, GETSSRB, and FREESSRB macros to request the pool manager services.

SRB/SSRB POOL MANAGER ENTRY POINTS

The pool manager entry points are:

IEAVSPM1 — entered key 0, supervisor state, enabled for DAT, system mode acceptable to SETFRR (not EUT), no locks required (except the CPU lock might be required for UNCOND and EXPAND type requests).

This entry point is called by the GETSRB macro and obtains an SRB and six-word parameter area from the SRB pool in subpool 245. The SRB is initialized as follows:

— SRB acronym field

— pointer to the parameter area

— FREEMAIN flags, which indicate the origin of the SRB

— other fields and the parameter area cleared.

IEAVSPM2 — entered key 0, supervisor state, enabled for DAT, system mode acceptable to SETFRR (not EUT), no locks required (except the CPU lock might be required).

This entry point is called by the GETSSRB macro and obtains an SSRB and its associated XSB from the SSRB pool in subpool 239. The SSRB/XSB are initialized as follows:

— SSRB acronym field

— pointer to a resource management termination routine (RMTR)

— pointer to the SSRB save area

— SSRB pointer to the XSB

— non-quiesceable and suspended flags set on

— FREEMAIN flags, which indicate the origin of the SSRB/XSB

— XSB acronym field

— other fields cleared.

IEAVSPM3 — entered key 0, supervisor state, enabled for DAT, system mode acceptable to SETFRR (not EUT), no locks required (except the CPU lock might be required).

This entry point is called by the FREESRB macro and frees an SRB and its six-word parameter area. If the specified SRB acronym field is not the same as when the SRB was obtained, the program issuing the macro is abended.

IEAVSPM4 — entered key 0, supervisor state, enabled for DAT, system mode acceptable to SETFRR (not EUT), no locks required (except the CPU lock might be required).

This entry point is called by the FREESSRB macro and frees an SSRB and its XSB. If the specified SSRB acronym field is not the same as when the SSRB was obtained, the program issuing the macro is abended.

SRB/SSRB POOL MANAGER RECOVERY CONSIDERATIONS

When an error occurs, the SRB/SSRB pool manager recovery routine (IEAVSPMR) records information about the error in the SDWA. The queue verifier routine (IEAVEQV1) then uses an SRB/SSRB verification routine in IEAVSPMR to verify that the SRB and SSRB pools are intact. Two tests are used to determine if a given storage area is a valid SRB or SSRB: (1) the storage address must be a valid virtual address, and (2) the acronym field must contain the correct acronym.

If an error is found with a pool, the queue verifier routine attempts to repair the pool, which might include removing invalid SRBs or SSRBs from their pools. Any removed blocks of storage are unavailable for the remainder of the IPL.

Fixed Data

The data that the SRB/SSRB pool manager recovery routines record in the SDWA is:

- SDWAMODN — NUCLEUS, pool manager is nucleus resident.
- SDWACSCT — IEAVESPM, CSECT name.
- SDWAREXN — IEAVESPM, recovery CSECT name.
- SDWACID — SC1C5,component ID.
- SDWASC — descriptive module name.
- SDWAMLVL — module level information.
- SDWARRL — IEAVSPMR, recovery routine label.

Variable Data

The variable data in the SDWAVRA is recorded in the key-length-data format.

- FRR parm area — the six-word parameter area passed to IEAVSPMR by the mainline routine is as follows:
  - Mainline CPU footprint — indicates if the CPU lock was obtained by the pool manager.
  - FRR CPU footprint — indicates if the CPU lock was held on entry to the recovery routine.
  - Return address — contents of register 14 on entry to the pool manager (caller's return address).
- ASCBASID — address space ID of the current ASCB.
- PSATOLD — address of the current TCB.

- General register 14 — contents of register 14 on entry to the mainline pool manager (caller's return address).
- Pool problem information — information recorded by the queue verifier routine if problems are found with the SRB or SSRB pools.
- Lock name — CPU, indicates that the CPU lock was held by the pool manager mainline routine.

SRB/SSRB POOL MANAGER ERROR CONDITIONS

If the IEAVSPM3 (FREESRB) or IEAVSPM4 (FREESSRB) routines are called and an error is detected, completion code X'05A' is issued and the caller is abended. Register 2 contains the address of the invalid SRB or SSRB.

Refer to System Codes for a description of code X'05A' and specific reason codes in register 15.

PROBLEM ANALYSIS FOR STOP/RESET SERVICES

When a unit of work (a current task or SRB) has been dispatched and is executing, the unit of work might need to be suspended. For example, to satisfy a page-in due to a page fault.

System routines (in key 0, supervisor state) use the stop/reset service to suspend and then reset a unit of work. The caller is not required to have addressability to the home address space to suspend a unit of work, and is not required to have addressability to the address space containing the unit of work to reset the unit of work.

STOP/RESET ENTRY POINTS

The stop/reset entry points are:

IEAVSUSC — entered disabled, key 0, supervisor state, no locks required.

This entry point is called by the paging supervisor to suspend a current task or SRB because a page fault occurred, or by system routines (other then the paging supervisor) to suspend the current task or SRB.

IEAVSUSF — entered disabled, key 0, supervisor state, no locks required.

This entry point is called by system routines to suspend the current task or SRB. The caller may specify a number of FRRs not to be copied when the normal stack is saved.

IEAVRSTC — entered disabled, key 0, supervisor state, no locks required.

This entry point is called by the paging supervisor or other system routines to reset a task or SRB that was suspended.

STOP/RESET RECOVERY CONSIDERATIONS

The stop recovery routine (STOPFRR) records information about the error and, depending on the error, either attempts to restore the system and unit of work to a consistent state, or attempts to complete the stop function.

The reset recovery routine (RESETFRR) records information about the error and then attempts to complete the reset function.

The reset STERM, reset schedule, and reset SRB recovery routine (IEAVSCHF) frees the SRB (if one was obtained but not scheduled), clears the stop/reset super bit (PSASTPRT), and releases the LOCAL lock (if the LOCAL lock was obtained by the calling routine).

Fixed Data

The data that the stop/reset recovery routines record in the SDWA is:

- SDWAMODN — NUCLEUS, stop/reset is nucleus resident.
- SDWACSCT — IEAVESRT, CSECT name.
- SDWAREXN — IEAVESRT, recovery routine.
- SDWACID — SC1C5, component ID.
- SDWAMLVL — module level information.
- SDWARRL — STOPFRR, RESETFRR, or IEAVSCHF, label of the recovery routine.

Variable Data

The variable data in the SDWA is recorded in the key-length-data format. The variable data recorded by the recovery routine is:

For the stop recovery (STOPFRR) and the reset recovery (RESETFRR) routines:

- FRR parm area - the six-word parameter area passed to STOPFRR and RESETFRR by the mainline routine is as follows:

  - General register 13 - caller's register save area address.

  - TCB/SSRB address - address of the TCB or SSRB to be reset.

  - RB address - address of the RB if a TCB is to be reset.

  - Request code - type of reset requested (conditional, unconditional, or page I/O error), or completion code for a termination reset.

  - Flag byte - if X'80', recovery has been entered recursively.

- ASCBASID - address space ID of the current ASCB.

- PSATOLD - address of the current TCB.

- General register 14 - contents of register 14 on entry to the mainline stop routine (caller's return address).

- General registers - for STOPFRR, contents of the original registers, if they were changed by the recovery routine.

- Abend code - for STOPFRR, the original abend code, if it was changed by the recovery routine.

For the reset STERM, reset schedule, and reset SRB recovery routine (IEAVSCHF):

- FRR parm area - the six-word parameter area passed to IEAVSCHF by the mainline routine as follows:

  - SSRB address - address of the SSRB associated with the scheduled SRB. (Note that an SSRB is obtained, made to look like an SRB, and scheduled as an SRB. The remainder of the SSRB is used as a work area by the scheduled routine. The SSRB is restored to an SSRB before it is returned to the SSRB pool.)

  - Flag byte - recovery footprint flags:

    X'80' - stop/reset super bit footprint, indicates the mainline code set the bit.

    X'40' - LOCAL lock footprint, indicates the mainline code had obtained the LOCAL lock and had not released it before the error occurred.

- ASCBASID - address space ID of the current ASCB.

- PSATOLD - address of the current TCB.

- If an SRB was obtained but not scheduled, the following are also present in the SDWAVRA:

  - IHASRB - identifies the following control block.

  - SRB - contents of the unscheduled SRB.

  - SRB parm area header - describes the following six-word parameter area.

  - SRB parm area - contents of the SRB parameter area.

- Lock name - LOCAL, indicates the LOCAL lock was held.

STOP/RESET ERROR CONDITIONS

The stop/reset services issue the X'059' completion code when an error exists, and abnormally terminates the program requesting the service.

Refer to System Codes for a description of code X'059' and specific reason codes in register 15.

PROBLEM ANALYSIS FOR SUSPEND/RESUME/TCTL SERVICES

The SUSPEND/RESUME/TCTL services are used to place an unlocked task in a suspended state (SUSPEND), resume an unlocked task from a suspended state (RESUME), and to transfer control from an SRB to an unlocked task (TCTL). These macros can only be issued by key 0, supervisor state routines.

SUSPEND can be issued in any cross memory mode and in task mode; it places the caller in a suspended state. Control is returned to the caller and the task is suspended only when the task incurs an interruption or enters the dispatcher (such as via CALLDISP).

RESUME can be issued in any cross memory mode and in SRB or task mode, with current addressability to the address space of the TCB that is to be resumed.

TCTL can be issued in SRB mode and home mode, with current addressability to the address space of the task to which control is to be transferred.

SUSPEND/RESUME/TCTL ENTRY POINTS

The SUSPEND/RESUME/TCTL entry points are:

IEAVSPND — entered enabled or disabled, key 0, supervisor state, task mode, no locks held, any cross memory state.

This entry point is called by the SUSPEND macro and places the current TCB/RB or previous RB is a suspended state.

IEAVRSUH — entered enabled, key 0, supervisor state, no locks held, SRB or task mode, home mode.

This entry point is called by the RESUME macro to resume a task in the home address space. This entry point performs an unconditional synchronous resume function. The caller must execute enabled and hold no locks unless the LOCAL lock is already held, because this entry point can require the LOCAL lock to serialize the resume function. This is the only entry point where RETURN=N can be specified to indicate that control should be transferred from the calling SRB to the resumed task.

IEAVRSUS — entered enabled, key 0, supervisor state, no locks held, SRB or task mode, any cross memory state, current addressability to the resumed TCB.

This entry point is called by the RESUME macro to resume a task in the address space specified by the input. Current addressability to the task to be resumed must have been established by the caller. This entry point performs an unconditional synchronous resume function. The caller must execute enabled and hold no locks unless the LOCAL lock of the address space of the resumed TCB is already held, because this entry point can require the specified lock to serialize the resume function.

IEAVRSCS — entered enabled or disabled, key 0, supervisor state, SRB or task mode, any cross memory state, locks can be held, current addressability to the resumed TCB.

This entry point is called by the RESUME macro to resume a task in the address space specified by the input. Current addressability to the task to be resumed must have been established by the caller. This entry point performs a conditional synchronous resume function. If serialization to perform the resume function is not available, the function is not performed and the caller receives a nonzero return code.

IEAVRSUA — entered enabled or disabled, key 0, supervisor state, SRB or task mode, any cross memory state, locks can be held, current addressability to the resumed TCB.

This entry point is called by the RESUME macro to resume a task in the address space specified by the input. Current addressability to the task to be resumed must have been established by the caller. This entry point performs an unconditional asynchronous resume function. If serialization to perform the resume function is not available, an SRB is obtained and, asynchronously, an unconditional synchronous function is performed; a nonzero return code is returned to the caller.

IEAVRSCA — entered enabled or disabled, key 0, supervisor state, SRB or task mode, any cross memory state, locks can be held, current addressability to the resumed TCB.

This entry point is called by the RESUME macro to resume a task in the address space specified by the input. Current addressability to the task to be resumed must have been established by the caller. This entry point performs a conditional asynchronous resume function. If serialization to perform the resume function is not available, an SRB is obtained conditionally, and if successful, asynchronously scheduled to perform an unconditional synchronous resume function. Return codes are returned to the caller to indicate whether the SRB could be obtained or not obtained.

IEAVTCTL — entered enabled or disabled, key 0, supervisor state, SRB mode, home mode, no locks held.

This entry point is called by the TCTL macro to transfer control from an SRB to a task in the home address space.

Note: Module IEAVETCL performs the functions just described. IEAVETCL resides above the 16 megabyte line and executes in 31-bit addressing mode. To allow programs that execute in 24-bit addressing mode to use SUSPEND/RESUME/TCTL services, the macros actually invoke entry points of IEAVEGLU, an addressing mode interface module. IEAVEGLU makes the transition to 31-bit addressing mode, if necessary, and invokes a corresponding entry point in IEAVETCL, where the function is performed. The following table shows the relationship between the entry points in IEAVEGLU and IEAVETCL.

IEAVEGLU Entry Point	**IEAVETCL Entry Point**
IEAVSPND	IEAVSPN1
IEAVRSUH	IEAVRSH1
IEAVRSUS	IEAVRSS1
IEAVRSCS	IEAVRSC1
IEAVRSUA	IEAVRSU1
IEAVRSCA	IEAVRSA1
IEAVTCTL	IEAVTCT1

IEAVETCR — entered disabled, key 0, supervisor state, any cross memory state, SRB or task mode.

This entry point is called by RTM and performs recovery processing for the resume and transfer control functions.

RESUME/TCTL RECOVERY CONSIDERATIONS

RESUME/TCTL processing is protected by an FRR (IEAVETCR) that receives control from RTM when an error occurs. The FRR records debugging information in the SDWA, attempts to restore the system and unit of work to a consistent state, and then percolates to the caller's recovery routine.

Fixed Data

The data that the RESUME/TCTL recovery routine recorded in the SDWA is:

- SDWAMODN — NUCLEUS, nucleus load module.
- SDWACSCT — IEAVETCL, mainline microfiche name.
- SDWAREXN — IEAVETCL, recovery microfiche name.
- SDWACID — SC1C5, component ID.
- SDWASC — RESUME or TCTL ABEND, subfunction in error.
- SDWAMLVL — module level information.
- SDWARRL — IEAVETCR, recovery routine name.

Variable Data

Variable data in the SDWAVRA is recorded in the key-length-data format. A header contains RSLG in key-length-data format followed by the RSLG mapping in key-length-data format as follows:

Offset	RESUME data	TCTL data
X'00'	flag byte: Bit 0=0,RESUME	flag byte: Bit 0=1,TCTL
X'01'	0	SVTDACTV
X'02'	flag byte: Bit 0=1,lock obtained Bit 1=1,SRB obtained	0
X'04'	PSASUPER	PSASUPER
X'08'	PSACSTK	PSACSTK
X'0C'	PSATOLD	PSATOLD
X'10'	PSAAOLD	PSAAOLD
X'14'	input TCB address	input TCB address
X'18'	input ASCB address	SVTDSREQ
X'1C'	PSAHLHI	ASCBSRQ
X'20'	home address space ID	—
X'22'	0	—
X'24'	ASCBTCBS	—
X'28'	address of SRB, or 0	—

SUSPEND/RESUME/TCTL ERROR CONDITIONS

The SUSPEND, RESUME, and TCTL macros issue the X'070' completion code when an error condition exists, and abnormally terminate the program issuing the macro.

Refer to System Codes for a description of code X'070' and specific reason codes in register 15.

CONTROL BLOCK OVERVIEW

Due to the complexity of the control block overview for Supervisor Control, it has been divided into two sections.

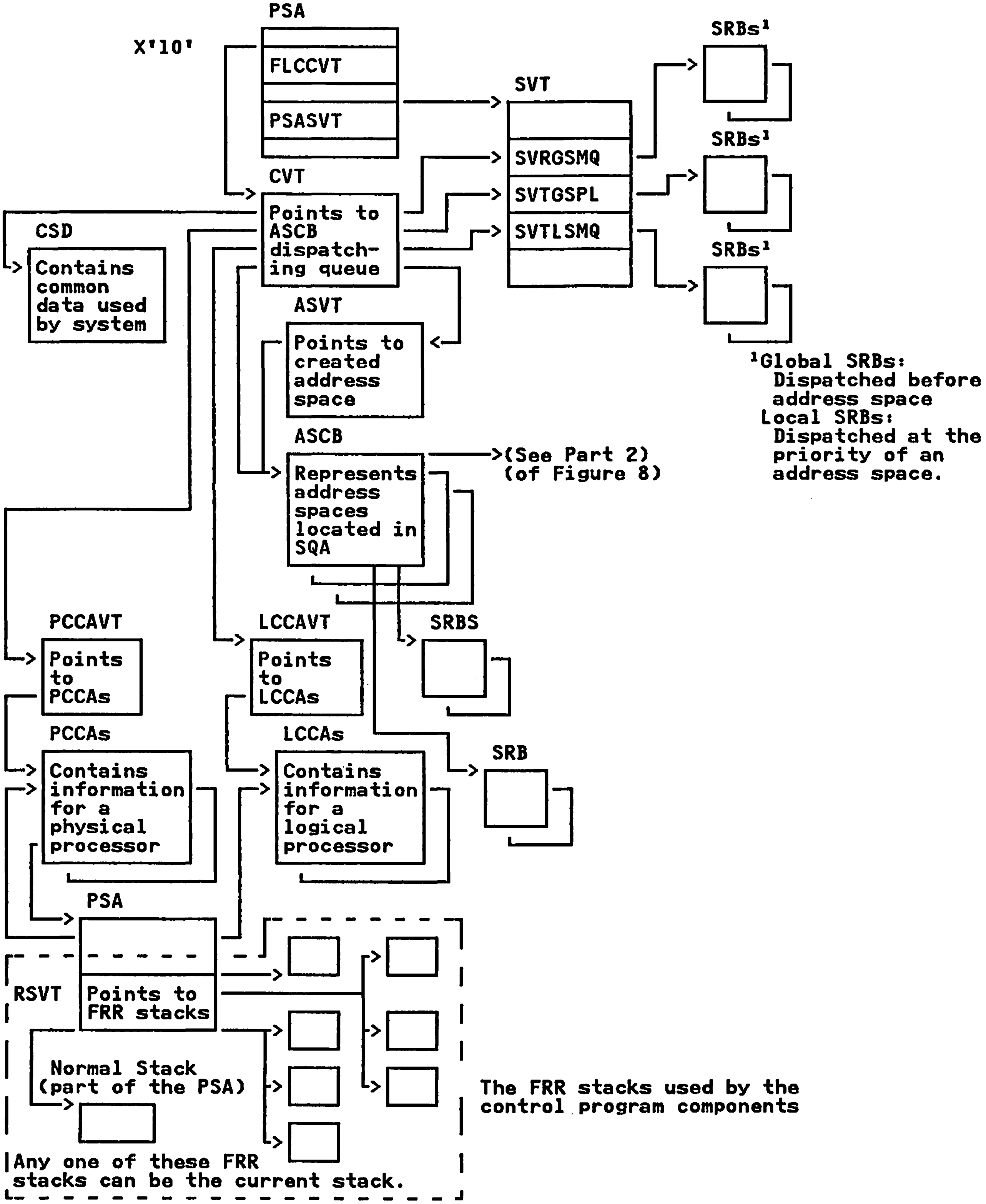

PSA
X'10'
FLCCVT
PSASVT
SVT
SVRGSMQ
SVTGSPL
SVTLSMQ
SRBs[1]
SRBs[1]
SRBs[1]
CVT
Points to ASCB dispatching queue
CSD
Contains common data used by system
ASVT
Points to created address space
ASCB
Represents address spaces located in SQA
(See Part 2) (of Figure 8)
[1]Global SRBs: Dispatched before address space
Local SRBs: Dispatched at the priority of an address space.
PCCAVT
Points to PCCAs
LCCAVT
Points to LCCAs
SRBS
SRB
PCCAs
Contains information for a physical processor
LCCAs
Contains information for a logical processor
PSA
RSVT
Points to FRR stacks
Normal Stack (part of the PSA)
Any one of these FRR stacks can be the current stack.
The FRR stacks used by the control program components

Figure 4 (Part 1 of 2). Control Block Overview

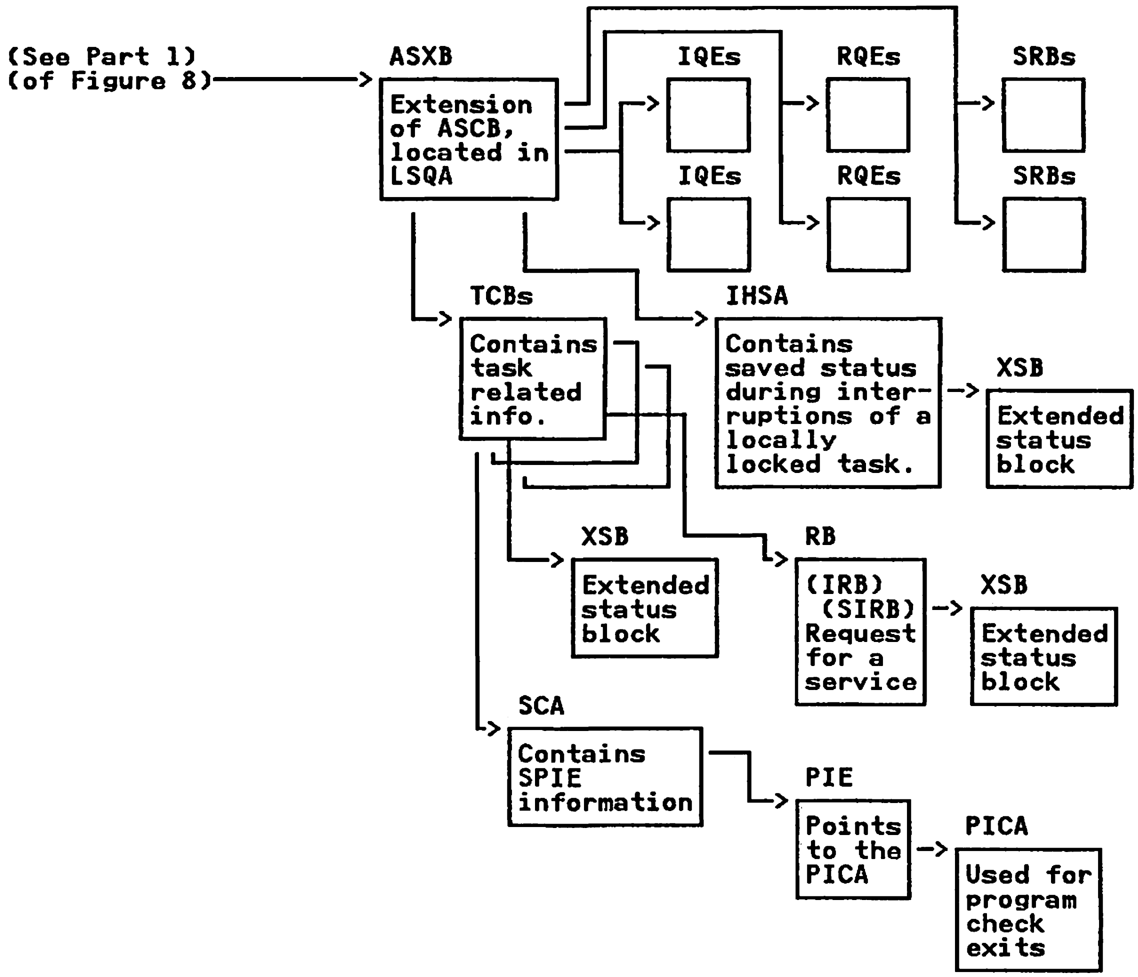

(See Part 1)
(of Figure 8)
ASXB
Extension of ASCB, located in LSQA
IQEs
IQEs
RQEs
RQEs
SRBs
SRBs
TCBs
Contains task related info.
IHSA
Contains saved status during interruptions of a locally locked task.
XSB
Extended status block
XSB
Extended status block
RB
(IRB) (SIRB) Request for a service
XSB
Extended status block
SCA
Contains SPIE information
PIE
Points to the PICA
PICA
Used for program check exits

Figure 4 (Part 2 of 2). Control Block Overview

PROCESS FLOW

The module flows for Supervisor Control are included in this section. Each figure contains the calling module or routine, the modules called, and the exit for each specific module.

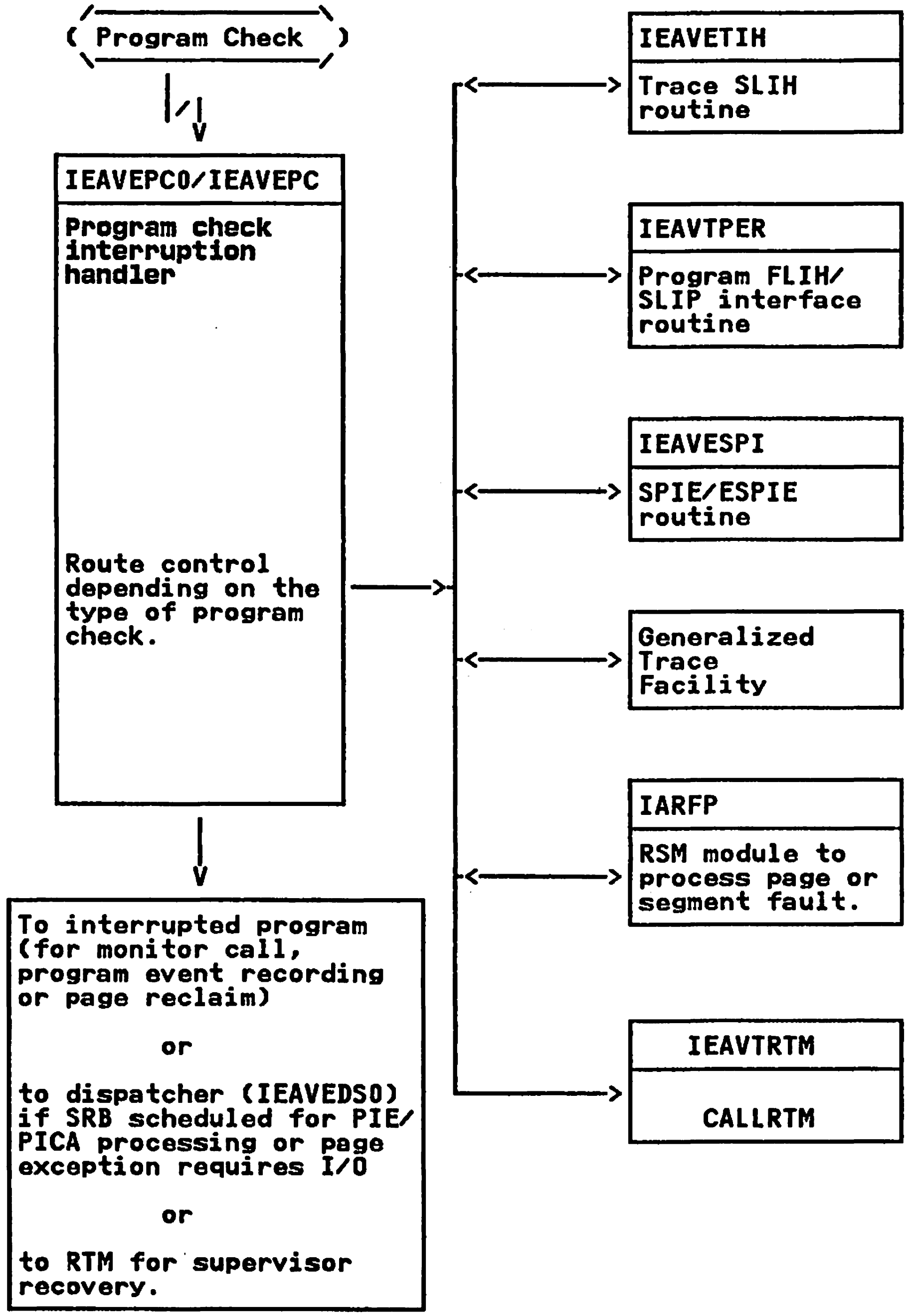

Program Check
IEAVEPC0/IEAVEPC
Program check interruption handler
Route control depending on the type of program check.
To interrupted program (for monitor call, program event recording or page reclaim)
or
to dispatcher (IEAVEDS0) if SRB scheduled for PIE/PICA processing or page exception requires I/O
or
to RTM for supervisor recovery.
IEAVETIH
Trace SLIH routine
IEAVTPER
Program FLIH/ SLIP interface routine
IEAVESPI
SPIE/ESPIE routine
Generalized Trace Facility
IARFP
RSM module to process page or segment fault.
IEAVTRTM
CALLRTM

Figure 5. Program Check Interruption Handlers

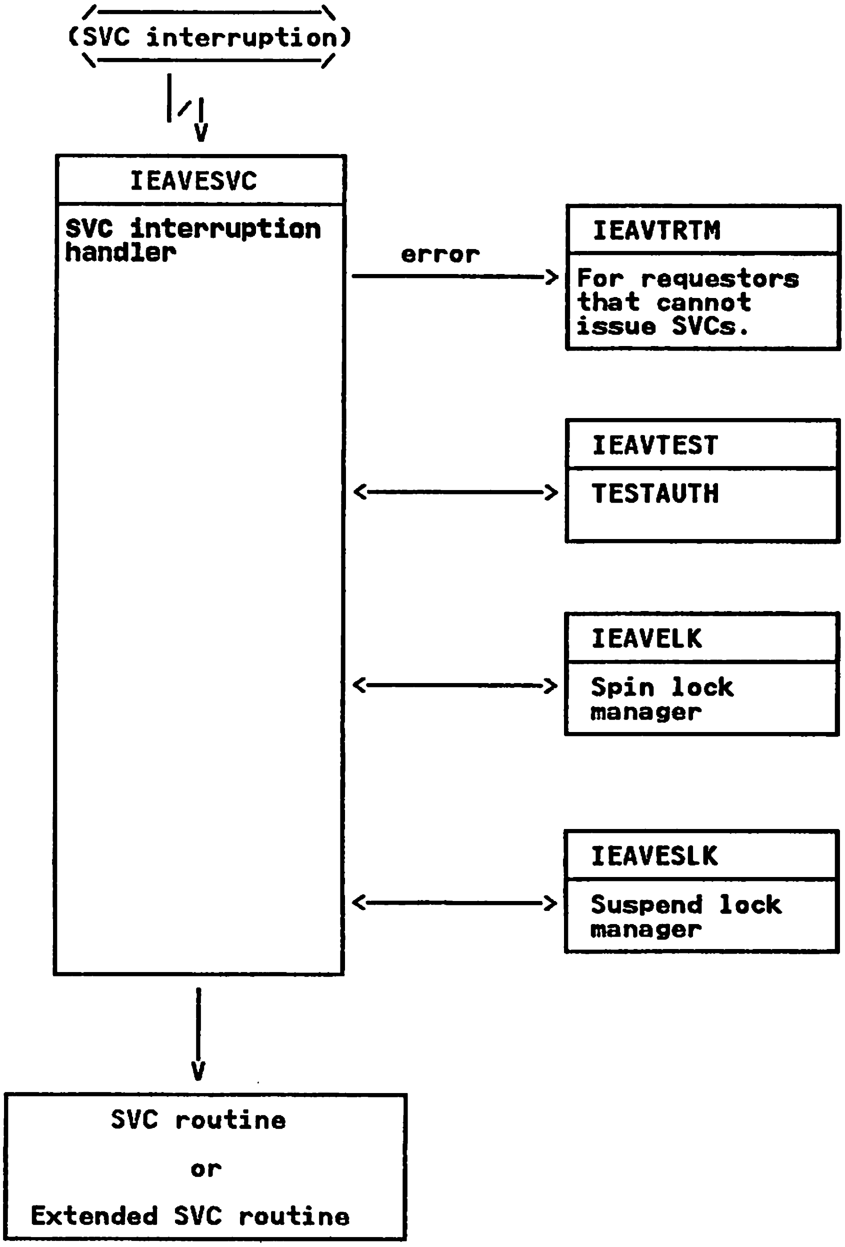


Figure 6. SVC Interruption Handler Process Flow

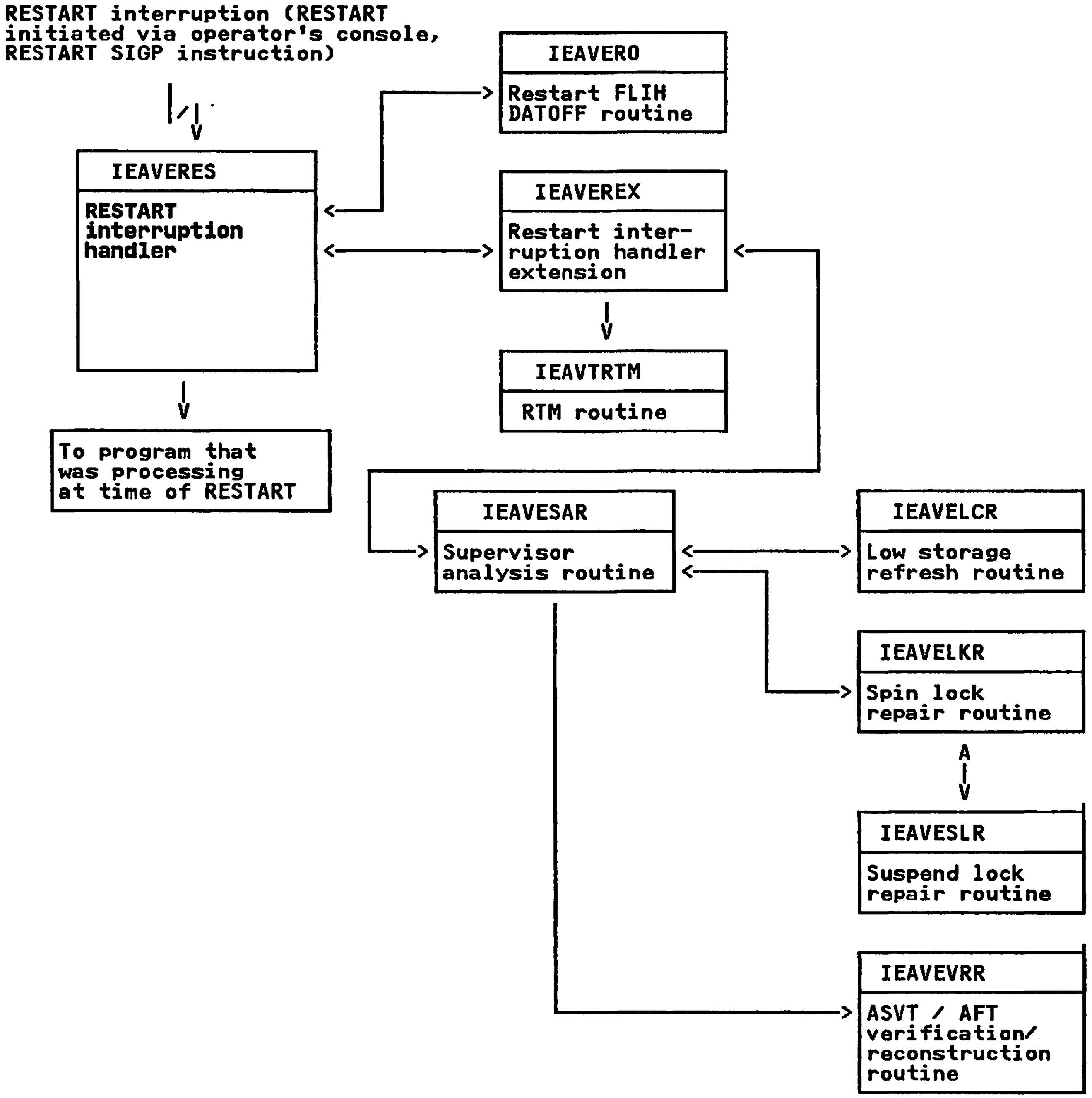

RESTART interruption (RESTART initiated via operator's console, RESTART SIGP instruction)
IEAVERO
Restart FLIH DATOFF routine
IEAVERES
RESTART interruption handler
IEAVEREX
Restart interruption handler extension
IEAVTRTM
RTM routine
To program that was processing at time of RESTART
IEAVESAR
Supervisor analysis routine
IEAVELCR
Low storage refresh routine
IEAVELKR
Spin lock repair routine
IEAVESLR
Suspend lock repair routine
IEAVEVRR
ASVT / AFT verification/ reconstruction routine

Figure 7. RESTART Interruption Handler Process Flow

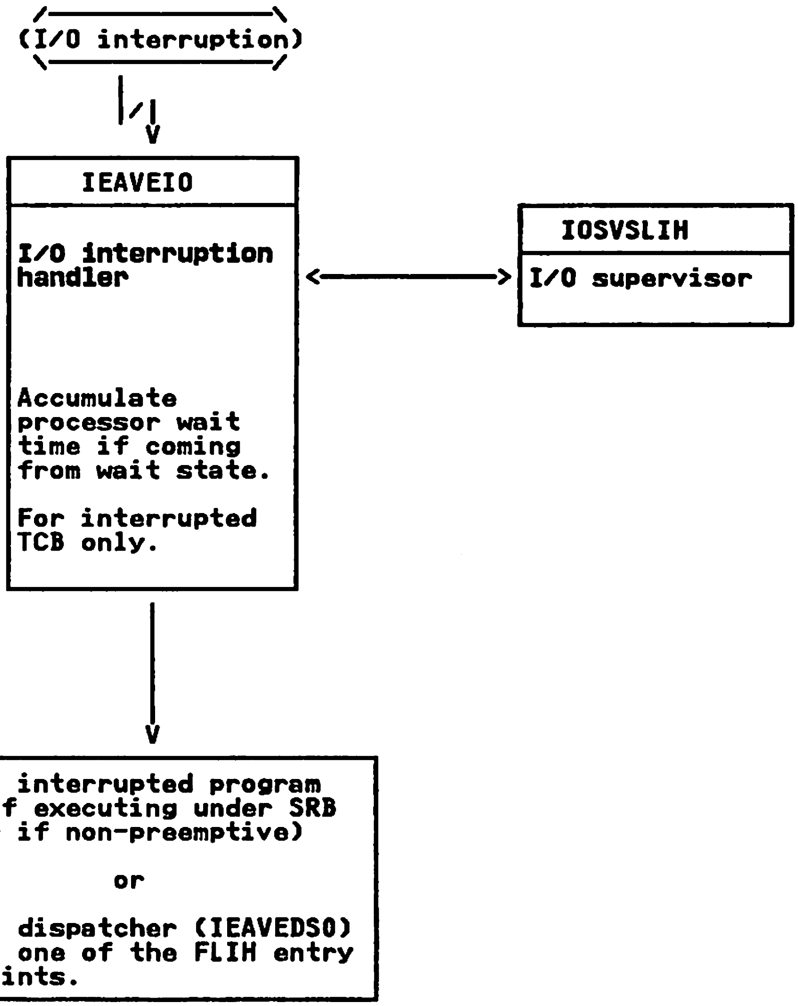

(I/O interruption)
IEAVEIO
I/O interruption handler
Accumulate processor wait time if coming from wait state.
For interrupted TCB only.
IOSVSLIH
I/O supervisor
To interrupted program (if executing under SRB or if non-preemptive)
or
to dispatcher (IEAVEDS0) at one of the FLIH entry points.

Figure 8. I/O Interruption Handler Process Flow

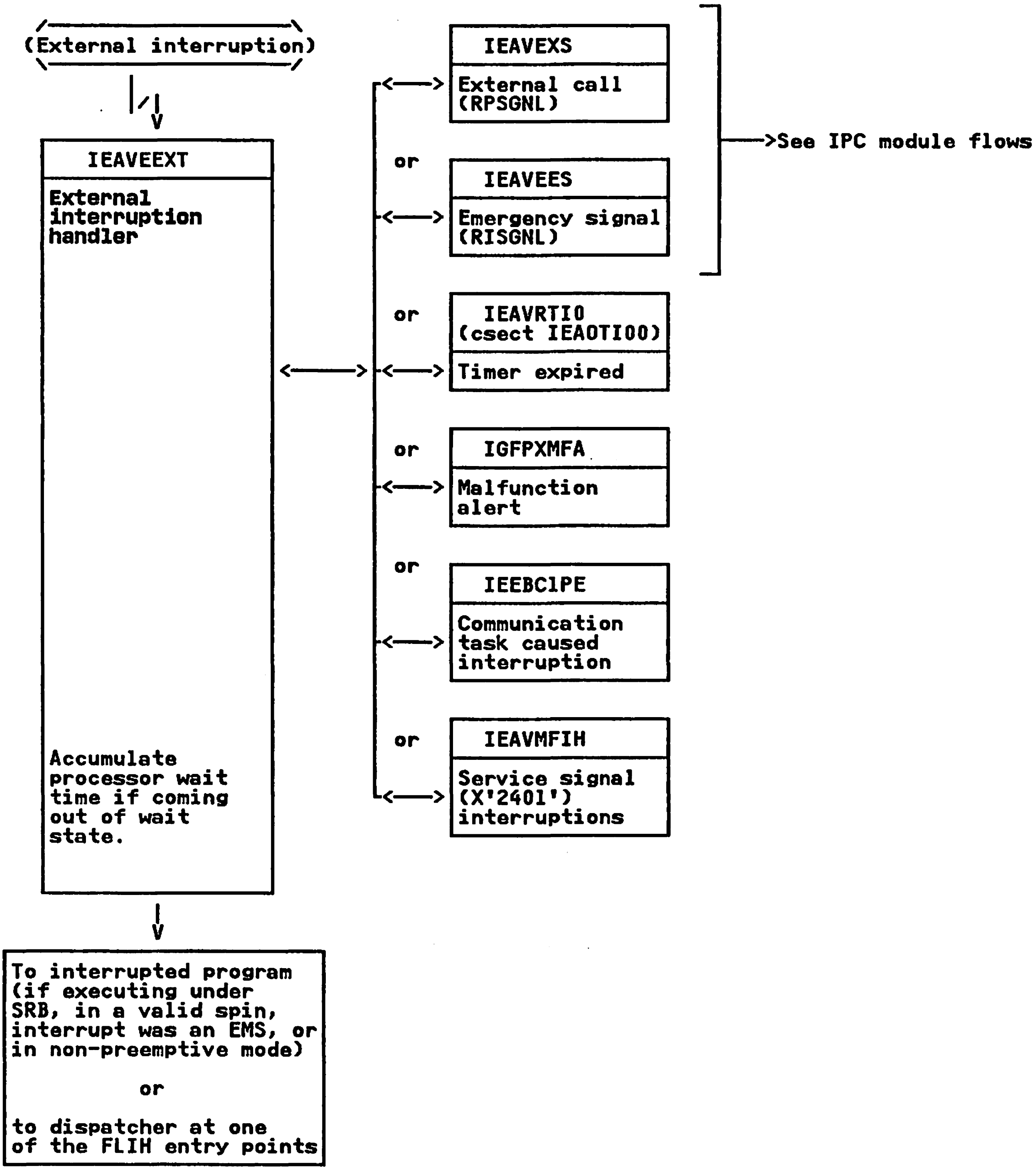

External interruption
IEAVEEXT
External interruption handler
Accumulate processor wait time if coming out of wait state.
IEAVEXS
External call (RPSGNL)
or
IEAVEES
Emergency signal (RISGNL)
See IPC module flows
or
IEAVRTI0 (csect IEAOTI00)
Timer expired
or
IGFPXMFA
Malfunction alert
or
IEEBC1PE
Communication task caused interruption
or
IEAVMFIH
Service signal (X'2401') interruptions
To interrupted program (if executing under SRB, in a valid spin, interrupt was an EMS, or in non-preemptive mode)
or
to dispatcher at one of the FLIH entry points

Figure 9. External Interruption Handler Process Flow

Emergency signal (RISGNL)

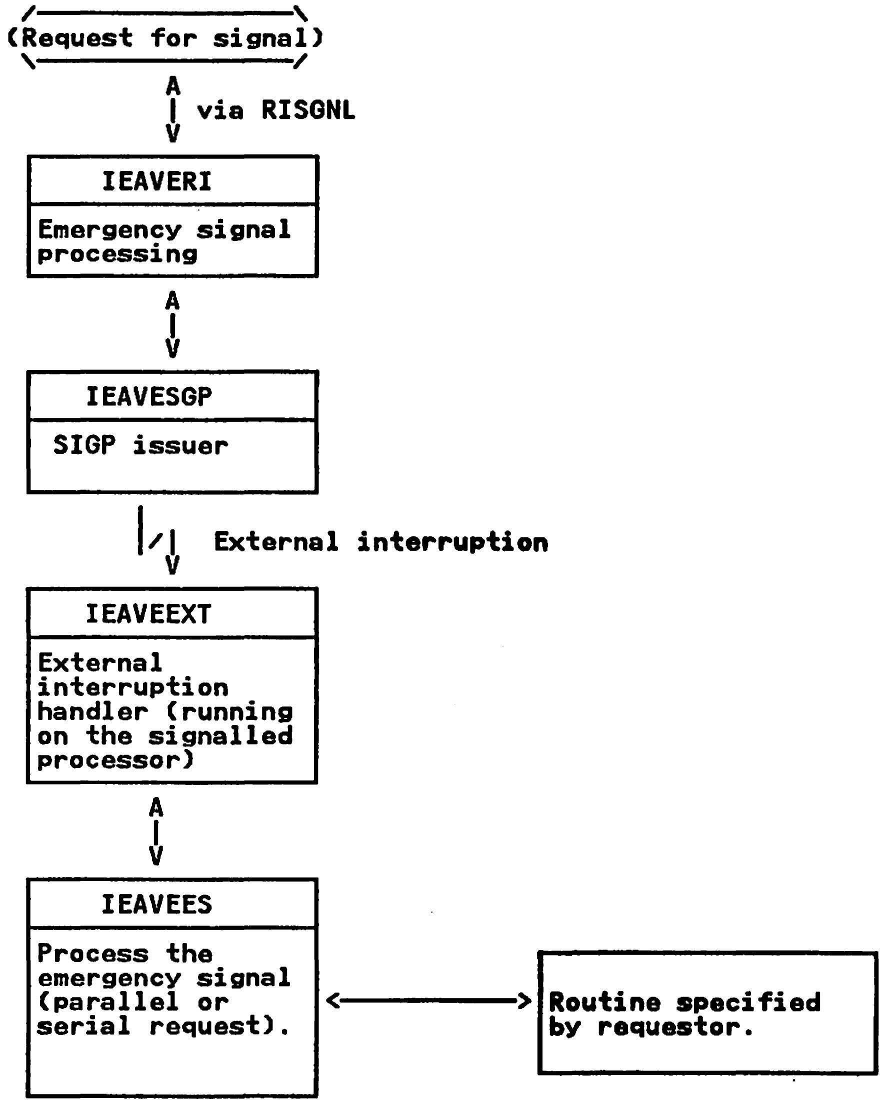

Request for signal
via RISGNL
IEAVERI
Emergency signal processing
IEAVESGP
SIGP issuer
External interruption
IEAVEEXT
External interruption handler (running on the signalled processor)
IEAVEES
Process the emergency signal (parallel or serial request).
Routine specified by requestor.

Figure 10. Interprocessor Communication (IPC) Remote Immediate Signal Process Flow

Emergency call (RPSGNL)

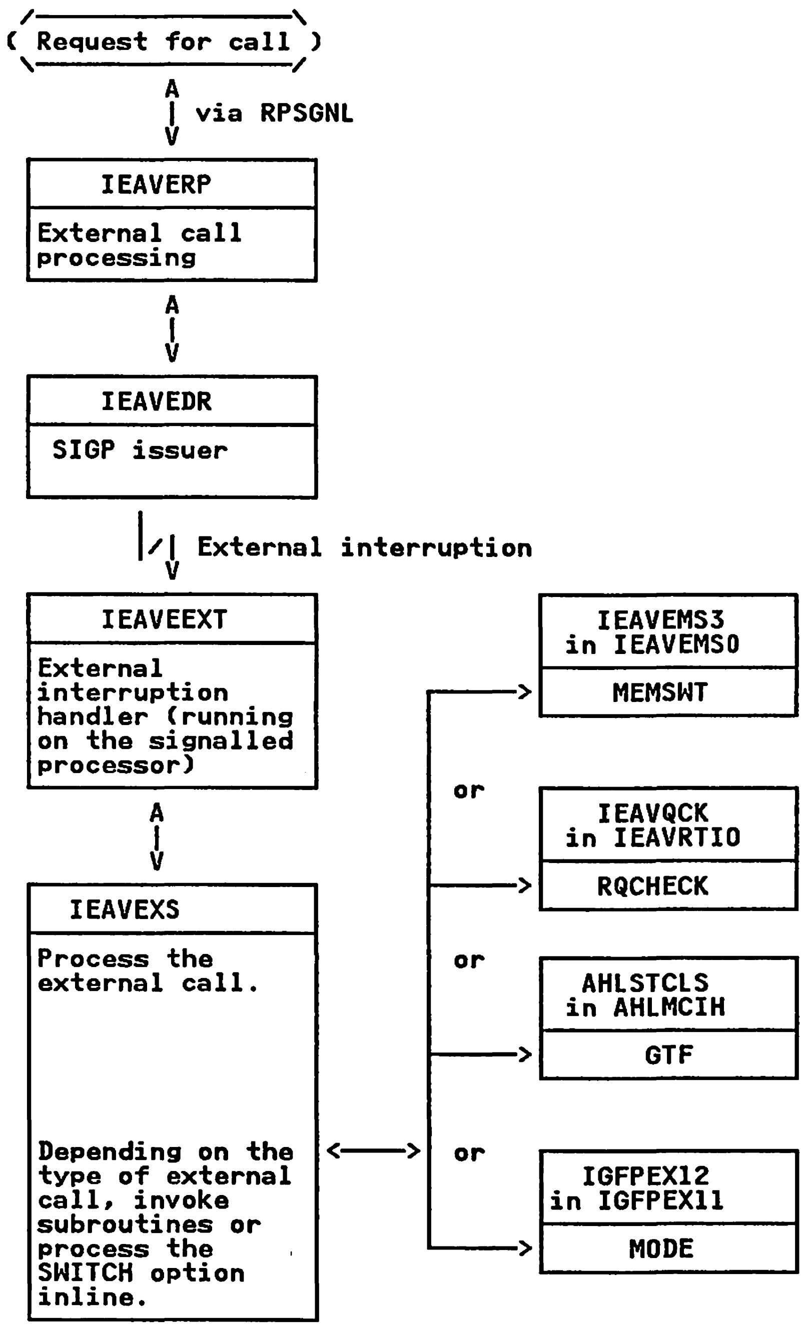


Figure 11. Interprocessor Communication (IPC) Remote Pendable Signal Process Flow

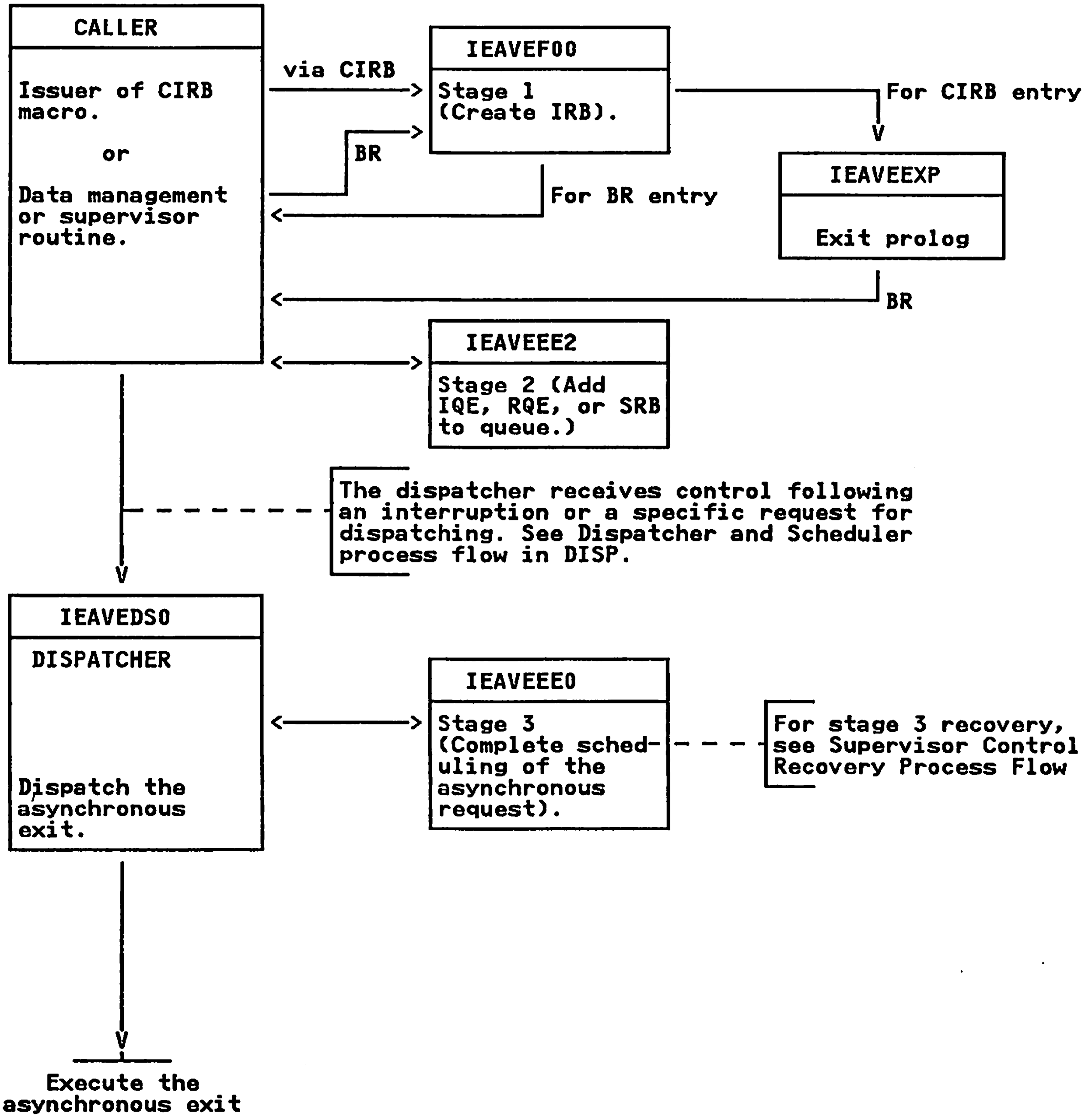

CALLER
Issuer of CIRB macro.
or
Data management or supervisor routine.
via CIRB
BR
IEAVEF00
Stage 1 (Create IRB).
For CIRB entry
For BR entry
IEAVEEXP
Exit prolog
BR
IEAVEEE2
Stage 2 (Add IQE, RQE, or SRB to queue.)
The dispatcher receives control following an interruption or a specific request for dispatching. See Dispatcher and Scheduler process flow in DISP.
IEAVEDS0
DISPATCHER
Dispatch the asynchronous exit.
IEAVEEE0
Stage 3 (Complete scheduling of the asynchronous request).
For stage 3 recovery, see Supervisor Control Recovery Process Flow
Execute the asynchronous exit

Figure 12. Exit Effectors for Asynchronous Exits Process Flow

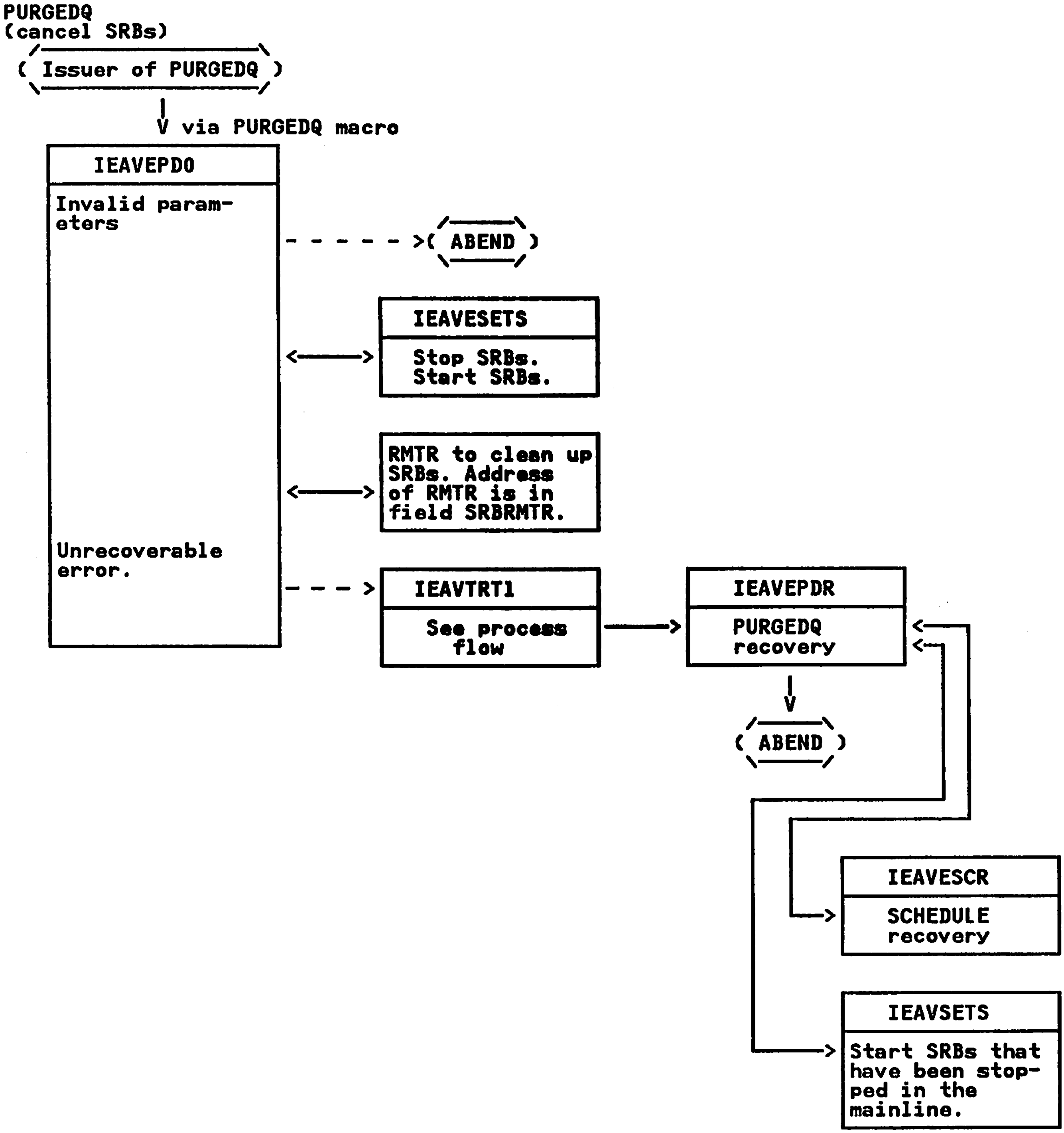


Figure 13. PURGEDQ Process Flow

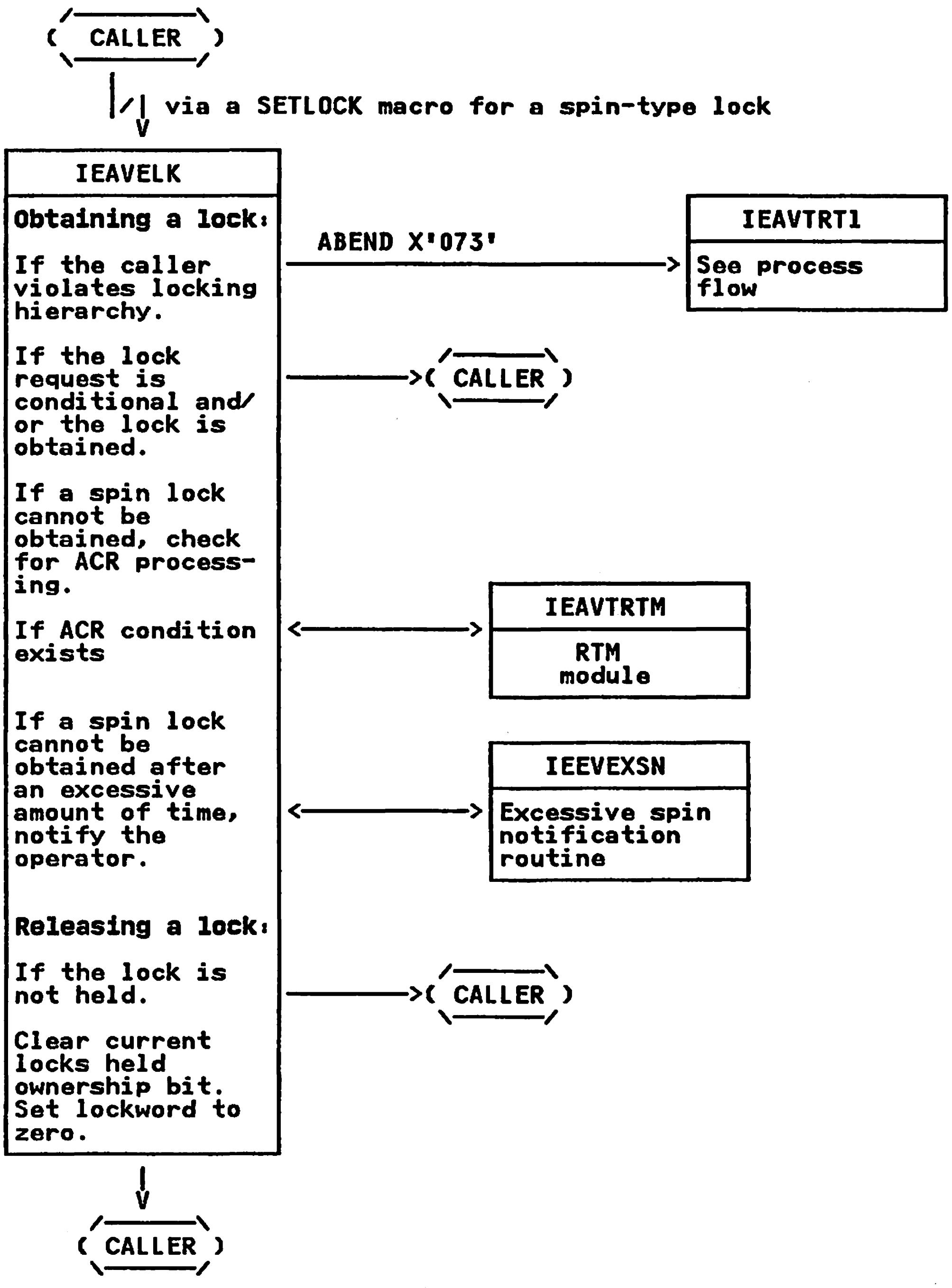
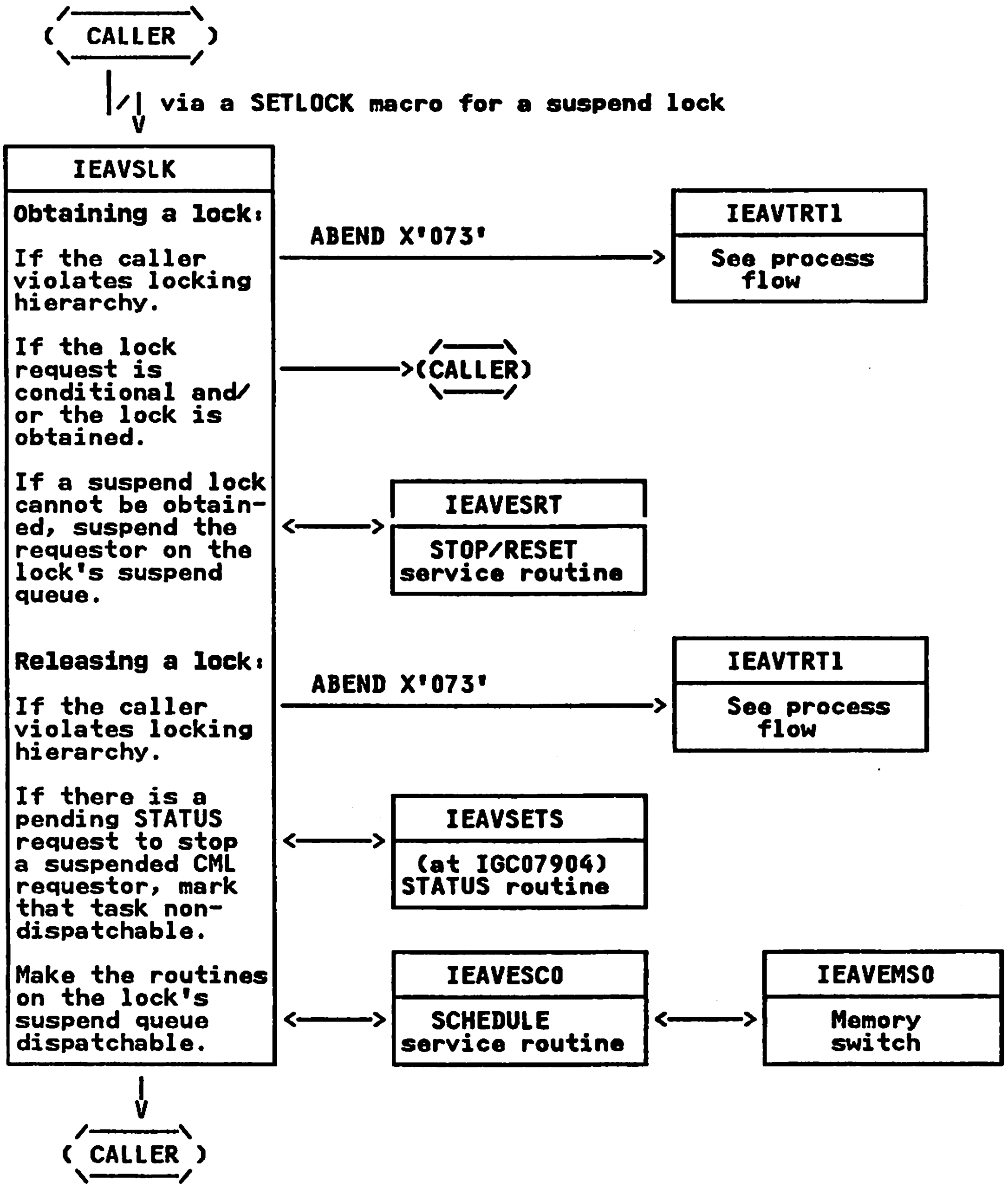

CALLER
via a SETLOCK macro for a spin-type lock
IEAVELK
Obtaining a lock:
If the caller violates locking hierarchy.
ABEND X'073'
IEAVTRT1
See process flow
If the lock request is conditional and/ or the lock is obtained.
CALLER
If a spin lock cannot be obtained, check for ACR processing.
If ACR condition exists
IEAVTRTM
RTM module
If a spin lock cannot be obtained after an excessive amount of time, notify the operator.
IEEVEXSN
Excessive spin notification routine
Releasing a lock:
If the lock is not held.
CALLER
Clear current locks held ownership bit. Set lockword to zero.
CALLER

Figure 14. Spin Lock Manager Process Flow



Obtaining a lock:

If the caller violates locking hierarchy.

If the lock request is conditional and/or the lock is obtained.

If a suspend lock cannot be obtained, suspend the requestor on the lock's suspend queue.

Releasing a lock:

If the caller violates locking hierarchy.

If there is a pending STATUS request to stop a suspended CML requestor, mark that task non-dispatchable.

Make the routines on the lock's suspend queue dispatchable.

Figure 15. Suspend Lock Manager Process Flow

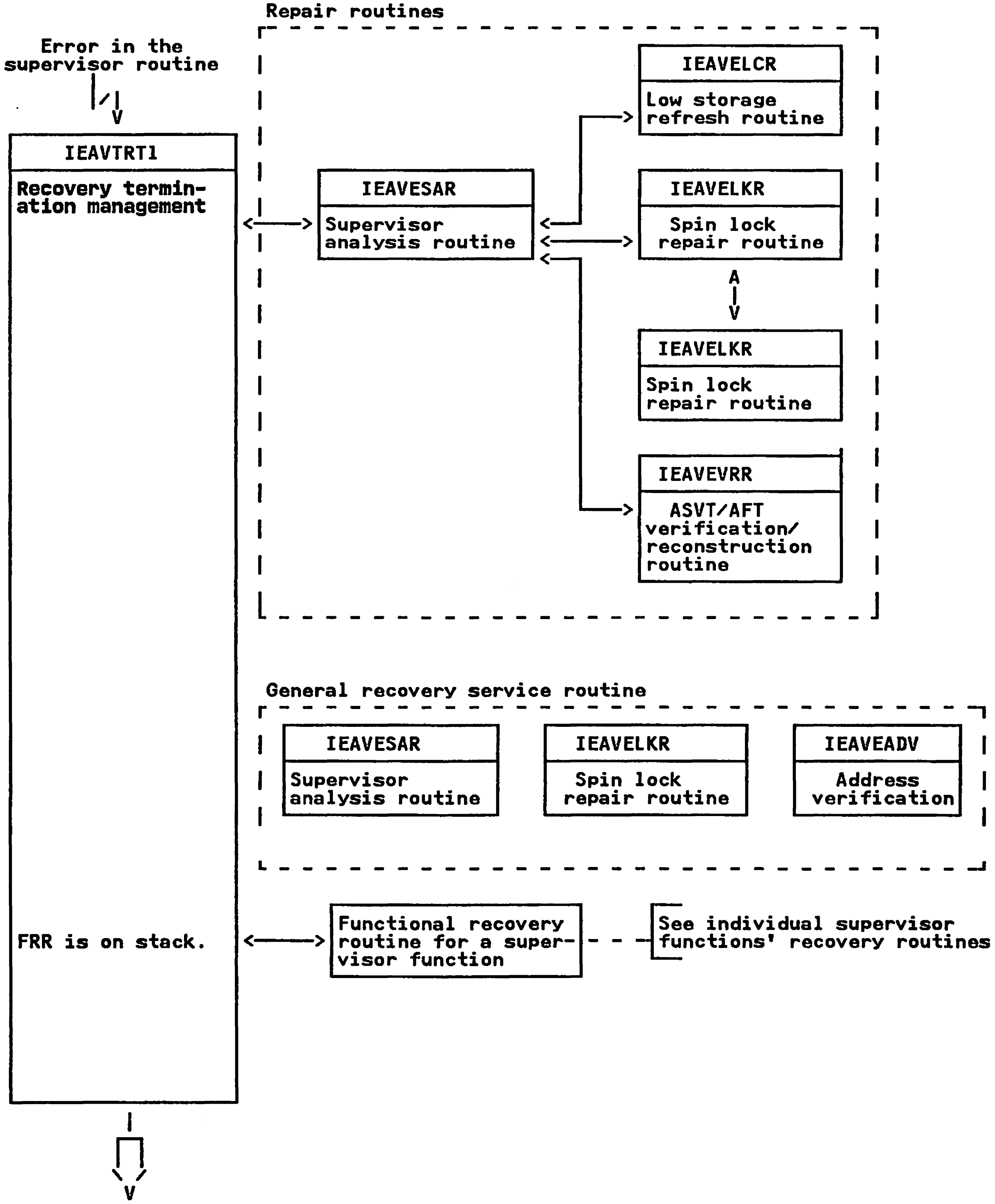

Repair routines
Error in the supervisor routine
IEAVTRT1
Recovery termination management
IEAVESAR
Supervisor analysis routine
IEAVELCR
Low storage refresh routine
IEAVELKR
Spin lock repair routine
IEAVELKR
Spin lock repair routine
IEAVEVRR
ASVT/AFT verification/ reconstruction routine
General recovery service routine
IEAVESAR
Supervisor analysis routine
IEAVELKR
Spin lock repair routine
IEAVEADV
Address verification
FRR is on stack.
Functional recovery routine for a supervisor function
See individual supervisor functions' recovery routines

Figure 16 (Part 1 of 2). Supervisor Control Recovery Process Flow

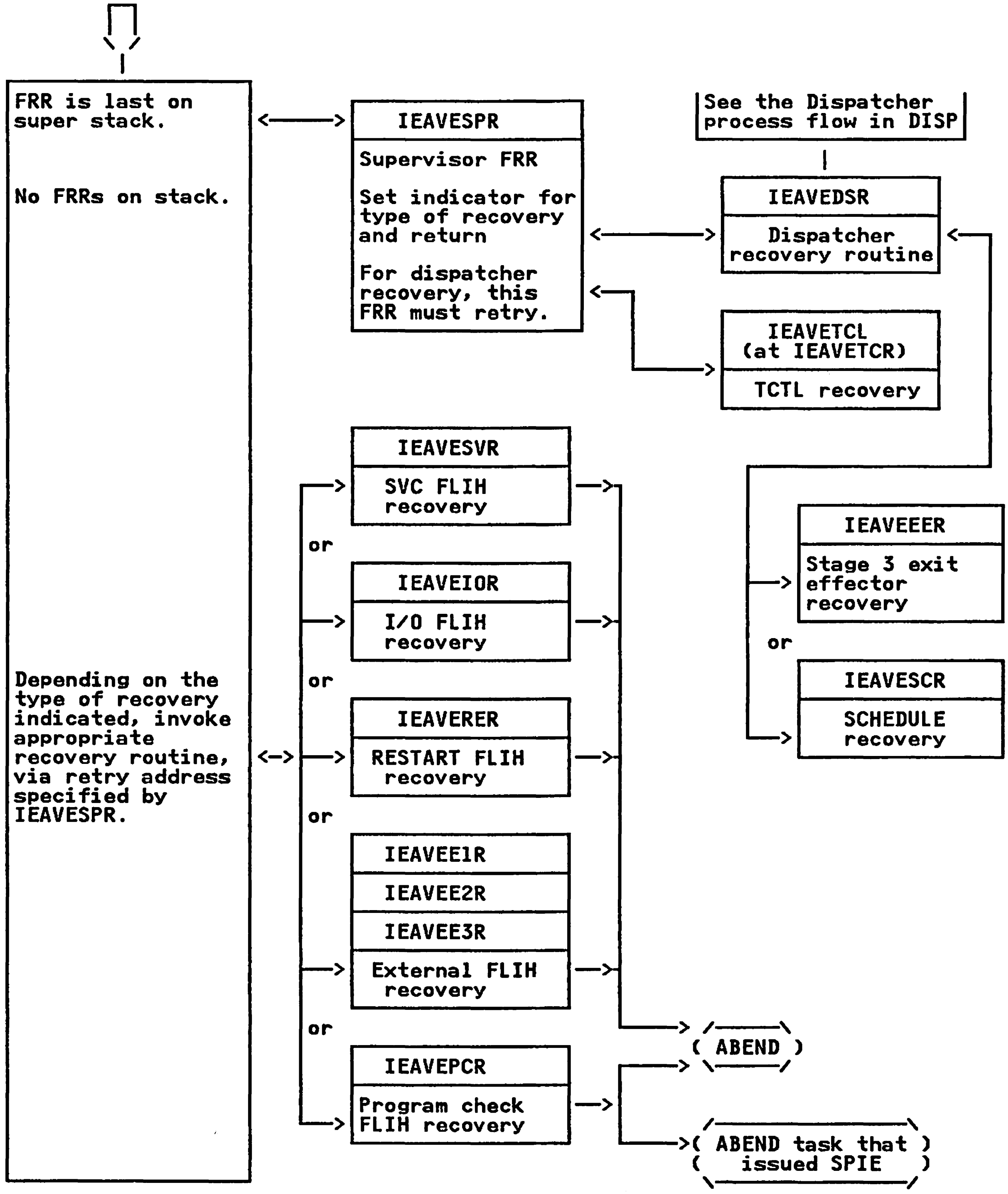
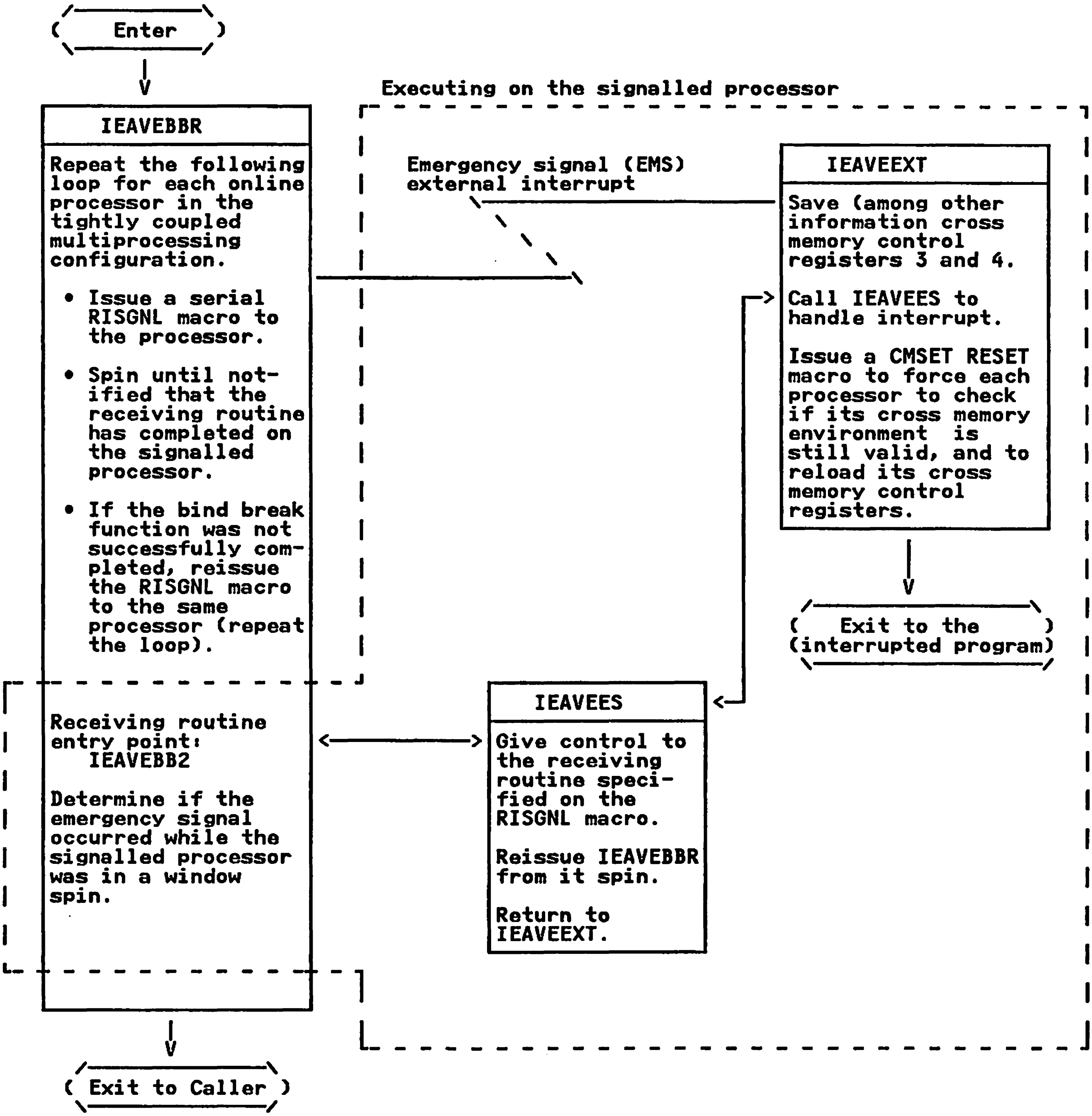

FRR is last on super stack.
No FRRs on stack.
Depending on the type of recovery indicated, invoke appropriate recovery routine, via retry address specified by IEAVESPR.
IEAVESPR
Supervisor FRR
Set indicator for type of recovery and return
For dispatcher recovery, this FRR must retry.
See the Dispatcher process flow in DISP
IEAVEDSR
Dispatcher recovery routine
IEAVETCL (at IEAVETCR)
TCTL recovery
IEAVESVR
SVC FLIH recovery
or
IEAVEIOR
I/O FLIH recovery
or
IEAVERER
RESTART FLIH recovery
or
IEAVEE1R
IEAVEE2R
IEAVEE3R
External FLIH recovery
or
IEAVEPCR
Program check FLIH recovery
IEAVEEER
Stage 3 exit effector recovery
or
IEAVESCR
SCHEDULE recovery
ABEND
ABEND task that issued SPIE

Figure 16 (Part 2 of 2). Supervisor Control Recovery Process Flow


Executing on the signalling processor
Enter
Executing on the signalled processor
Emergency signal (EMS) external interrupt
Exit to the (interrupted program)
Exit to Caller

IEAVEBBR

Repeat the following loop for each online processor in the tightly coupled multiprocessing configuration.

- Issue a serial RISGNL macro to the processor.
- Spin until notified that the receiving routine has completed on the signalled processor.
- If the bind break function was not successfully completed, reissue the RISGNL macro to the same processor (repeat the loop).

Receiving routine entry point: IEAVEBB2

Determine if the emergency signal occurred while the signalled processor was in a window spin.

IEAVEEXT

Save (among other information cross memory control registers 3 and 4.

Call IEAVEES to handle interrupt.

Issue a CMSET RESET macro to force each processor to check if its cross memory environment is still valid, and to reload its cross memory control registers.

IEAVEES

Give control to the receiving routine specified on the RISGNL macro.

Reissue IEAVEBBR from it spin.

Return to IEAVEEXT.

Figure 17. Bind Break Process Flow

METHOD OF OPERATION

This section contains method of operation diagrams. The diagrams use either hipo format or prologue format. The following figure shows the symbols used in hipo format method-of-operation 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.

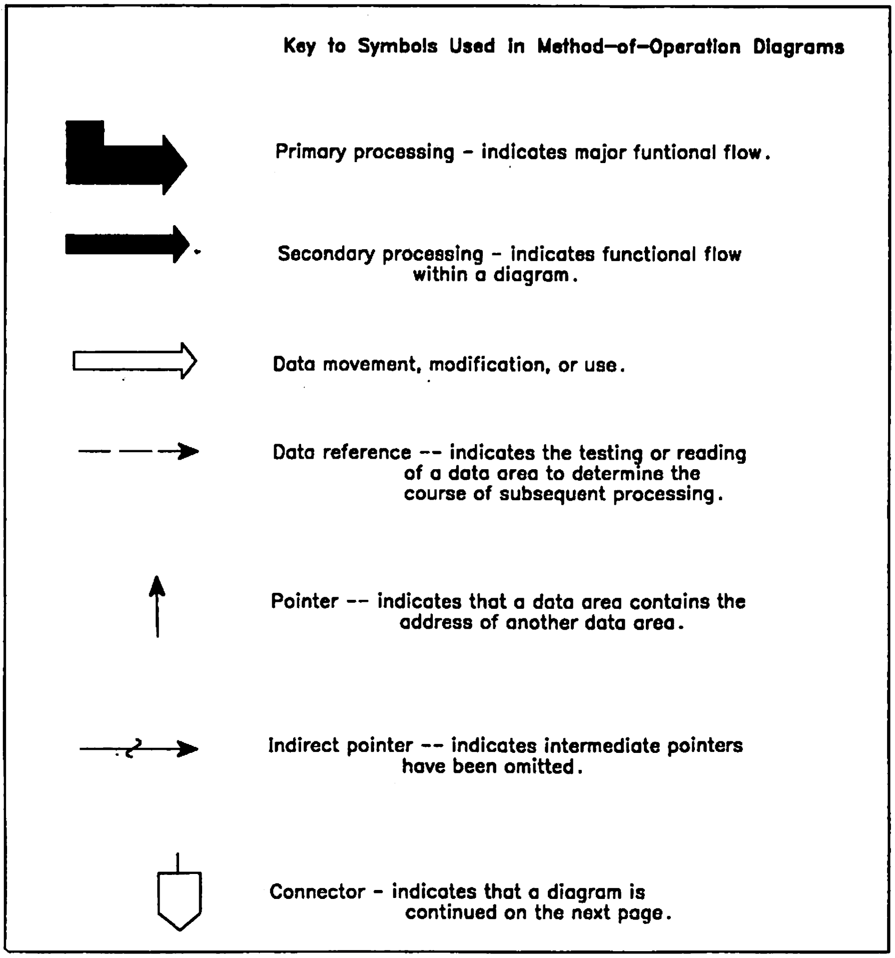

Key to Symbols Used in Method-of-Operation Diagrams
Primary processing - indicates major funtional flow.
Secondary processing - indicates functional flow within a diagram.
Data movement, modification, or use.
Data reference -- indicates the testing or reading of a data area to determine the course of subsequent processing.
Pointer -- indicates that a data area contains the address of another data area.
Indirect pointer -- indicates intermediate pointers have been omitted.
Connector - indicates that a diagram is continued on the next page.

Figure 18. Symbols Used in Method of Operation Diagrams

The prologue 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 that brief module descriptions are also included in the last volume of the System Logic Library (which includes module descriptions for all modules described in the System Logic Library).

Module Operation, which includes:

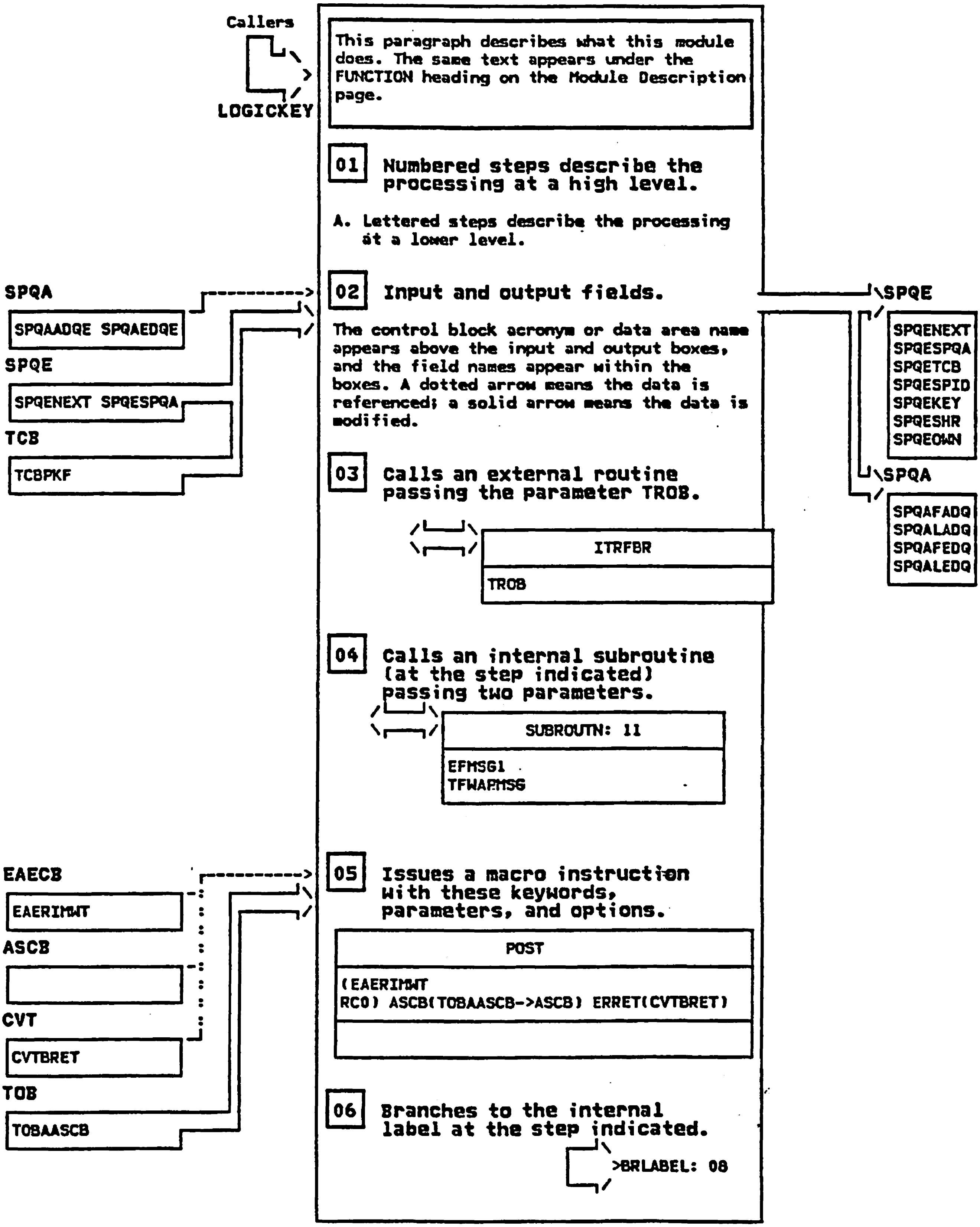
- Operation, which explains how the module performs its function.
- Recovery operation, which explains how the module performs any recovery.

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, retur 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 produces, 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. The following illustrates the graphic symbols and format used in the logic diagrams.

LOGICKEY - Key to the Logic Diagrams STEP 01



This paragraph describes what this module does. The same text appears under the FUNCTION heading on the Module Description page.

01 Numbered steps describe the processing at a high level.

A. Lettered steps describe the processing at a lower level.

02 Input and output fields.

The control block acronym or data area name appears above the input and output boxes, and the field names appear within the boxes. A dotted arrow means the data is referenced; a solid arrow means the data is modified.

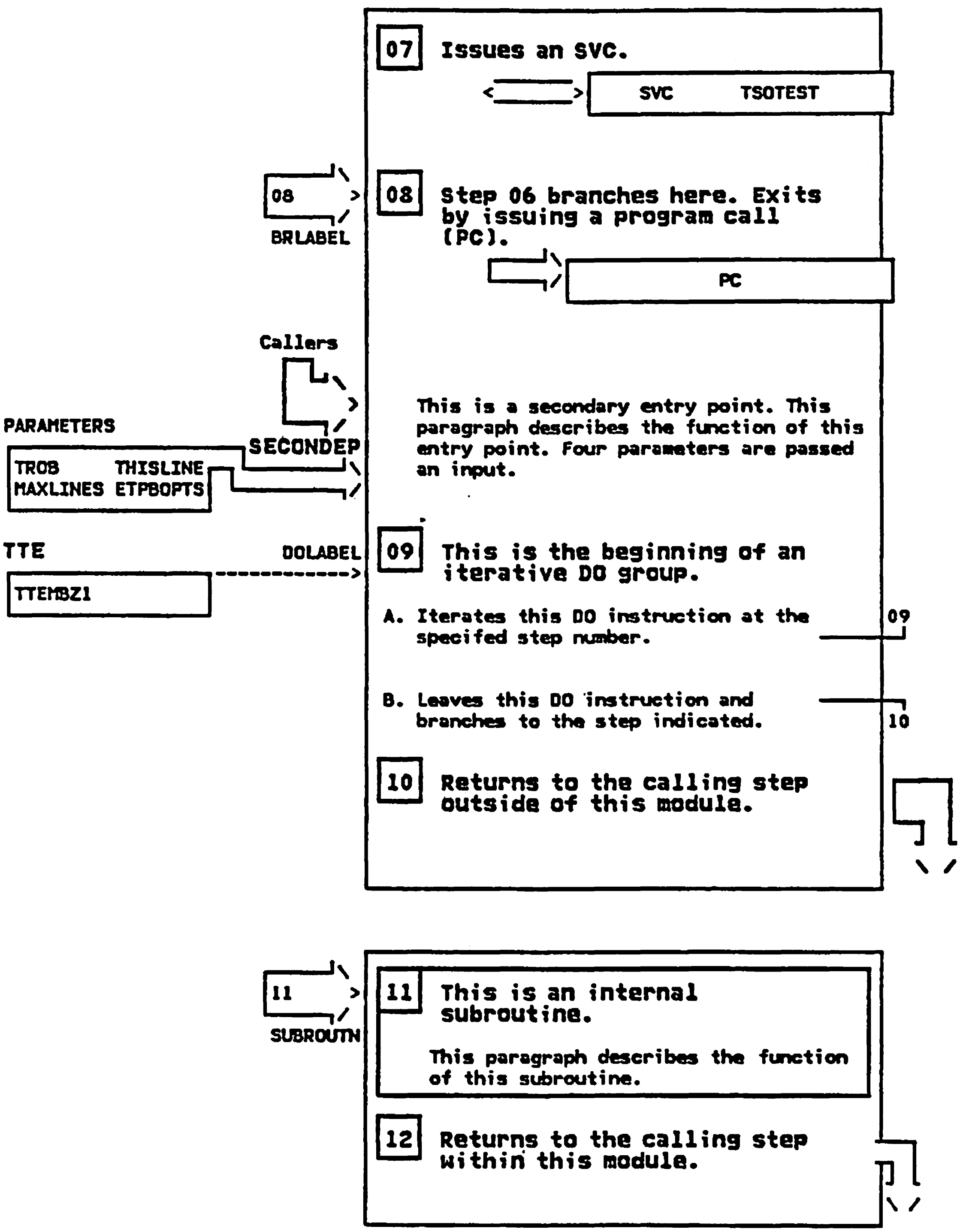
03 Calls an external routine passing the parameter TROB.

04 Calls an internal subroutine (at the step indicated) passing two parameters.

05 Issues a macro instruction with these keywords, parameters, and options.

06 Branches to the internal label at the step indicated.

LOGICKEY - Key to the Logic Diagrams STEP 07



07 Issues an SVC.

08 Step 06 branches here. Exits by issuing a program call (PC).

This is a secondary entry point. This paragraph describes the function of this entry point. Four parameters are passed an input.

09 This is the beginning of an iterative DO group.

A. Iterates this DO instruction at the specifed step number.

B. Leaves this DO instruction and branches to the step indicated.

10 Returns to the calling step outside of this module.

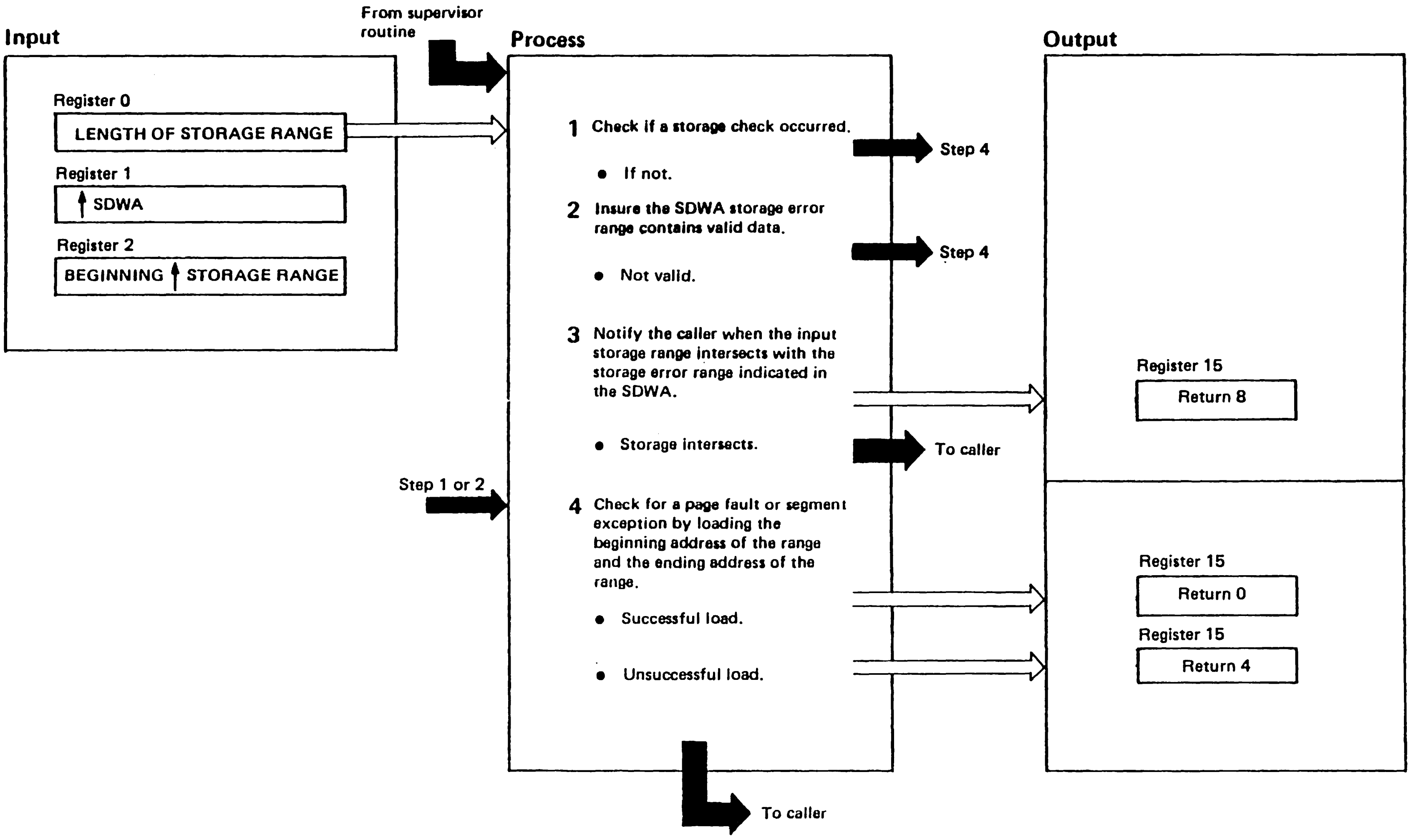
11 This is an internal subroutine.

This paragraph describes the function of this subroutine.

12 Returns to the calling step within this module.

This page left blank

Diagram SUP-1. Address Verification (IEAVEADV) (Part 1 of 2)



Input

Register 0
LENGTH OF STORAGE RANGE

Register 1
↑ SDWA

Register 2
BEGINNING ↑ STORAGE RANGE

From supervisor routine

Process

1 Check if a storage check occurred.

- If not. → Step 4

2 Insure the SDWA storage error range contains valid data.

- Not valid. → Step 4

3 Notify the caller when the input storage range intersects with the storage error range indicated in the SDWA.

- Storage intersects. → To caller

Step 1 or 2

4 Check for a page fault or segment exception by loading the beginning address of the range and the ending address of the range.

- Successful load.
- Unsuccessful load.

To caller

Output

Register 15
Return 8

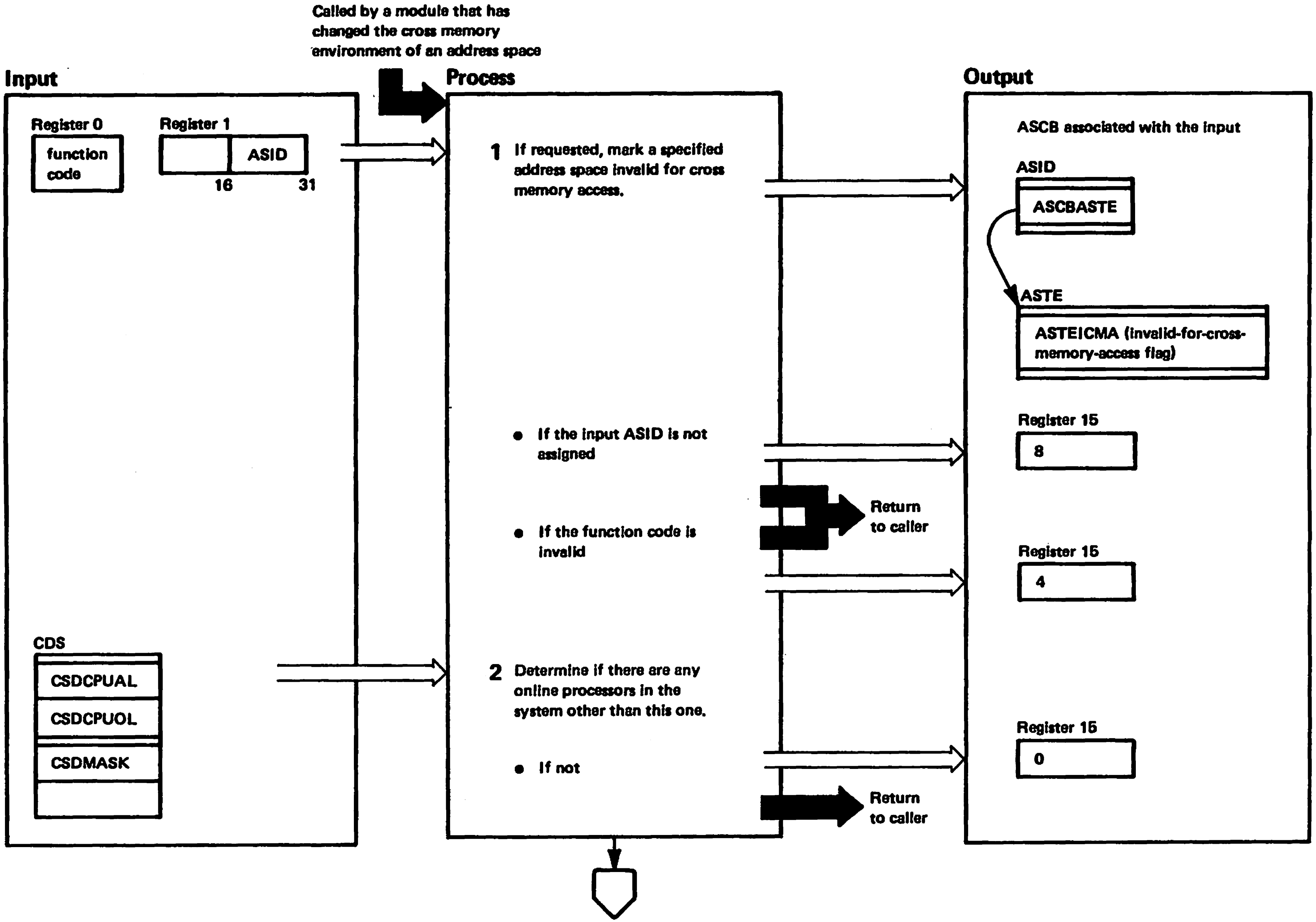
Register 15
Return 0

Register 15
Return 4

Diagram SUP-1. Address Verification (IEAVEADV) (Part 2 of 2)

Extended Description	Module	Label
1 The address verification routine checks the SDWA flags for indication of a storage check error. If a storage check did not occur processing continues at step 4.	IEAVEADV	
2 The error range validity is checked via the SDWA flags. If it is not valid, processing continues at step 4.		
3 A check is made to see if the input storage range intersects with the storage error range indicated in the SDWA. If so, return is to the caller with a code of 8 in register 15.		
4 The final test is to check if the indicated storage is in real storage by doing an LRA on the beginning and ending addresses. If not in storage, a return code of 4 is returned to the caller in register 15. If it is in storage, a return code of 0 is returned to the caller.		

Diagram SUP-2. Bind Break Service Routine (IEAVEBBR) (Part 1 of 6)



1 If requested, mark a specified address space invalid for cross memory access.

- If the input ASID is not assigned
- If the function code is invalid

2 Determine if there are any online processors in the system other than this one.

- If not

Diagram SUP-2. Bind Break Service Routine (IEAVEBBR) (Part 2 of 6)

Extended Description	Module	Label

The bind break module (IEAVEBBR) is called in two situations:

- When a module changes the cross memory environment (the AX, LX, or LT) of an address space, it calls IEAVEBBR to determine if the change made the cross memory environment of any tightly-coupled online processor invalid. If it did, the unit of work on that processor is abended. If a processor's cross memory environment is still valid, IEAVEBBR ensures that the processor's cross memory control registers contain current information.

- When an address space can no longer be accessed using cross memory instructions, or when an address space that can be accessed in cross memory mode is being terminated, the module handling the situation calls IEAVEBBR to set the address space's invalid-for-cross-memory-access (ASTEICMA) flag, and to break all active binds to the address space. If any tightly-coupled online processor's primary or secondary address space is the same as the address space that is no longer accessible, the unit of work on that processor is abended.

At entry, register 0 contains a function code indicating which of the above functions IEAVEBBR is to perform. If the ASTEICMA flag is to be set, register 1 specifies the target ASID.

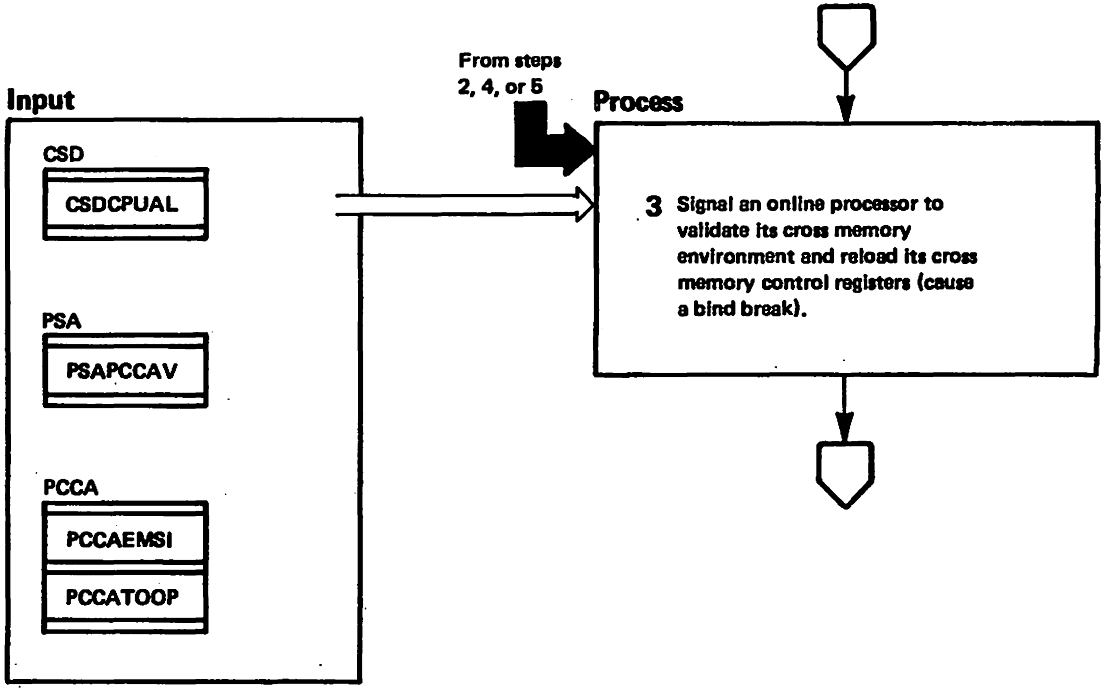
1 If the function code is 4, IEAVEBBR sets to one the ASTEICMA flag associated with the address space identified in register 1. To locate the specified ASCB, IEAVEBBR calls IEAVECMS at entry point IEAVLACB. If IEAVLACB determines that the specified ASID is not assigned, IEAVEBBR sets a return code of 8 and returns to the caller.

If the function code is invalid (neither 0 nor 4) IEAVEBBR sets a return code of 4 and returns to the caller.

Extended Description	Module	Label

2 If the current system configuration includes only one online processor, there are no binds to break and IEAVEBBR's work is finished. IEAVEBBR sets a return code of zero and returns to the caller.

Diagram SUP-2. Bind Break Service Routine (IEAVEBBR) (Part 3 of 6)



3 Signal an online processor to validate its cross memory environment and reload its cross memory control registers (cause a bind break).

Diagram SUP-2. Bind Break Service Routine (IEAVEBBR) (Part 4 of 6)

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

3 .IEAVEBBR ensures that every tightly-coupled online processor checks whether its cross memory environment is still valid after the change, and, if necessary, updates its cross memory control registers to reflect the change. To do this, IEAVEBBR issues a RISGNL SERIAL macro to each online processor except the one on which it is executing. The macro triggers the following sequence of events. See the Bind Break Module Flow figure for a picture of these events.

- An external interrupt occurs on the signalled processor.
- IEAVEBBR enters a spin loop on the signalling processor, and loops until the interrupt handler clears the PCCAEMSI word indicating that the receiving routine has completed on the signalled processor.
- The external interrupt FLIH (IEAVEEXT) gets control on the signalled processor and saves the processor's control registers 3 and 4 (the primary and secondary ASIDs and the AX value) in the PSA.
- Because this is an emergency signal (EMS) interrupt, IEAVEEXT gives control to the emergency signal SLIH (IEAVEES), which gives control to the RISGNL receiving routine (IEAVEBB2 in IEAVEBBR).
- IEAVEBB2 determines whether the EMS interrupt occurred while the signalled processor was in a window spin, and sets the LCCABBRC field in the signalling processor's LCCA accordingly. IEAVEBBR uses this field later to determine whether the bind break function was completed successfully. (Binds cannot be broken when the signalled processor is in a window spin.)
- When the signalled processor is not in a window spin, IEAVEBB2 also issues a precautionary CMSET SET macro, which sets the primary and secondary ASID to the home ASID. The CMSET SET macro is necessary because IEAVEES informs the signalling processor to resume executing before it returns control to IEAVEEXT, which then performs the bind break function. The CMSET macro ensures that the signalling processor's cross memory environment is valid between now and the time IEAVEEXT actually performs the bind break operation.

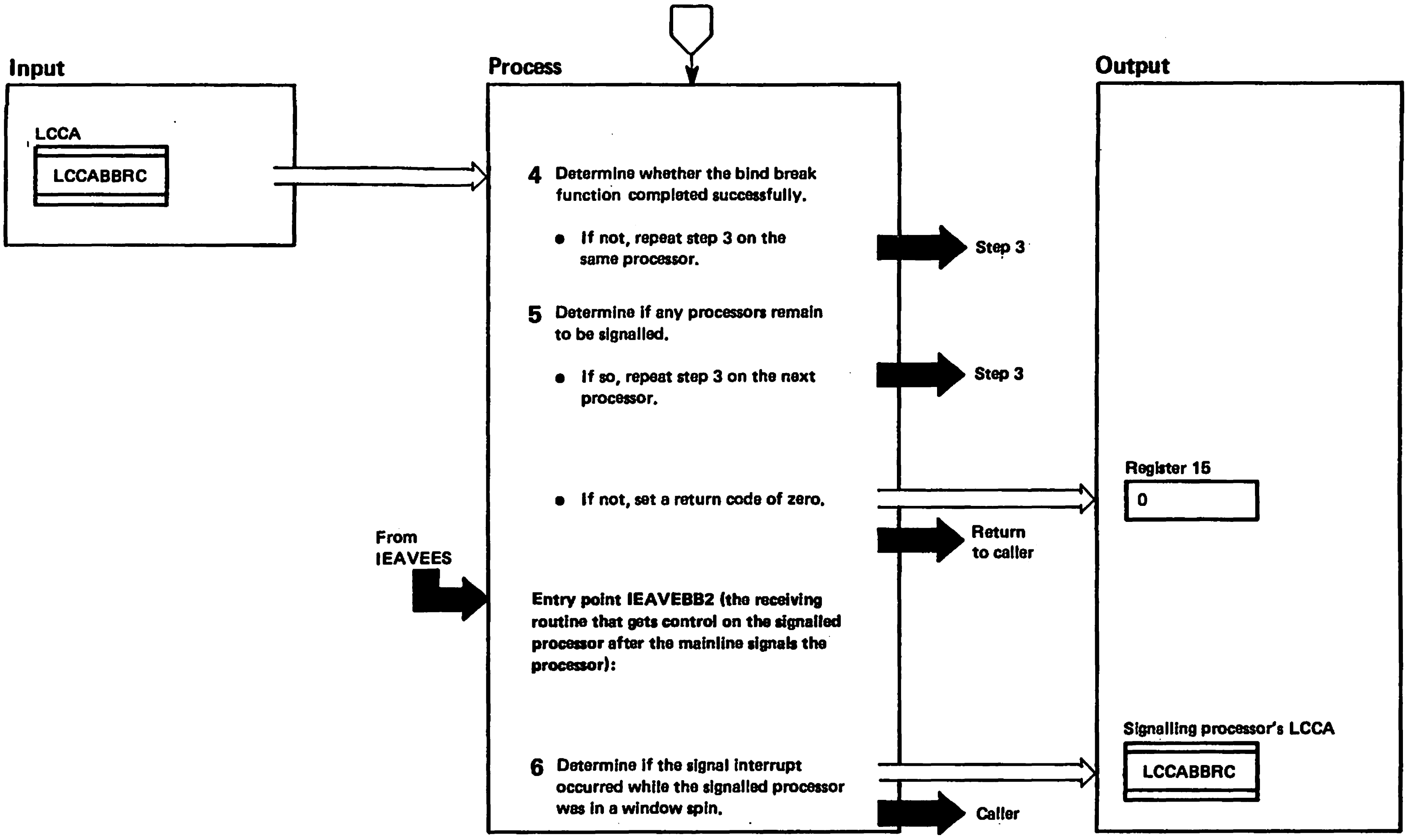
Extended Description **Module** **Label**

3 (continued)

- IEAVEBB2 returns to IEAVEES, which releases the signalling processor from its spin by clearing the PCCAEMSI word in the signalling processor's PCCA. IEAVEES then returns to IEAVEEXT. Thus, IEAVEBBR resumes executing on the signalling processor while IEAVEEXT completes the bind break function on the signalled processor. The next step (step 4) describes what IEAVEBBR does when it resumes executing. The next bullets describe how IEAVEEXT completes the bind break function.
- If the signalled processor was in a window spin when the interrupt occurred, IEAVEEXT returns to the interrupted program. It cannot complete the bind break function.
- If the signalled processor was not in a window spin when the interrupt occurred, IEAVEEXT issues a CMSET RESET,CHKAUTH=(YES) macro. The CMSET service routine (IEAVECMS) first checks whether the signalled processor's primary and secondary ASIDs can still be accessed using cross memory instructions, and whether they can access each other. If they can, IEAVECMS reloads the signalled processor's cross memory control registers. This ensures that, if applicable, the signalled processor picks up the cross memory change that triggered the call to IEAVEBBR. Note that the CMSET RESET is done on all eligible online processors, even though the processor might not have an active bind to the address space whose cross memory environment changed (neither its primary nor secondary address space is the same as the changed address space). In this instance, the CMSET RESET macro resets, but does not change, the control register values.

  If the processor's cross memory environment is not valid after the change, IEAVECMS terminates the unit of work in progress by issuing ABEND X'058' with the appropriate reason code.
- IEAVEEXT returns to the interrupted unit of work.

Diagram SUP-2. Bind Break Service Routine (IEAVEBBR) (Part 5 of 6)



4 Determine whether the bind break function completed successfully.

- If not, repeat step 3 on the same processor.

5 Determine if any processors remain to be signalled.

- If so, repeat step 3 on the next processor.
- If not, set a return code of zero.

Entry point IEAVEBB2 (the receiving routine that gets control on the signalled processor after the mainline signals the processor):

6 Determine if the signal interrupt occurred while the signalled processor was in a window spin.

Diagram SUP-2. Bind Break Service Routine (IEAVEBBR) (Part 6 of 6)

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

4 IEAVEBBR checks the LCCABBRC field to determine whether the interrupt occurred while the signalled processor was in a window spin. If it did not, IEAVEBBR assumes that the bind break function completed successfully and continues processing at the next step.

If the signalled processor was in a window spin, IEAVEBBR adds one to the number of unsuccessful signal attempts and reissues the RISGNL macro to the same processor. When the number of attempts made exceeds a specified limit, IEAVEBBR calls the excessive spin notification routine (IEEVEXSN) to issue message IEE331A. IEAVEBBR then resets the spin count and continues signalling the processor.

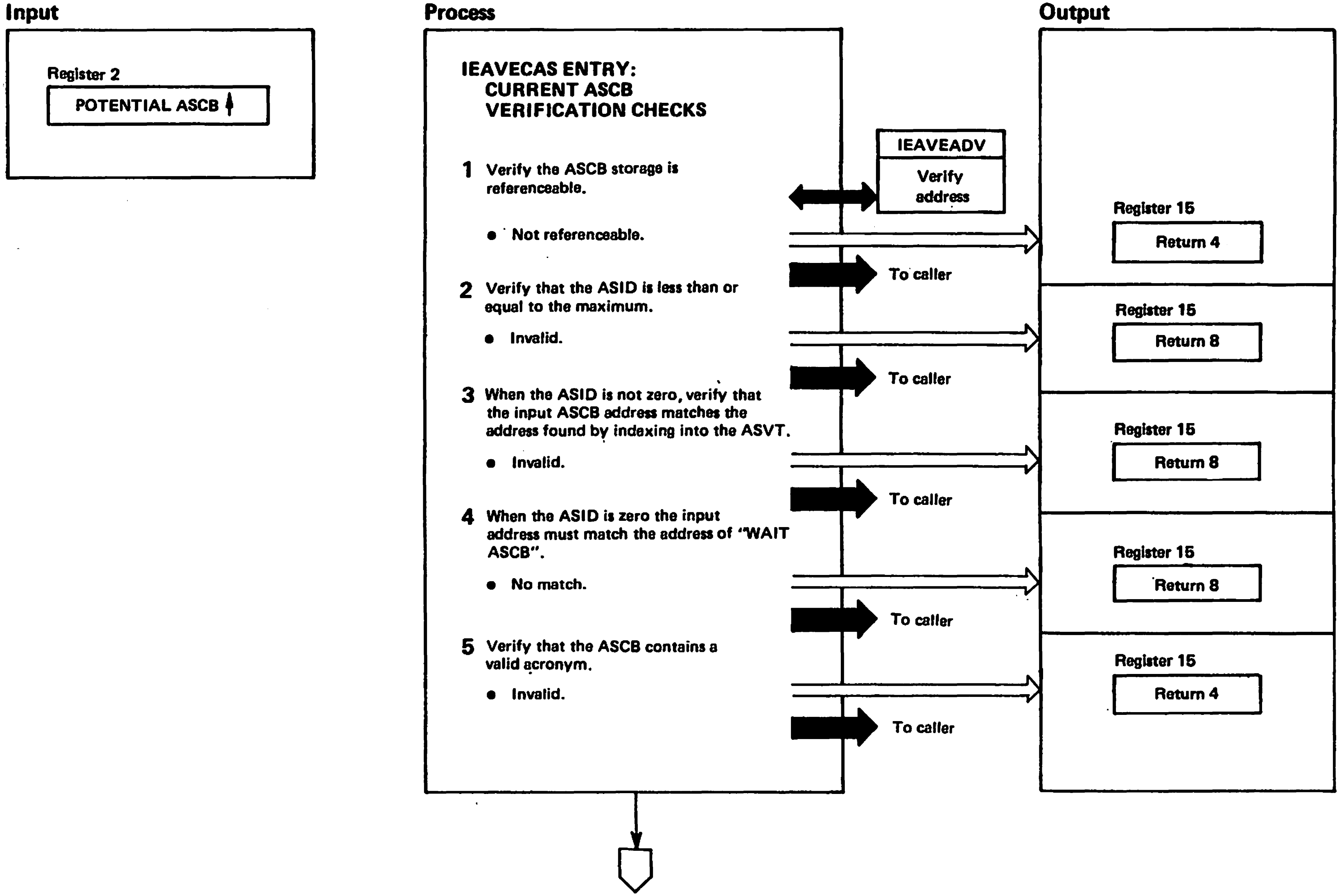
5 IEAVEBBR looks for another online processor to signal. If there is one, IEAVEBBR repeats step 3 for that processor. After all the online processors have been signalled, IEAVEBBR sets a return code of zero and returns to the caller.

6 IEAVEBB2's processing is described in step 3.

Recovery Processing:

IEAVEBBR does not establish its own recovery environment. If an error occurs while IEAVEBBR is executing, the caller's recovery routine receives control.

Diagram SUP-3. Control Block Verification Routine (IEAVECBV) (Part 1 of 10)



Input

Register 2

POTENTIAL ASCB

Process

IEAVECAS ENTRY:
CURRENT ASCB
VERIFICATION CHECKS

1 Verify the ASCB storage is referenceable.

- Not referenceable.

2 Verify that the ASID is less than or equal to the maximum.

- Invalid.

3 When the ASID is not zero, verify that the input ASCB address matches the address found by indexing into the ASVT.

- Invalid.

4 When the ASID is zero the input address must match the address of "WAIT ASCB".

- No match.

5 Verify that the ASCB contains a valid acronym.

- Invalid.

IEAVEADV

Verify address

To caller

To caller

To caller

To caller

To caller

Output

Register 15

Return 4

Register 15

Return 8

Register 15

Return 8

Register 15

Return 8

Register 15

Return 4

Diagram SUP-3. Control Block Verification Routine (IEAVECBV) (Part 2 of 10)

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

This module will determine whether an input address is the address of a valid 1) current ASCB, 2) general ASCB, 3) SRB, 4) TCB, or 5) STCB.

1-5 **For current ASCB verification (IEAVECAS), the input address must pass the following criteria:**

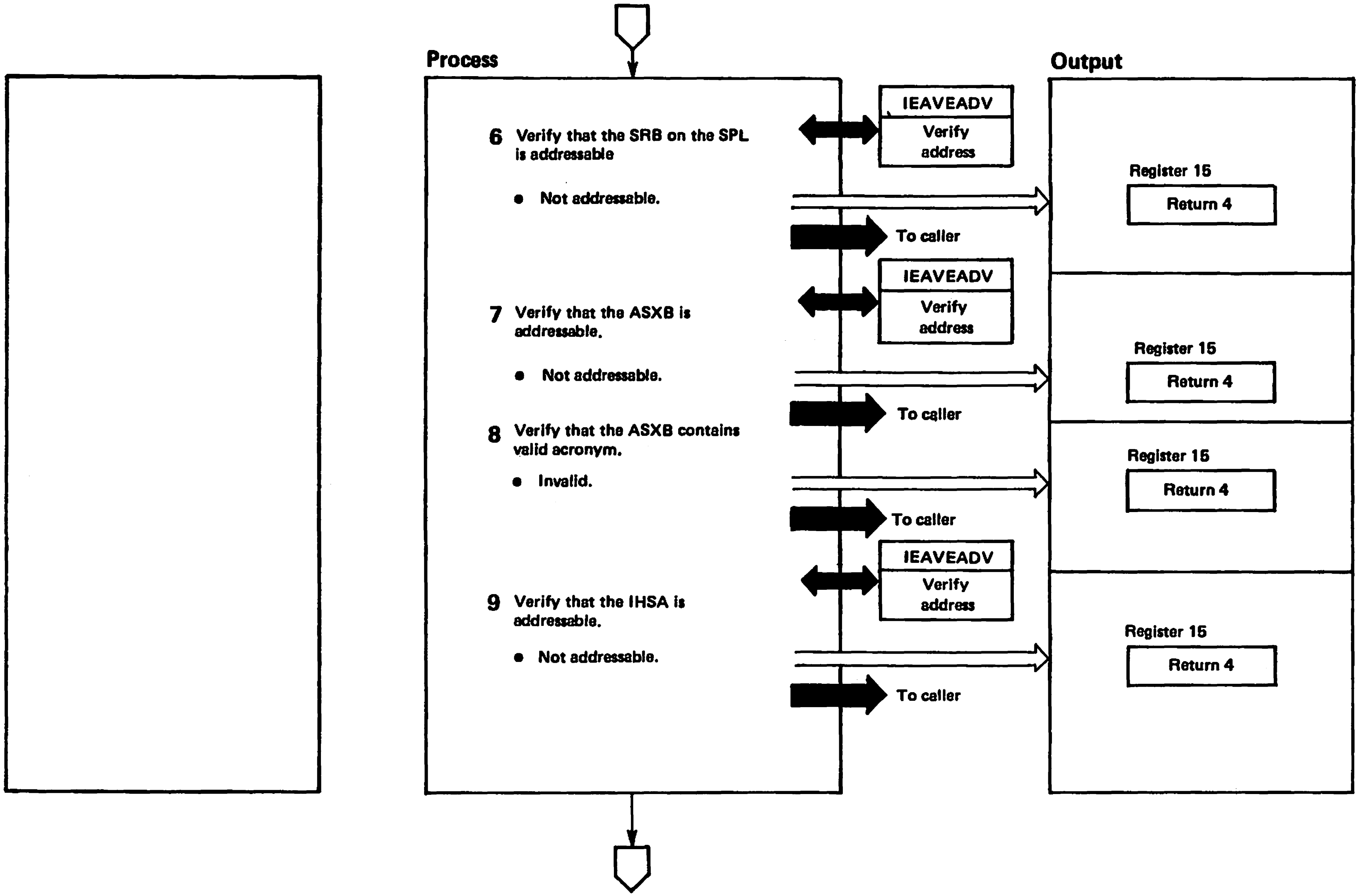
- **Addressable potential ASCB storage.**
- **ASID ≤ maximum.**
- **When the ASID ≠ 0, the input address matches the address found by indexing into the ASVT.**
- **When the ASID = 0, the input address must match the address of 'WAIT ASCB".**
- **Valid acronym (ASCB).**
- **Addressable and valid SRB address on SPL.**
- **Addressable ASXB.**
- **ASXB must have a valid acronym, an addressable IHSA, and an addressable local work/save area vector table.**

A return code of 0 indicates a valid control block.

A return code of 4 indicates a control block contains bad information.

A return code of 8 indicates not a control block.

Diagram SUP-3. Control Block Verification Routine (IEAVECBV) (Part 3 of 10)



6 Verify that the SRB on the SPL is addressable

- Not addressable.

7 Verify that the ASXB is addressable.

- Not addressable.

8 Verify that the ASXB contains valid acronym.

- Invalid.

9 Verify that the IHSA is addressable.

- Not addressable.

Diagram SUP-3. Control Block Verification Routine (IEAVECBV) (Part 4 of 10)

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

6-9 For current ASCB verification (IEAVECAS), the input address must pass the following criteria:

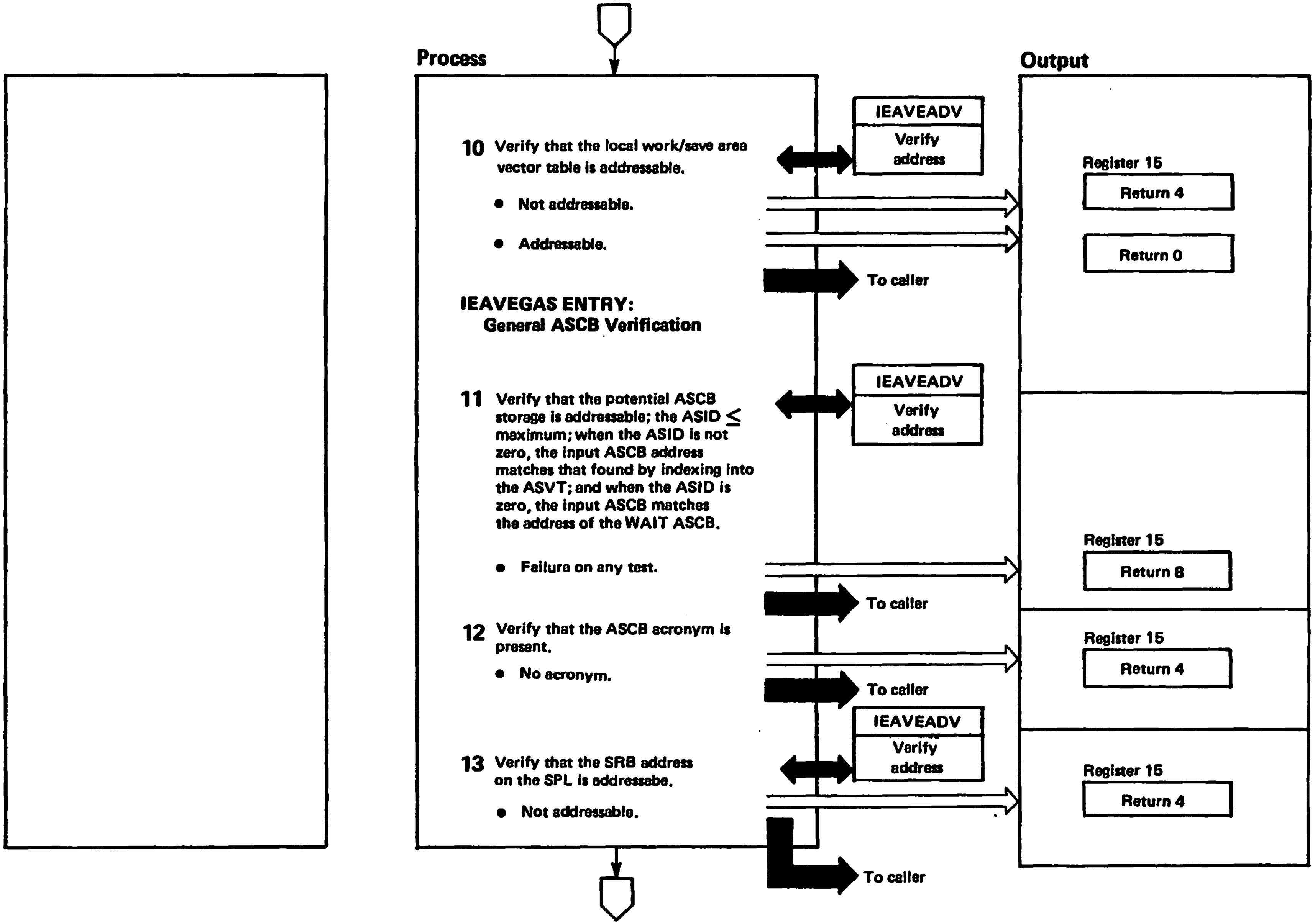
- Addressable potential ASCB storage.
- ASID ≤ maximum.
- When the ASID ≠ 0, the input address matches the address found by indexing into the ASVT.
- When the ASID = 0, the input address must match the address of "WAIT ASCB".
- Valid acronym (ASCB).
- Addressable and valid SRB address on SPL.
- Addressable ASXB.
- ASXB must have a valid acronym, an addressable IHSA, and an addressable local work/save area vector table.

A return code of 0 indicates a valid control block.

A return code of 4 indicates a control block contains bad information.

A return code of 8 indicates not a control block.

Diagram SUP-3. Control Block Verification Routine (IEAVECBV) (Part 5 of 10)



Process

10 Verify that the local work/save area vector table is addressable.

IEAVEADV
Verify address

- Not addressable.
- Addressable.

To caller

IEAVEGAS ENTRY:
General ASCB Verification

11 Verify that the potential ASCB storage is addressable; the ASID ≤ maximum; when the ASID is not zero, the input ASCB address matches that found by indexing into the ASVT; and when the ASID is zero, the input ASCB matches the address of the WAIT ASCB.

IEAVEADV
Verify address

- Failure on any test.

To caller

12 Verify that the ASCB acronym is present.

- No acronym.

To caller

13 Verify that the SRB address on the SPL is addressabe.

IEAVEADV
Verify address

- Not addressable.

To caller

Output

Register 15
Return 4

Return 0

Register 15
Return 8

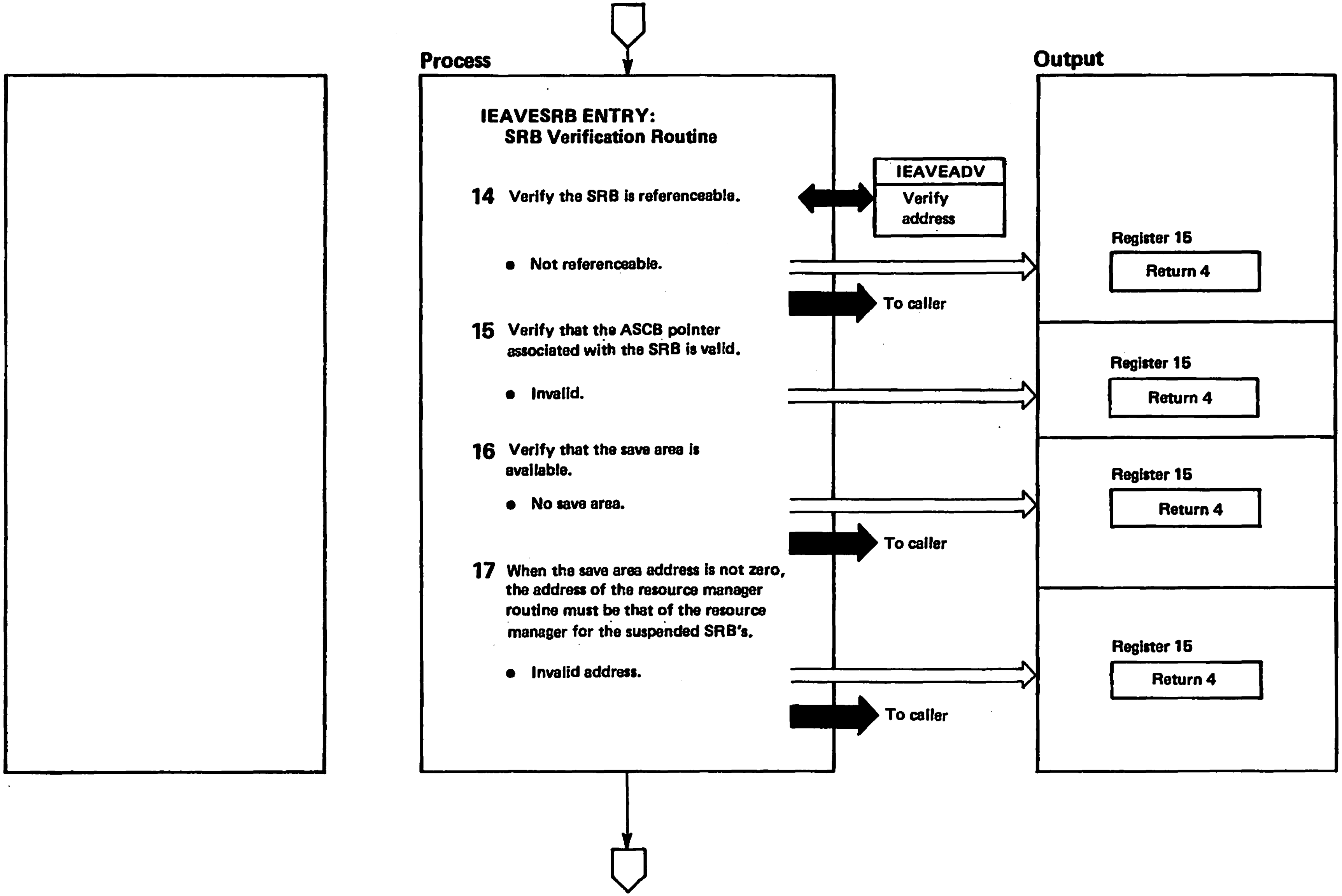
Register 15
Return 4

Register 15
Return 4

Diagram SUP-3. Control Block Verification Routine (IEAVECBV) (Part 6 of 10)

Extended Description	Module	Label
10 ASXB must have a referenceable local work/save area vector table. A return code of 0 indicates a valid control block. A return code of 4 indicates a control block contains bad information.		
11-13 For general ASCB verification (IEAVEGAS), the input address must pass the first six criteria listed under the current ASCB verification. The return codes indicate the same conditions.		

Diagram SUP-3. Control Block Verification Routine (IEAVECBV) (Part 7 of 10)



Process

IEAVESRB ENTRY:
SRB Verification Routine

14 Verify the SRB is referenceable.

- Not referenceable.

15 Verify that the ASCB pointer associated with the SRB is valid.

- Invalid.

16 Verify that the save area is available.

- No save area.

17 When the save area address is not zero, the address of the resource manager routine must be that of the resource manager for the suspended SRB's.

- Invalid address.

Output

Register 15
Return 4

Register 15
Return 4

Register 15
Return 4

Register 15
Return 4

Diagram SUP-3. Control Block Verification Routine (IEAVECBV) (Part 8 of 10)

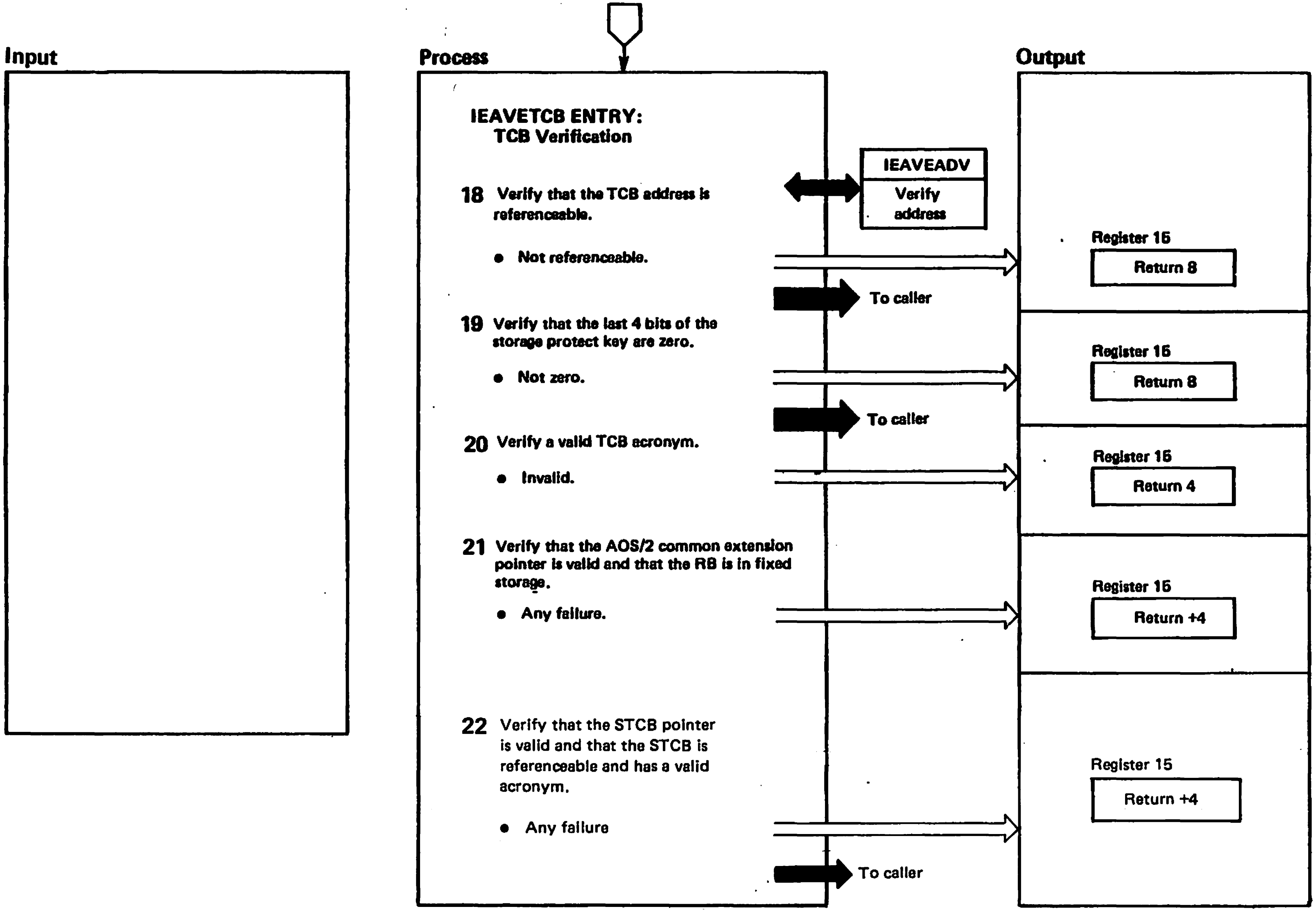
Extended Description **Module** **Label**

14-17 For SRB verification, the following criteria must be met:

- Referenceable SRB storage.
- Valid ASCB pointer.
- Valid save area data.
- When the save area address ≠ 0, the address of the resource manager routine must be that of the resource manager for suspended SRB's.
- When the save address = 0, the routine entry point address must be non-zero.

Return codes indicate the same conditions as indicated under current ASCB verification.

Diagram SUP-3. Control Block Verification Routine (IEAVECBV) (Part 9 of 10)



Input

Process

IEAVETCB ENTRY:
TCB Verification

18 Verify that the TCB address is referenceable.

- Not referenceable.

19 Verify that the last 4 bits of the storage protect key are zero.

- Not zero.

20 Verify a valid TCB acronym.

- Invalid.

21 Verify that the AOS/2 common extension pointer is valid and that the RB is in fixed storage.

- Any failure.

22 Verify that the STCB pointer is valid and that the STCB is referenceable and has a valid acronym.

- Any failure

Diagram SUP-3. Control Block Verification Routine (IEAVECBV) (Part 10 of 10)

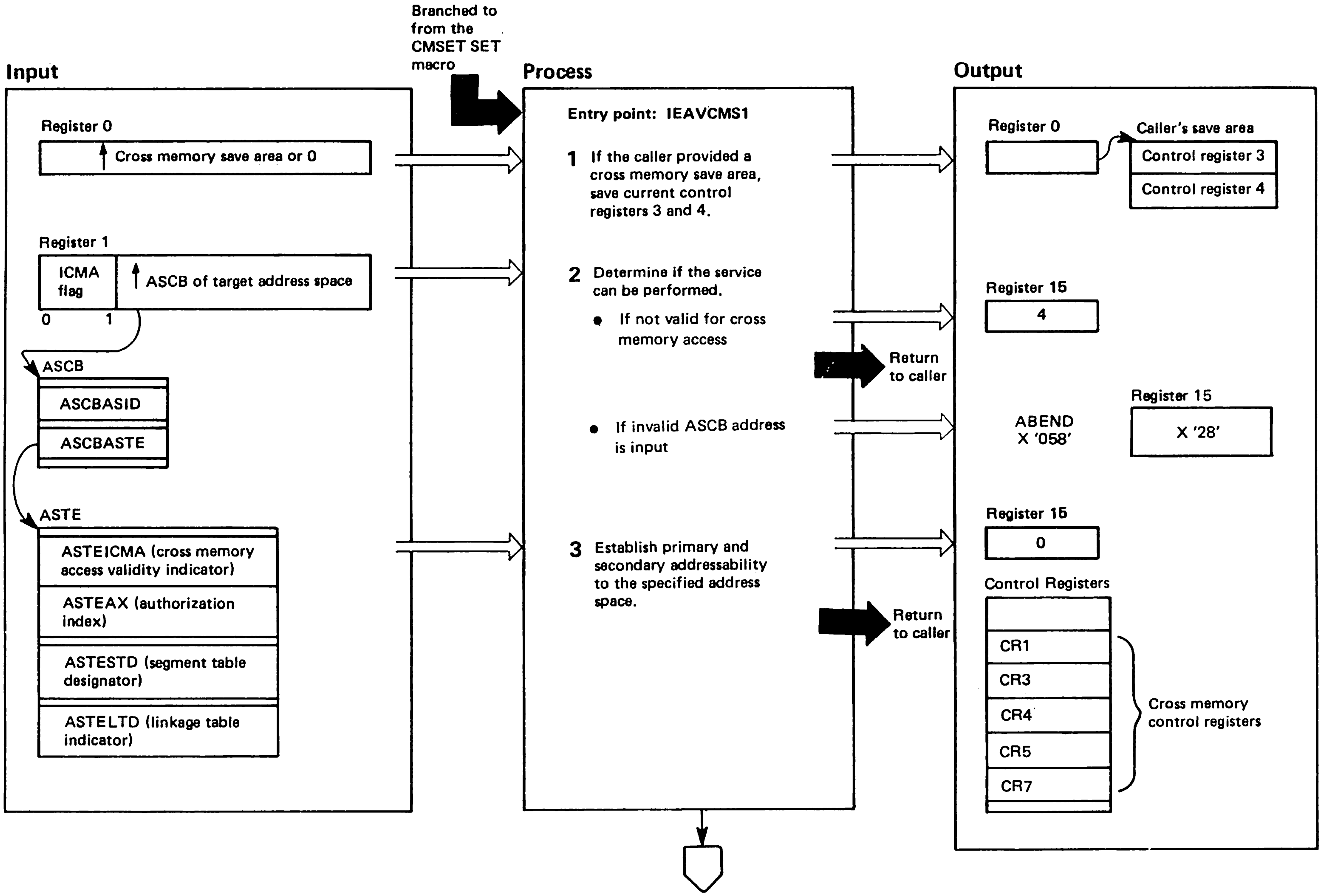
Extended Description **Module** **Label**

18-22 For TCB verification, the following criteria must be met:

- Referenceable potential TCB storage.
- Last 4 bits of the storage protect key must be zero.
- Valid acronym.
- Valid AOS/2 common extension pointer.
- Current RB in fixed storage.
- Valid STCB must exit.

Return codes are the same as for the current ASCB verification routine.

Diagram SUP-4. CMSET, LOCASCB, and CALLDISP Macro Service Routine (IEAVECMS) (Part 1 of 8)



Branched to from the CMSET SET macro
Input
Register 0
Cross memory save area or 0
Register 1
ICMA flag
ASCB of target address space
0
1
ASCB
ASCBASID
ASCBASTE
ASTE
ASTEICMA (cross memory access validity indicator)
ASTEAX (authorization index)
ASTESTD (segment table designator)
ASTELTD (linkage table indicator)
Process
Return to caller
Return to caller
Output
Register 0
Caller's save area
Control register 3
Control register 4
Register 15
4
ABEND X '058'
Register 15
X '28'
Register 15
0
Control Registers
CR1
CR3
CR4
CR5
CR7
Cross memory control registers

Entry point: IEAVCMS1

1 If the caller provided a cross memory save area, save current control registers 3 and 4.

2 Determine if the service can be performed.

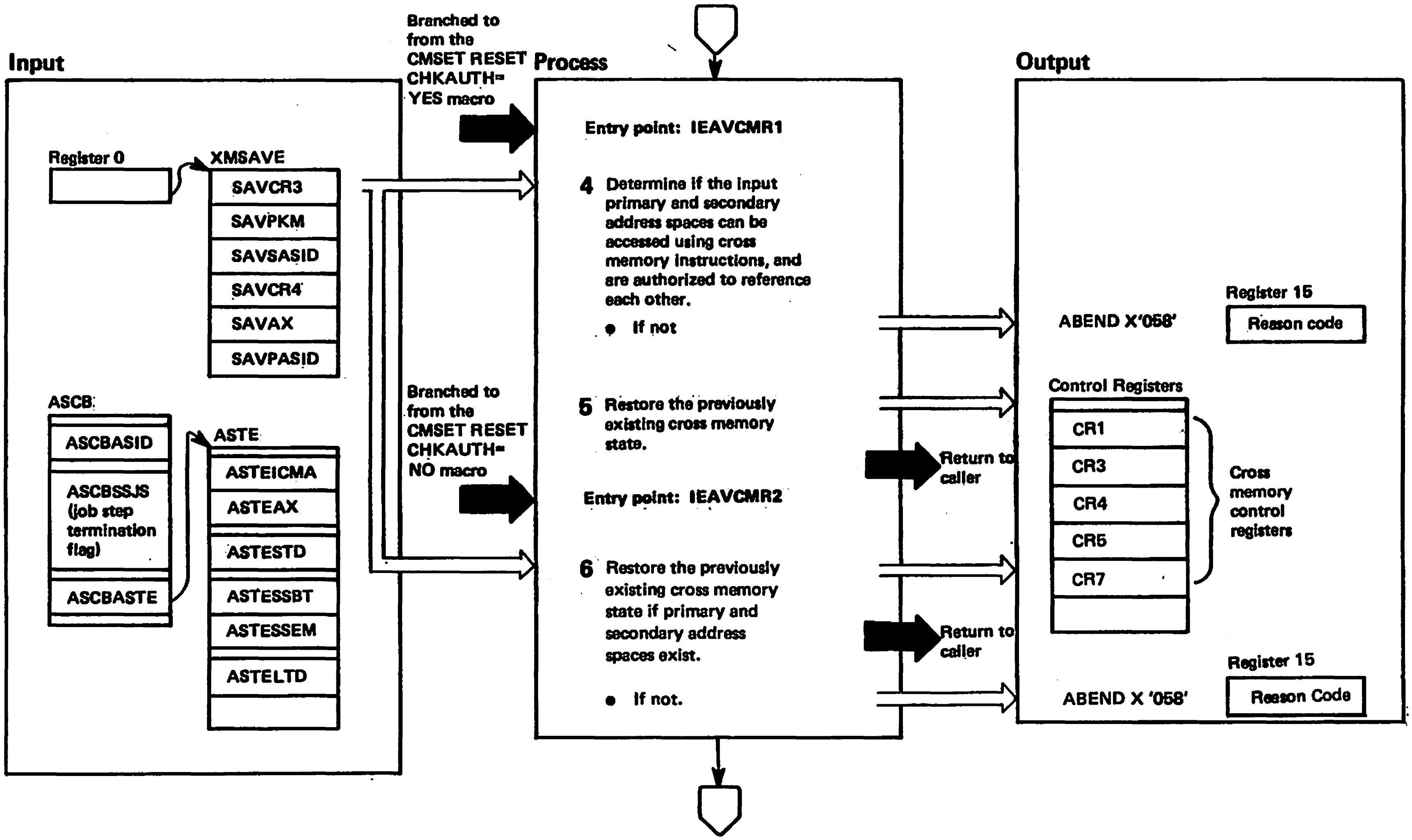
- If not valid for cross memory access
- If invalid ASCB address is input

3 Establish primary and secondary addressability to the specified address space.

Diagram SUP-4. CMSET, LOCASCB, and CALLDISP Macro Service Routine (IEAVECMS) (Part 2 of 8)

Extended Description	Module	Label
When any of the following macros is issued, IEAVECMS receives control to perform the requested service.		
• CMSET SET • CMSET RESET with authorization checking • CMSET RESET without authorization checking • CMSET SSARTO • CMSET SSARBACK • LOCASCB • CALLDISP (specifying the BRANCH=YES option).		
Each macro has its own entry point and is described separately in this diagram.		
CMSET SET processing:		
The CMSET SET service routine establishes an input ASCB as both the caller's primary and secondary address space. The caller specifies whether IEAVECMS is to perform the service unconditionally or only when the input ASCB can be accessed in cross memory mode.		
1 If the caller provided a cross memory save area, IEAVECMS saves current cross memory control registers 3 and 4 in the caller's save area.	IEAVECMS	IEAVCMS1
2 If the caller requested that IEAVECMS perform the service only if the input ASCB can be accessed in cross memory mode, IEAVECMS checks the invalid-for-cross-memory-access indicator (the ASTEICMA field) in the input ASCB's ASTE. If it is set to 1, IEAVECMS sets a return code of 4 and returns to the caller. If the input ASCB address is not a valid ASCB, IEAVECMS issues an ABEND with an X'058' completion code and an X'28' reason code in register 15.		
3 When the request can be performed, IEAVECMS establishes primary and secondary addressability to the specified ASCB by changing the processor's cross memory control registers. To do this, IEAVECMS:		
• Loads the ASID of the specified address space into control register 3. This establishes it as the secondary ASID. In the process, the PSW key-mask contained in the high order bits of control register 3 is set to zero.		
• Loads the ASID of the specified address space and the caller's authorization index (AX) into control register 4. This sets the new PASID and AX.		
• Loads the caller's linkage table designator (LTD) into control register 5, primary segment table designator (STD) into control register 1, and secondary STD into control register 7.		
IEAVECMS returns to the caller with a return code of 0. This indicates that the specified address space is now the PASID and SASID.		

Diagram SUP-4. CMSET, LOCASCB, and CALLDISP Macro Sarvice Reoutine (IEAVECMS) (Part 3 of 8)



Input

Register 0

XMSAVE

SAVCR3

SAVPKM

SAVSASID

SAVCR4

SAVAX

SAVPASID

ASCB

ASCBASID

ASCBSSJS (job step termination flag)

ASCBASTE

ASTE

ASTEICMA

ASTEAX

ASTESTD

ASTESSBT

ASTESSEM

ASTELTD

Branched to from the CMSET RESET CHKAUTH= YES macro

Branched to from the CMSET RESET CHKAUTH= NO macro

Process

Entry point: IEAVCMR1

4 Determine if the input primary and secondary address spaces can be accessed using cross memory instructions, and are authorized to reference each other.

- If not

5 Restore the previously existing cross memory state.

Return to caller

Entry point: IEAVCMR2

6 Restore the previously existing cross memory state if primary and secondary address spaces exist.

Return to caller

- If not.

Output

ABEND X'058'

Register 15

Reason code

Control Registers

CR1

CR3

CR4

CR5

CR7

Cross memory control registers

ABEND X '058'

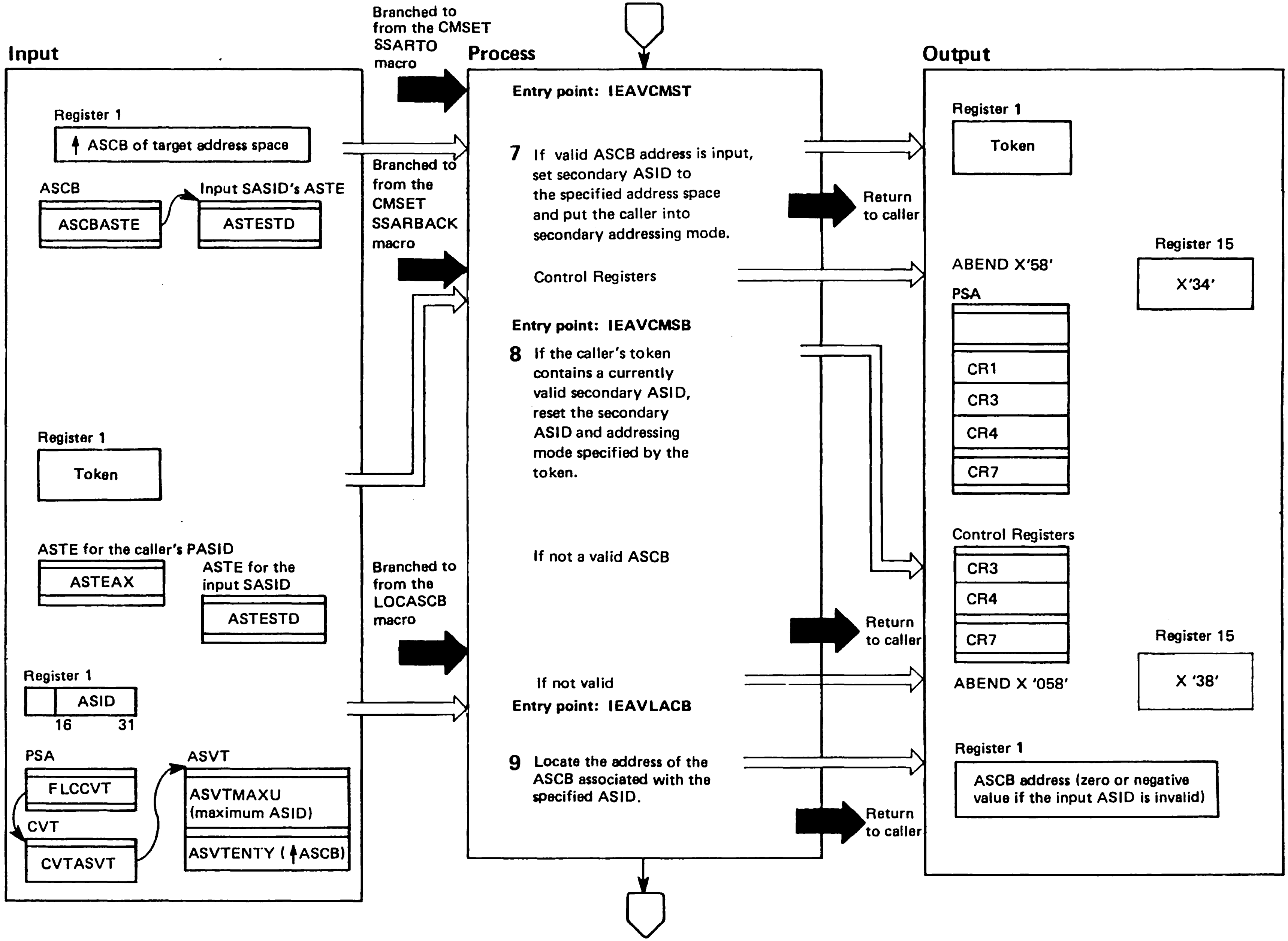
Register 15

Reason Code

Diagram SUP-4. CMSET, LOCASCB, and CALLDISP Macro Service Routine (IEAVECMS) (Part 4 of 8)

Extended Description	Module	Label
CMSET RESET processing: The CMSET RESET service routine restores a previously existing cross memory environment. At entry, register 0 points to a save area containing the cross memory status to be restored.		
4 When the caller specified the CHKAUTH=YES option, IEAVECMS determines if the input primary and secondary address spaces meet the following conditions: • Both address spaces can currently be accessed in cross memory mode • The job step task in neither address space has terminated • Both ASIDs are assigned • The address spaces are authorized to reference each other If any of these conditions is not satisfied, IEAVECMS issues ABEND X'058' with a unique reason code. (See *System Messages* and *System Codes* for the specific reason codes.)		IEAVCMR1
5 When the processor's cross memory environment can be restored, IEAVECMS loads: • The primary segment table designator (STD) into control register 1 • The secondary ASID into control register 3 • The primary ASID and authorization index into control register 4 • The linkage table designator into control register 5 • The secondary STD into control register 7		
6 When the CMSET RESET macro is issued and no authorization check is requested, IEAVECMS restores the previously existing cross memory environment as described in step 5 if the input primary and secondary address spaces are currently assigned. If the address spaces do not exist, IEAVECMS issues ABEND X'058' with a unique reason code. (See *System Messages* and *System Codes* for the specific reason codes.)		IEAVCMR2

Diagram SUP-4. CMSET, LOCASCB, and CALLDISP Macro Service Routine (IEAVECMS) (Part 5 of 8)



Entry point: IEAVCMST

7 If valid ASCB address is input, set secondary ASID to the specified address space and put the caller into secondary addressing mode.

Control Registers

Entry point: IEAVCMSB

8 If the caller's token contains a currently valid secondary ASID, reset the secondary ASID and addressing mode specified by the token.

If not a valid ASCB

If not valid

Entry point: IEAVLACB

9 Locate the address of the ASCB associated with the specified ASID.

Diagram SUP-4. CMSET, LOCASCB, and CALLDISP Macro Service Routine (IEAVECMS) (Part 6 of 8)

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

CMSET SSARTO processing:

The CMSET SSARTO service routine unconditionally establishes the input address space as the secondary address space and puts the caller into secondary addressing mode. As a result, the caller executes in secondary mode and accesses data in the secondary address space. IEAVCMST

7 To establish secondary addressability to the input address space, IEAVECMS:

- Checks the validity of the input ASCB address. If the address is not valid, IEAVECMS issues an ABEND with an X '058' completion code and an X '34' reason code in register 15.
- Saves in register 1 a token containing the current SASID and PSW S-bit. IEAVECMS returns this token to the caller. The caller uses it when issuing a CMSET SSARBACK macro, which restores the cross memory environment to its state prior to executing this macro.
- Loads the new SASID into control register 3.
- Puts an authorization index (AX) of 1 into control register 4, which authorizes the caller to access any address space in secondary mode.
- Loads the new SASID's segment table designator (STD) into control register 7.
- Sets the S-bit inthe PSW, which puts the processor in secondary addressing mode.
- When cross memory hardware is being simulated, loads control register 1 with the SASID's STD.
- Returns to the caller.

CMSET SSARBACK processing:

The CMSET SSARBACK service routine restores the secondary ASID and cross memory addressing mode that existed before the CMSET SSARTO macro was executed. IEAVECMS receives as input the token it returned to the caller after processing the CMSET SSARTO macro. The token contains the SASID and PSW S-bit values that are to be restored. IEAVCMSB

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

8 To restore the cross memory environment, IEAVECMS:

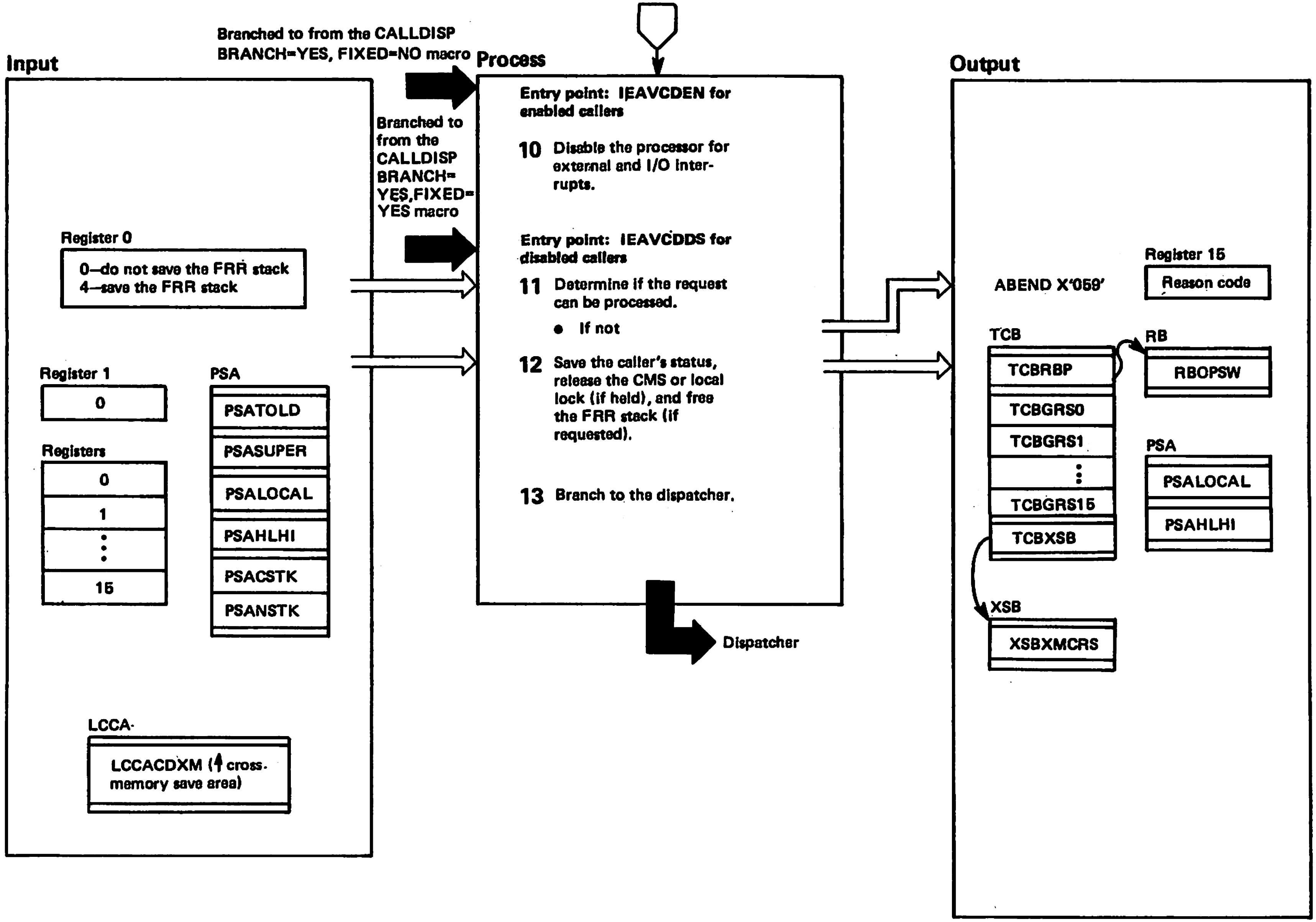
- Verifies that the input SASID contained in the token is currently assigned. If the input SASID is not currently , assigned, IEAVECMS issues an X'38' reason code in register 15.
- Puts the input SASID into control register 3
- Puts the PASID's authorization index (AX) into control register 4
- Puts the SASID's STD into control register 7
- Restores the previous addressing mode by adjusting the PSW S-bit
- Returns to the caller

LOCASCB processing:

The LOCASCB macro service routine locates and returns the address of the ASCB associated with the ASID specified in register 1.

9 If the input ASID is assigned, IEAVECMS returns the associated ASCB's address in register 1. If the ASID is not assigned, IEAVECMS returns a zero or negative value in register 1. IEAVLACB

Diagram SUP-4. CMSET, LOCASCB, and CALLDISP Macro Service Routine (IEAVECMS) (Part 7 of 8)



Input

Register 0

0—do not save the FRR stack
4—save the FRR stack

Register 1

0

Registers

0
1
⋮
15

PSA

PSATOLD
PSASUPER
PSALOCAL
PSAHLHI
PSACSTK
PSANSTK

LCCA

LCCACDXM (↑ cross-memory save area)

Branched to from the CALLDISP BRANCH=YES, FIXED=NO macro

Branched to from the CALLDISP BRANCH=YES,FIXED=YES macro

Process

Entry point: IEAVCDEN for enabled callers

10 Disable the processor for external and I/O interrupts.

Entry point: IEAVCDDS for disabled callers

11 Determine if the request can be processed.

- If not

12 Save the caller's status, release the CMS or local lock (if held), and free the FRR stack (if requested).

13 Branch to the dispatcher.

Dispatcher

Output

ABEND X'059'

Register 15

Reason code

TCB

TCBRBP
TCBGRS0
TCBGRS1
⋮
TCBGRS15
TCBXSB

RB

RBOPSW

PSA

PSALOCAL
PSAHLHI

XSB

XSBXMCRS

Diagram SUP-4. CMSET, LOCASCB, and CALLDISP Macro Service Routine (IEAVECMS) (Part 8 of 8)

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

CALLDISP processing:

The CALLDISP service routine provides a means for key 0, supervisor state, task mode routines to give up control and enter the dispatcher.

10 **If the caller is enabled, IEAVECMS disables the processor for external and I/O interrupts and saves register 14-1 in the PSA.**

IEAVCDEN

11 **To determine if the request can be processed, IEAVECMS checks the following conditions in the order specified. If a condition is not met, IEAVECMS issues ABEND X'06D' with a unique reason code. (See *System Messages* and *System Codes* for the specific reason codes.)**

IEAVCDDS

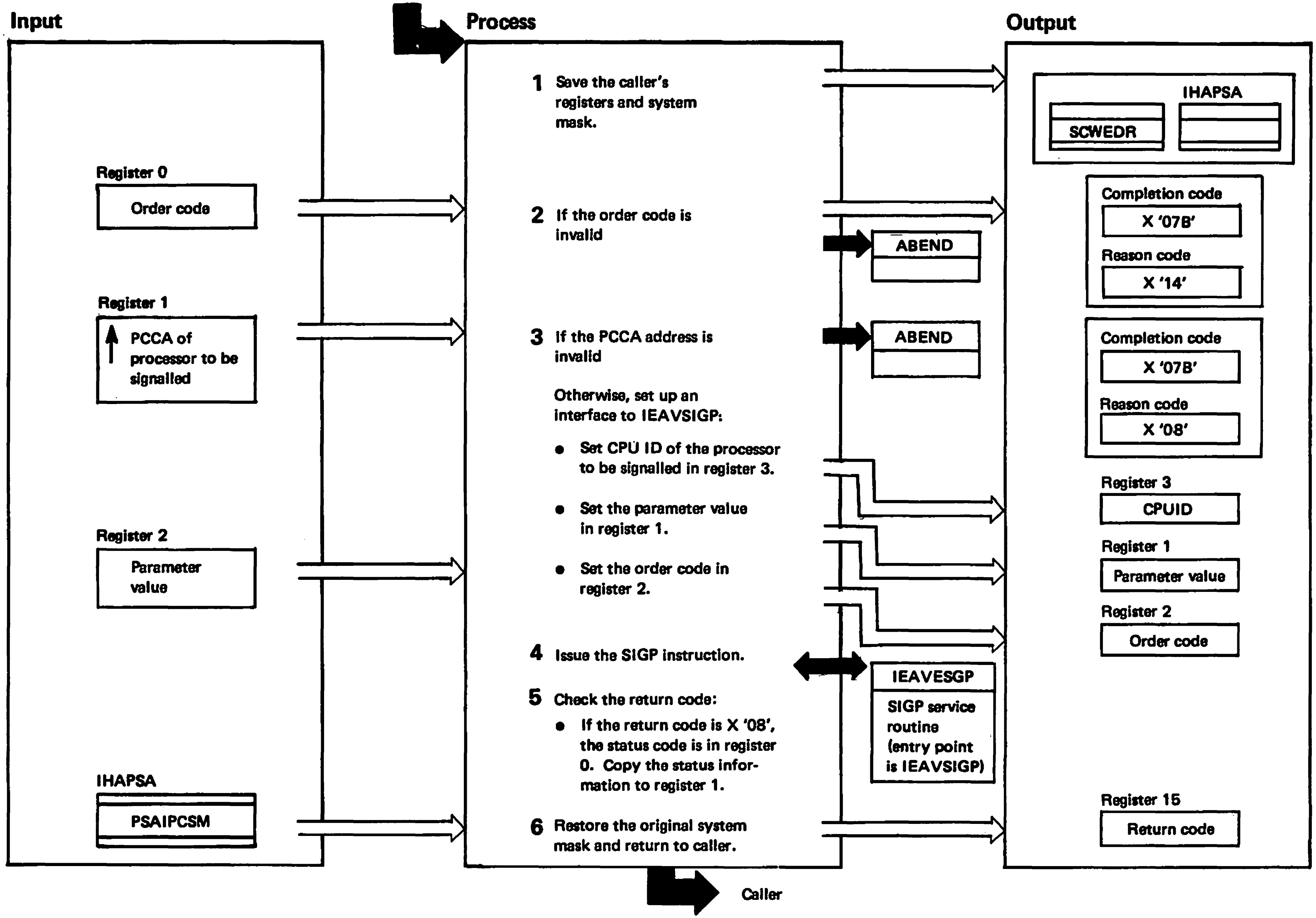
- The caller is in TCB mode
- Register 1 contains a zero
- No super bits are set (PSASUPER=0)
- Register 0 contains a 0 or a 4
- The normal FRR stack is the current one
- If the FRR stack is not being saved, RTM1 is not active
- If the FRR stack is not being saved, only the CML or LOCAL lock is held. Note that no locks are required.
- If the FRR is being saved and the LOCAL or CML lock is held then an EUTFRR must exist
- If the FRR stack is being saved, no locks can be held except as stated above.

12 **In preparation for branching to the dispatcher, IEAVECMS:**

- **Saves the caller's registers in the TCB.**
- **Saves the caller's cross memory registers in the TCB's XSB.**
- **Saves the caller's resume PSW in the RB.**
- **If the caller holds the LOCAL or CML lock, frees it.**
- **If the caller requests it, frees the FRR stack. Otherwise, the dispatcher later decides whether to free the FRR stack.**

13 **IEAVECMS branches to the dispatcher at entry point IEAODS1.**

Diagram SUP-5. Interprocessor Communication (IPC) Direct Signal Routine (IEAVEDR) (Part 1 of 2)



Input

Register 0

Order code

Register 1

PCCA of processor to be signalled

Register 2

Parameter value

IHAPSA

PSAIPCSM

Process

1 Save the caller's registers and system mask.

2 If the order code is invalid

3 If the PCCA address is invalid

Otherwise, set up an interface to IEAVSIGP:

- Set CPU ID of the processor to be signalled in register 3.
- Set the parameter value in register 1.
- Set the order code in register 2.

4 Issue the SIGP instruction.

5 Check the return code:

- If the return code is X '08', the status code is in register 0. Copy the status information to register 1.

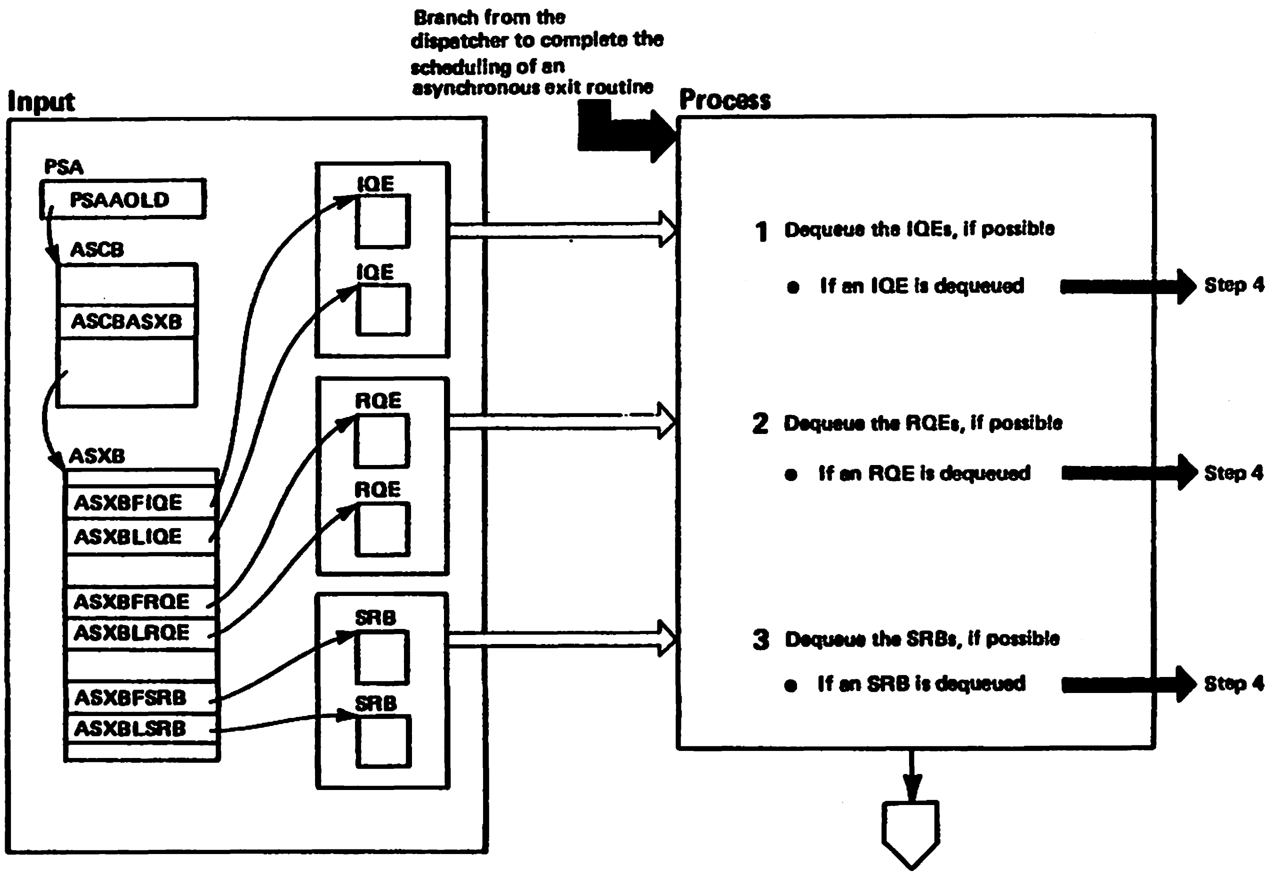
6 Restore the original system mask and return to caller.

Output

Diagram SUP-5. Interprocessor Communication (IPC) Direct Signal Routine (IEAVEDR) (Part 2 of 2)

Extended Description	Module	Label
IEAVEDR provides the user with the necessary interfaces and facilities to signal another online processor via the SIGP hardware instruction. The SIGP instruction allows the user to invoke a specific hardware function on the signalled processor.		
1 Saves the caller's registers in the SCWA. Save the system mask in PSAIPCSM.	IEAVEDR	
2 Validates the order code. If the order code is not valid, issues an X'07B' ABEND and a reason code of X '14'.	IEAVEDR	
3 Validates the PCCA address. If the PCCA address is invalid, issues an X'07B', ABEND and a reason code of X '08'. If the PCCA address is valid, extracts the physical processor ID to be signalled from the PCCA in anticipation of calling the SIGP service routine (entry point IEAVSIGP in module IEAVESGP).	IEAVESGP	IEAVSIGP
4 Calls IEAVESGP to issue the SIGP instruction.		
5 Checks the return code from IEAVESGP. If the return code is X '08', status has been returned to register 0 and IEAVEDR copies the status information into register 1.	IEAVEDR	
6 Returns to the caller after restoring the original system mask and registers.		

Diagram SUP-6. Stage 3 Exit Effector (IEAVEEE0) (Part 1 of 6)



1 Dequeue the IQEs, if possible

- If an IQE is dequeued

2 Dequeue the RQEs, if possible

- If an RQE is dequeued

3 Dequeue the SRBs, if possible

- If an SRB is dequeued

Diagram SUP-6. Stage 3 Exit Effector (IEAVEEE0) (Part 2 of 6)

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

The stage 3 exit effector (the last routine used to schedule an asynchronous exit), dequeues IQEs (interruption queue elements), RQEs (request queue elements) or SRBs (service request blocks) from asynchronous exit queues pointed to by the ASCB. For each element removed, it initializes and chains to the TCB an IRB/XSB or SIRB/XSB pair. The dispatcher enters the stage 3 exit effector as a subroutine.

1 Supervisor services use IQEs as a general interface for requesting scheduling of an asynchronous routine. — Module: IEAVEEE0
For each IQE on the asynchronous exit queue, the stage 3 exit effector does the following:

- Determines if the IQE can be dequeued at this time. An IQE cannot be dequeued if:
  a. The IQE has been purged by DUMP (IQEPURGE=1).
  b. The IRB (interruption request block) is already being used (RBFACTV=1).
  c. The task that the asynchronous exit is to process is executing on another processor or holds a lock.
  d. The asynchronous exit is being scheduled to the error task and an error recovery procedure is executing on that task.
  e. Asynchronous exits have been suppressed for the intended task (TCBFX=1).
  f. The IQE is for an attention exit and either all asynchronous exits or attention exits are suppressed (TCBFX=1 or, TCBATT=1) for the intended task or one of the task's descendants in the task tree.
  g. The resume or transfer control function is executing for the TCB that the stage 3 exit effector is checking (TCBS3A=1 and TCBACTIV=1). The stage 3 exit effector turns on the flags.
  h. The TCB has any enabled, unlocked task (EUT) mode FRRs (TCBNSSP≠0).
  i. The vector facility environment is being established for the task, and an SRB has been scheduled to complete the environment (STCBPIQ=1).
- If the IQE can be dequeued and the IRB is for an attention exit, the stage 3 exit effector (IEAVEEE0) turns off the TCBTIOTG flag (which exit prolog sets to ensure that TGET/TPUT SVRBs are purged). If the TCBS3MR flag is set (making the task non-dispatchable until the attention exit executes), IEAVEEE0 turns off the flag and adds one to the ready TCBs count (ASCBTCBs).
- For each IQE that can be dequeued, IEAVEEE0 removes it from the queue and chains the specified IRB to the TCB, as described in step 4.

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

2 Data management uses RQEs as a special interface in scheduling an asynchronous exit.

- For each RQE on the asynchronous exit queue, a series of tests are made to determine if it can be dequeued at this time. An RQE cannot be dequeued if:
  a. Asynchronous exits are suppressed for the task (TCBFX=1).
  b. The task it is being scheduled to is active an another processor or holds a lock.
  c. The IRB is already in use (RBFACTV=1).
  d. The asynchronous exit is being scheduled to the address space's error task and an error recovery procedure is already executing on the error task.
  e. The resume or transfer control function is executing for the TCB that the stage 3 exit effector is checking (TCBS3A=1 and TCBACTIV=1). The stage 3 exit effector turns on the flags.
  f. The TCB has any EUT mode FRRs (TCBNSSP≠0).
  g. The vector facility environment is being established for the task, and an SRB has been scheduled to cpmp complete the environment (STCBPIQ=1).
- For those RQEs that can be dequeued, IEAVEEE0 removes the RQE from the queue and chains the specified IRB to the TCB, as described in step 4.

3 SRBs on the queue represent requests by IOS to schedule non-resident error recovery procedures.
There is a single system IRB per address space that runs only under the error task of the address space. The stage 3 exit effector tries to schedule this SIRB for only the top SRB on the queue.

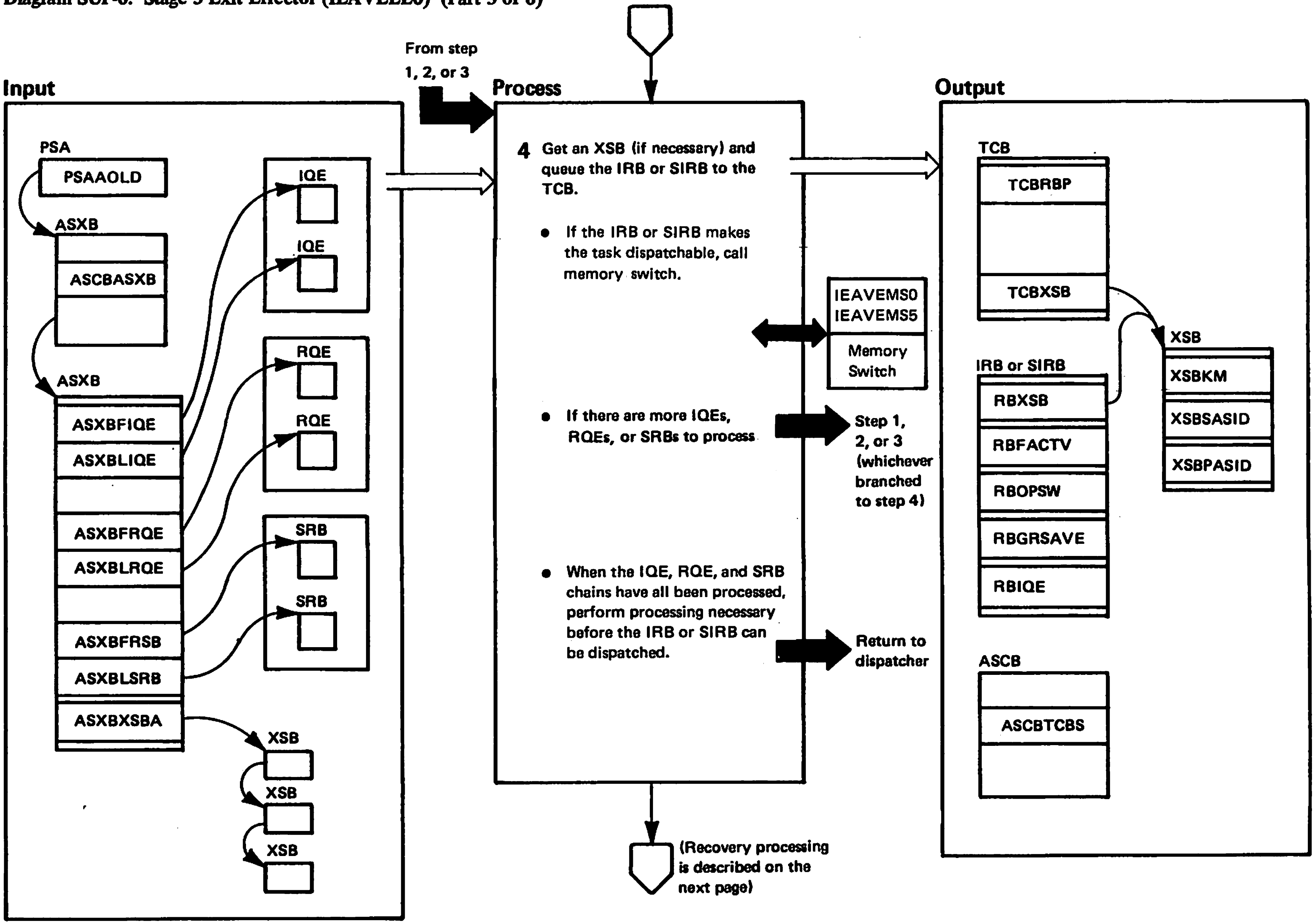
The SIRB cannot be scheduled if:

- The error task is already executing on another processor or holds a lock.
- An error recovery procedure is already executing in the address space.
- The resume or transfer control function is executing for the error task (TCBS3A=1 and TCBACTIV=1). The stage 3 exit effector turns on the flags.
- The error task has any EUT mode FRRs (TCBNSSP≠0).

If the error task can be scheduled, IEAVEEE0 dequeues the top SRB and chains the SIRB to the error task, as described in step 4.

- The error task has any EUT mode FRRs (TCBNSSP=0).
- The vector facility environment is being established for the task, and an SRB has been scheduled to complete the environment (STCBPIQ=1).

Diagram SUP-6. Stage 3 Exit Effector (IEAVEEE0) (Part 3 of 6)



4 Get an XSB (if necessary) and queue the IRB or SIRB to the TCB.

- If the IRB or SIRB makes the task dispatchable, call memory switch.
- If there are more IQEs, RQEs, or SRBs to process
- When the IQE, RQE, and SRB chains have all been processed, perform processing necessary before the IRB or SIRB can be dispatched.

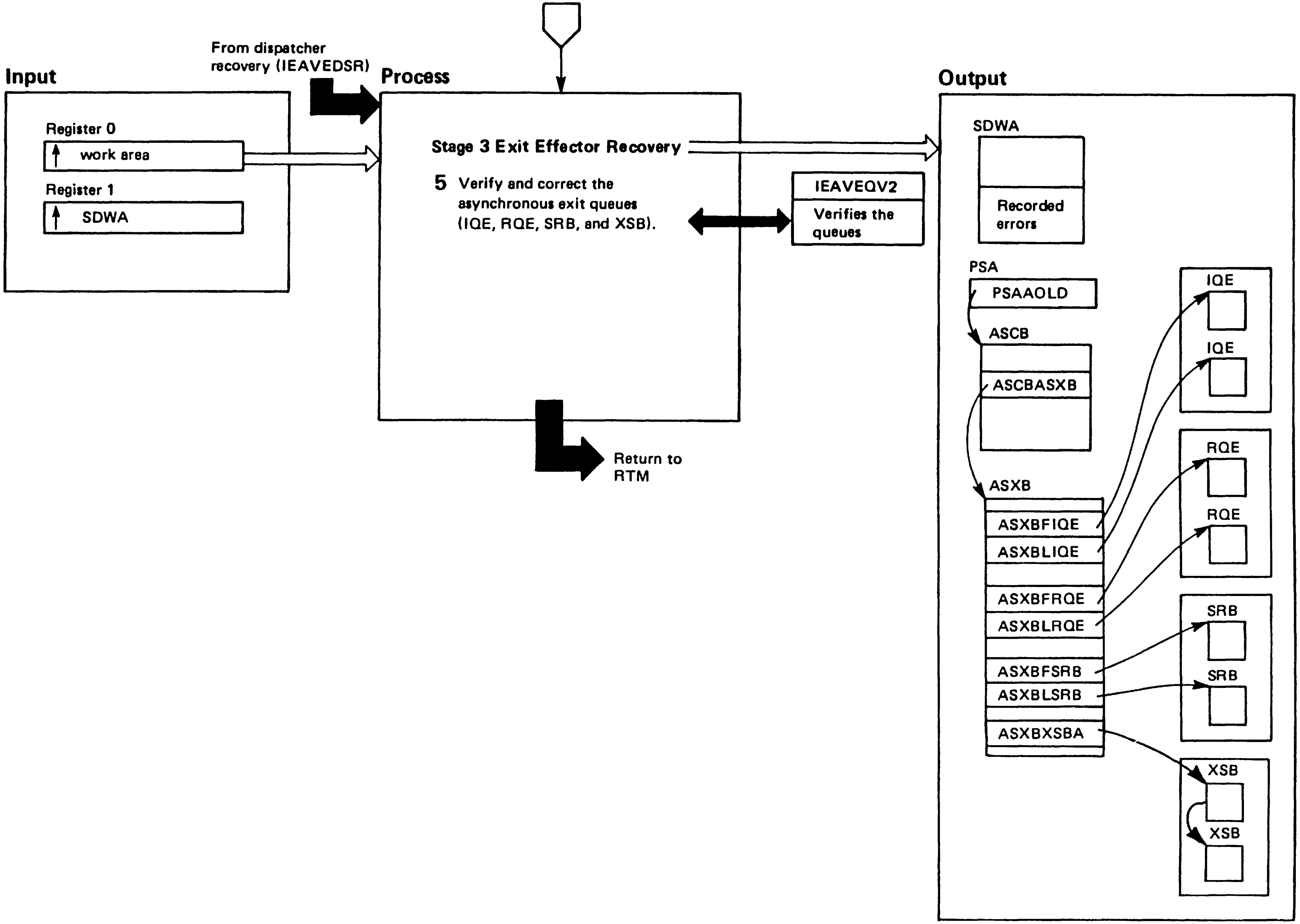
Diagram SUP-6. Stage 3 Exit Effector (IEAVEEE0) (Part 4 of 6)

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

4 In order to schedule the asynchronous routine, the stage 3 exit effector:

- Gets an XSB when preparing an IRB for scheduling. (Every RB on the TCB/RB queue must have an XSB; an SIRB already has one). If an XSB from a previous IRB is available (ASXBXSBA≠0), IEAVEEE0 uses it. Otherwise, it gets storage for a new one.
- Initializes the XSB for non-cross memory mode execution (the primary and secondary ASIDs equal the home ASID).
- Uses the PSW and TCB keys to form the key mask in the XSB.
- Places the IRB on the RB chain of the specified task. The IRB becomes the current RB for that task.
- Updates the TCBXSB field to point to the new top RB's XSB.
- Marks the IRB active (RBFACTV=1) so that any other requests for use of the same IRB will be deferred.
- Moves the saved registers of the previously current routine from the TCB to the IRB general register save area.
- Sets the address portion of the RBOPSW to the address specified in the RBEP field. This ensures that the dispatcher gives control to the asynchronous routine at the specified entry point.
- If the ASCBPER bit is on, turns on the PER bit in the RBOPSW.
- Sets the RBIQE to point to the queue element that scheduled the asynchronous routine (IQE, RQE, or SRB) so that the asynchronous exit gets control with specific register contents.
- If the task is ready but previously was not, increases the count of ready TCBs (ASCBTCBS) by one, updates the TCB ready pointer (ASCBTNEW) if the task is of higher priority than the current ASCBTNEW, and calls Memory Switch (at entry point IEAVEMS5) to check for a processor in a wait to pick up the task.

Diagram SUP-6. Stage 3 Exit Effector (IEAVEEE0) (Part 5 of 6)

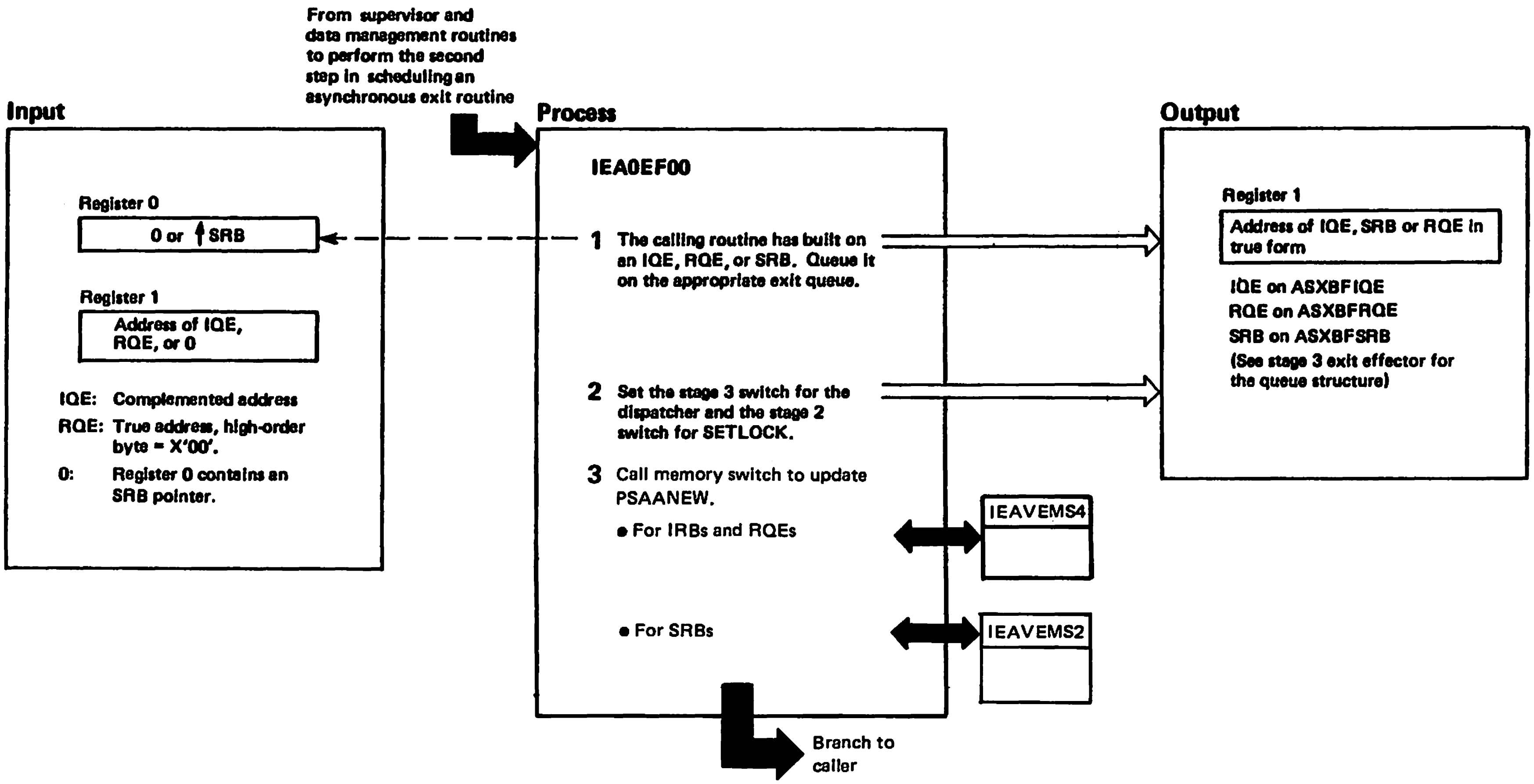


5 Verify and correct the asynchronous exit queues (IQE, RQE, SRB, and XSB).

Diagram SUP-6. Stage 3 Exit Effector (IEAVEEE0) (Part 6 of 6)

Extended Description	Module	Label
5 The stage 3 exit effector recovery routine verifies and corrects the exit effector queues (which consist of an IQE queue, an RQE queue, an SRB queue and an XSB queue). It uses the queue verifier (IEAVEQV0) to perform this verification. It calls the routine once for each queue. After each call, it will store a word of zeros into the recording area to delimit the end of the recorded output. The verification of each queue element is performed as follows: ● For an IQE, the address verification routine ensures that the IQE address, the TCB address contained in the IQE, and the IRB address contained in the IQE, are all referenceable. ● For an RQE, verification includes ensuring that the RQE storage and the IRB and TCB storage pointed to by RQERRQ and RQETCB are all referenceable. ● For an SRB, verification ensures that the SRB storage is referenceable. ● For an XSB, verification ensures that the XSB storage can be referenced.	IEAVEEER	IEAVEEEF

Diagram SUP-7. Stage 2 Exit Effector (IEAVEEE2) (Part 1 of 2)



From supervisor and data management routines to perform the second step in scheduling an asynchronous exit routine

Input

Register 0

0 or ↑SRB

Register 1

Address of IQE, RQE, or 0

IQE: Complemented address

RQE: True address, high-order byte = X'00'.

0: Register 0 contains an SRB pointer.

Process

IEA0EF00

1 The calling routine has built on an IQE, RQE, or SRB. Queue it on the appropriate exit queue.

2 Set the stage 3 switch for the dispatcher and the stage 2 switch for SETLOCK.

3 Call memory switch to update PSAANEW.

- For IRBs and RQEs
- For SRBs

IEAVEMS4

IEAVEMS2

Branch to caller

Output

Register 1

Address of IQE, SRB or RQE in true form

IQE on ASXBFIQE

RQE on ASXBFRQE

SRB on ASXBFSRB

(See stage 3 exit effector for the queue structure)

Diagram SUP-7. Stage 2 Exit Effector (IEAVEEE2) (Part 2 of 2)

Extended Description

Module	Label
IEAVEEE2	IEA0EF00

1 The exit queue on which the stage 2 exit affector places the input queue element depends on whether the queue element is an IQE (interruption queue element), an RQE (request queue element), or an SRB (service request block).

Type of Queue Element	Purpose	Type of Exit Queue
IQE	Supervisor routine wants to schedule an asynchronous exit routine.	ASXBFIQE ASXBLIQE
RQE	Data management routine wants to schedule an asynchronous routine.	ASXBFRQE ASXBLRQE
SRB	I/O supervisor wants to schedule an error recovery procedure (ERP).	ASXBFSRB ASXBLSRB

2 The stage exit effector sets the stage 3 switch (ASCBS3S=1) to indicate to the dispatcher that an asynchronous event is available for scheduling and causes the dispatcher to call the stage 3 exit effector.

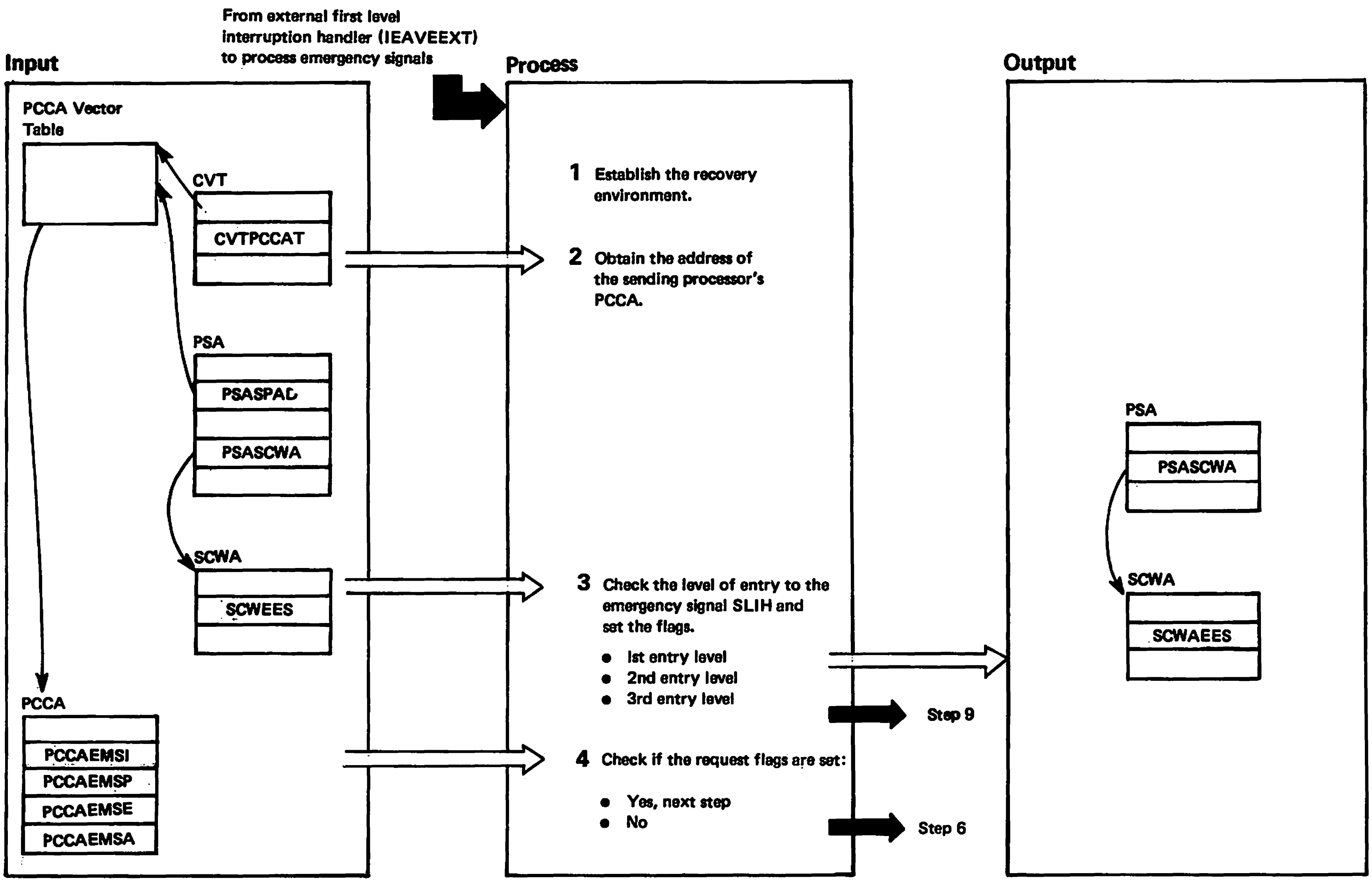
The SETLOCK service checks the stage 2 switch (ASCBS2S) when it releases the local lock.

3 Memory switch is invoked to alert the system of the new asynchronous work in the address space.

If the work is an SRB entry point IEAVEMS2 is called.

Otherwise IEAVEMS4 is called.

Diagram SUP-8. Emergency Signal Second Level Interruption Handler (IEAVEES) (Part 1 of 8)



1 Establish the recovery environment.

2 Obtain the address of the sending processor's PCCA.

3 Check the level of entry to the emergency signal SLIH and set the flags.

- Ist entry level
- 2nd entry level
- 3rd entry level

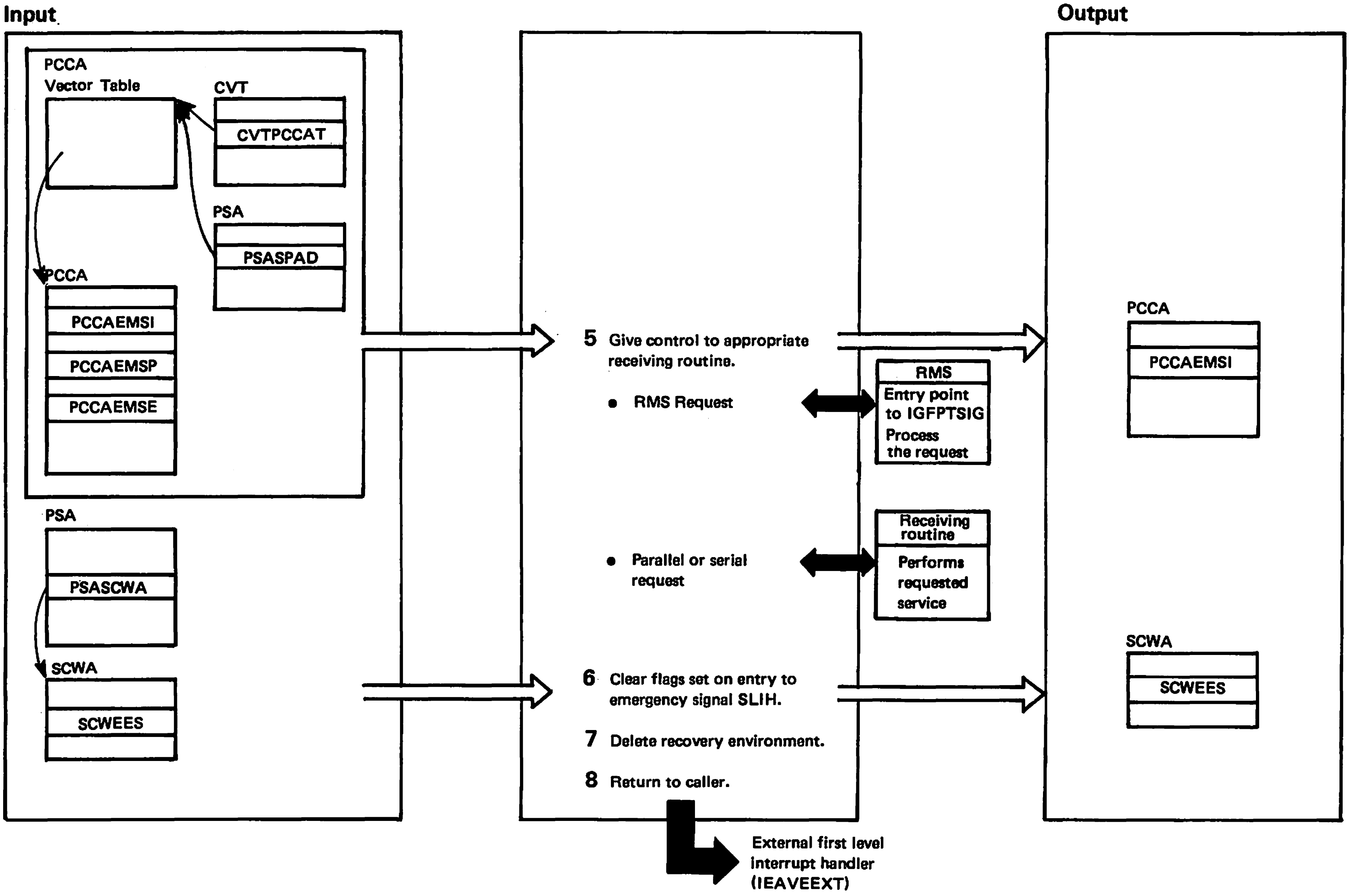
4 Check if the request flags are set:

- Yes, next step
- No

Diagram SUP-8. Emergency Signal Second Level Interruption Handler (IEAVEES) (Part 2 of 8)

Extended Description	Module	Label
When an emergency signal (EMS) interruption occurs, the external FLIH (IEAVEEXT) gives control to the emergency signal second level interruption handler (IEAVEES). IEAVEES routes control to the specified receiving routine to process the emergency signal.		
An emergency signal can be one of three types:		
● A recovery management support (RMS) request		
● A request for serial processing		
● A request for parallel processing		
For RMS requests, IEAVEES branches to an RMS service routine. If the request is for serial processing, the signalling routine and receiving routine execute serially. For parallel requests, they can execute simultaneously. Step 5 describes these differences in greater detail.		
1 IEAVEES establishes a recovery routine to handle errors in the receiving routines and to clear entry flags if an error occurs in the emergency signal SLIH.	IEAVEES	
2 IEAVEES indexes into the PCCA vector table, using the PSASPAD, to obtain the signalling processor's PCCA.		
3 IEAVEES checks the bits in SCWEES to determine the level of the entry. If this is the first or second entry level, IEAVEES sets the bits in SCWEES. If this is the third entry level, IEAVEES processes as an error and continues with step 9.		
4 IEAVEES checks the bits in PCCAEMSI of the sending processor to determine if there is a valid request. If there is no request, IEAVEES continues at step 6.		

Diagram SUP-8. Emergency Signal Second Level Interruption Handler (IEAVEES) (Part 3 of 8).



5 Give control to appropriate receiving routine.

- RMS Request
- Parallel or serial request

6 Clear flags set on entry to emergency signal SLIH.

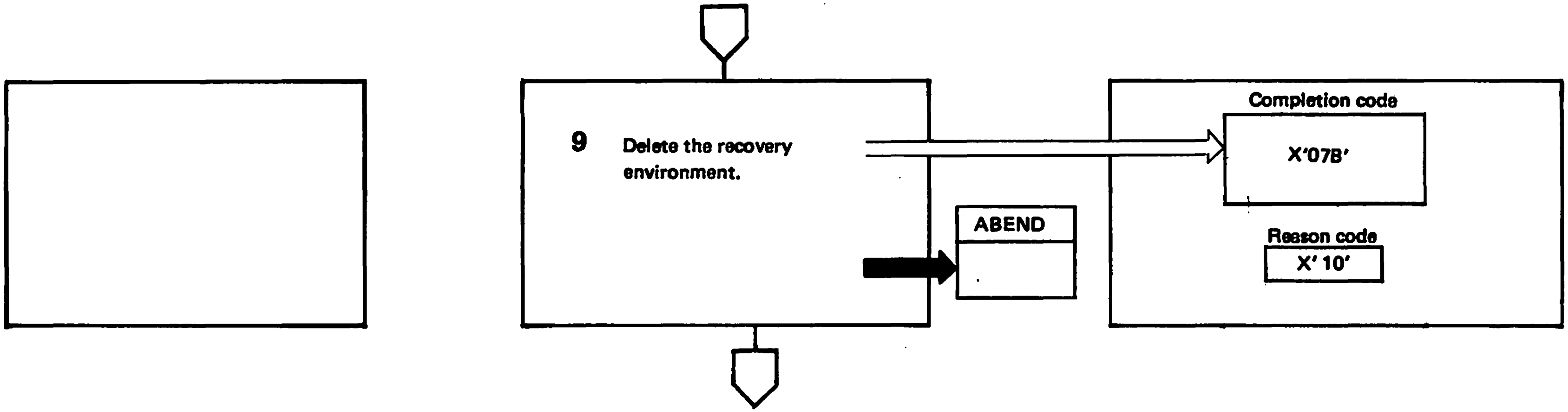
7 Delete recovery environment.

8 Return to caller.

Diagram SUP-8. Emergency Signal Second Level Interruption Handler (IEAVEES) (Part 4 of 8)

Extended Description	Module	Label
5 For an RMS request, IEAVEES gives control to the RMS service routine specified in a VCON. For both parallel and serial requests, IEAVEES gives control to the receiving routine specified in the PCCAEMSE field of the signalling processor's PCCA.		
When handling a serial request, IEAVEES informs the signalling processor that the request has been received, then gives control to the receiving routine. The signalling processor spins in IEAVERI while the receiving routine executes. After the receiving routine completes, IEAVEES clears the serial indicator bit in the PCCAEMSI buffer to allow IEAVERI to return control to the issuer of the RISGNL macro. If the receiving routine fails, IEAVEES sets the fail indicator in PCCAEMSI to allow IEAVERI to abend the signalling routine.	IEAVEES	RETRYPT
When handling a parallel request, IEAVEES clears the parallel indicator bit in the signalling processor's PCCAEMSI buffer before giving control to the receiving routine. This allows control to be returned to the routine issuing the RISGNL macro, which then may execute simultaneously with the receiving routine.		
6 IEAVEES cleans up the entry flags set in the SCWA (SCWEES) for IEAVEES.		RETRTN1
7 IEAVEES deletes the recovery environment.		
8 Exits to the external FLIH.		

Diagram SUP-8. Emergency Signal Second Level Interruption Handler (IEAVEES) (Part 5 of 8)



9 Delete the recovery environment.

Diagram SUP-8. Emergency Signal Second Level Interruption Handler (IEAVEES) (Part 6 of 8)

9 Second level recursion has occurred indicating that a receiving routine has opened a window without proper serialization. IEAVEES deletes the recovery environment and abends. This error causes the receiving routine to fail and IEAVEES passes the error to the signalling routine by setting a bit in PCCAEMSI.

Diagram SUP-8. Emergency Signal Second Level Interruption Handler (IEAVEES) (Part 7 of 8)

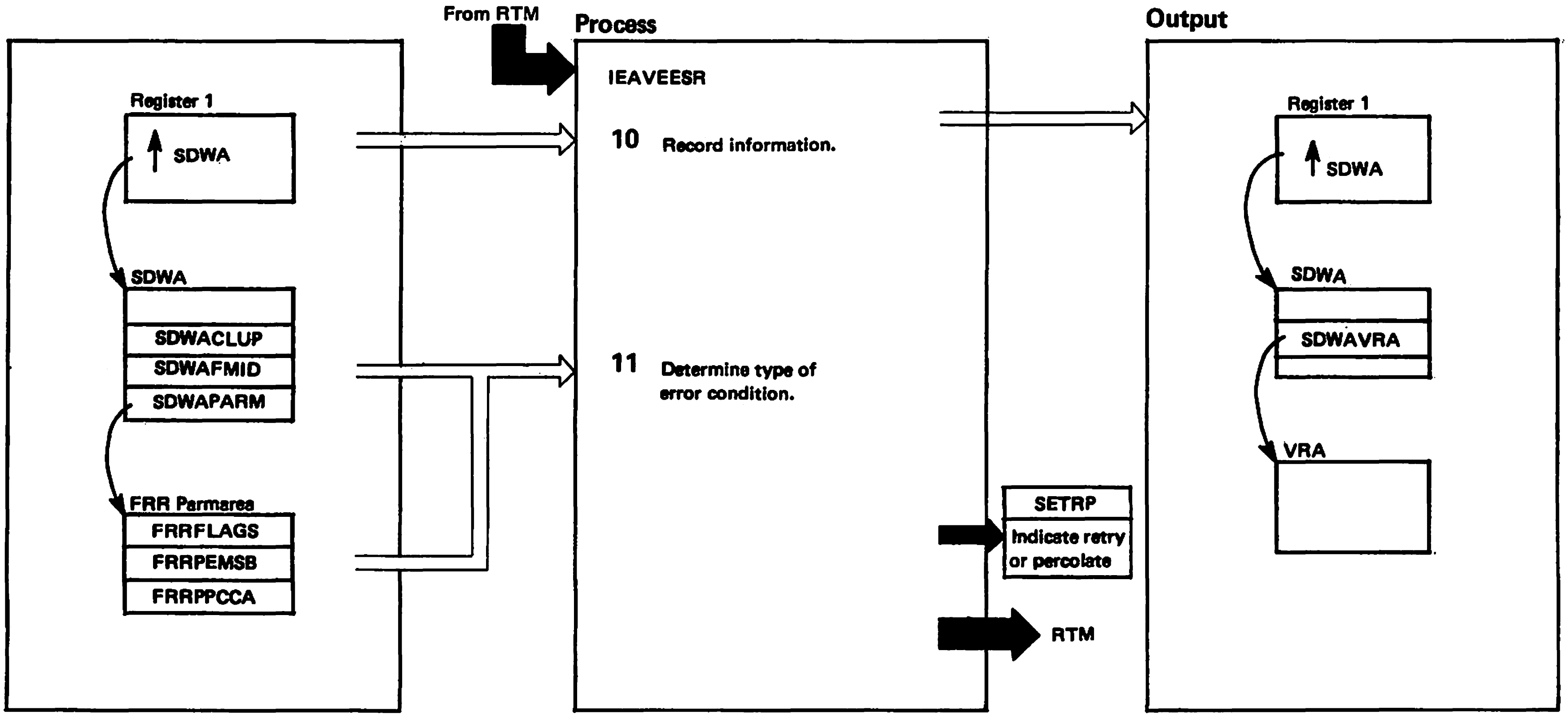
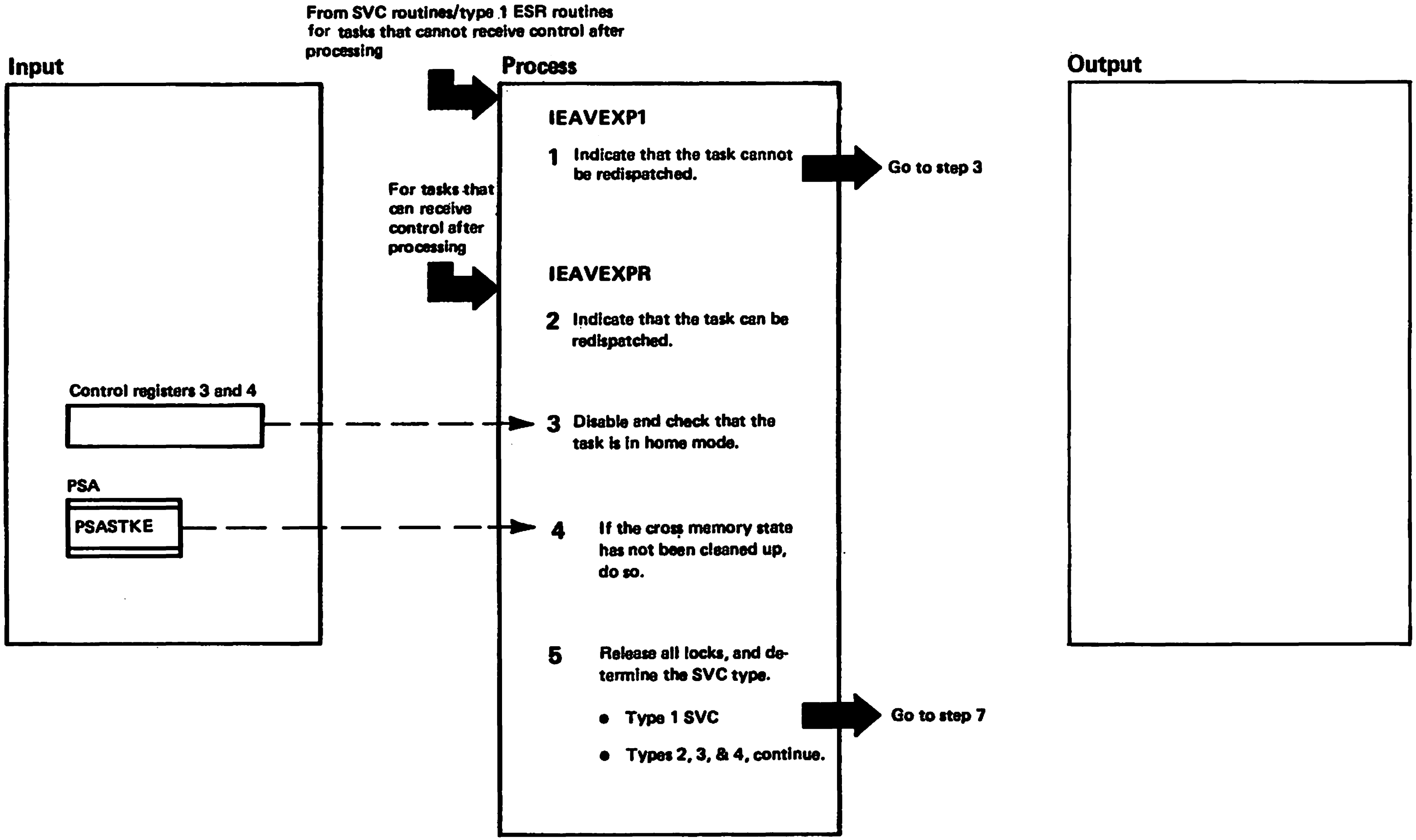


Diagram SUP-8. Emergency Signal Second Level Interruption Handler (IEAVEES) (Part 8 of 8)

Extended Description	Module	Label
10 Indicates error information in the SDWA and VRA areas.	IEAVEES	IEAVEESR
11 Depending on the reason entered, uses the SETRP macro, indicating to RTM to either retry to the mainline or to percolate.		

Diagram SUP-9. EXIT Prolog Processing (IEAVEEXP) (Part 1 of 6)


From SVC routines/type 1 ESR routines for tasks that cannot receive control after processing
Input
Process
Output
For tasks that can receive control after processing
Control registers 3 and 4
PSA
PSASTKE
Go to step 3
Go to step 7

IEAVEXP1

1 Indicate that the task cannot be redispatched.

IEAVEXPR

2 Indicate that the task can be redispatched.

3 Disable and check that the task is in home mode.

4 If the cross memory state has not been cleaned up, do so.

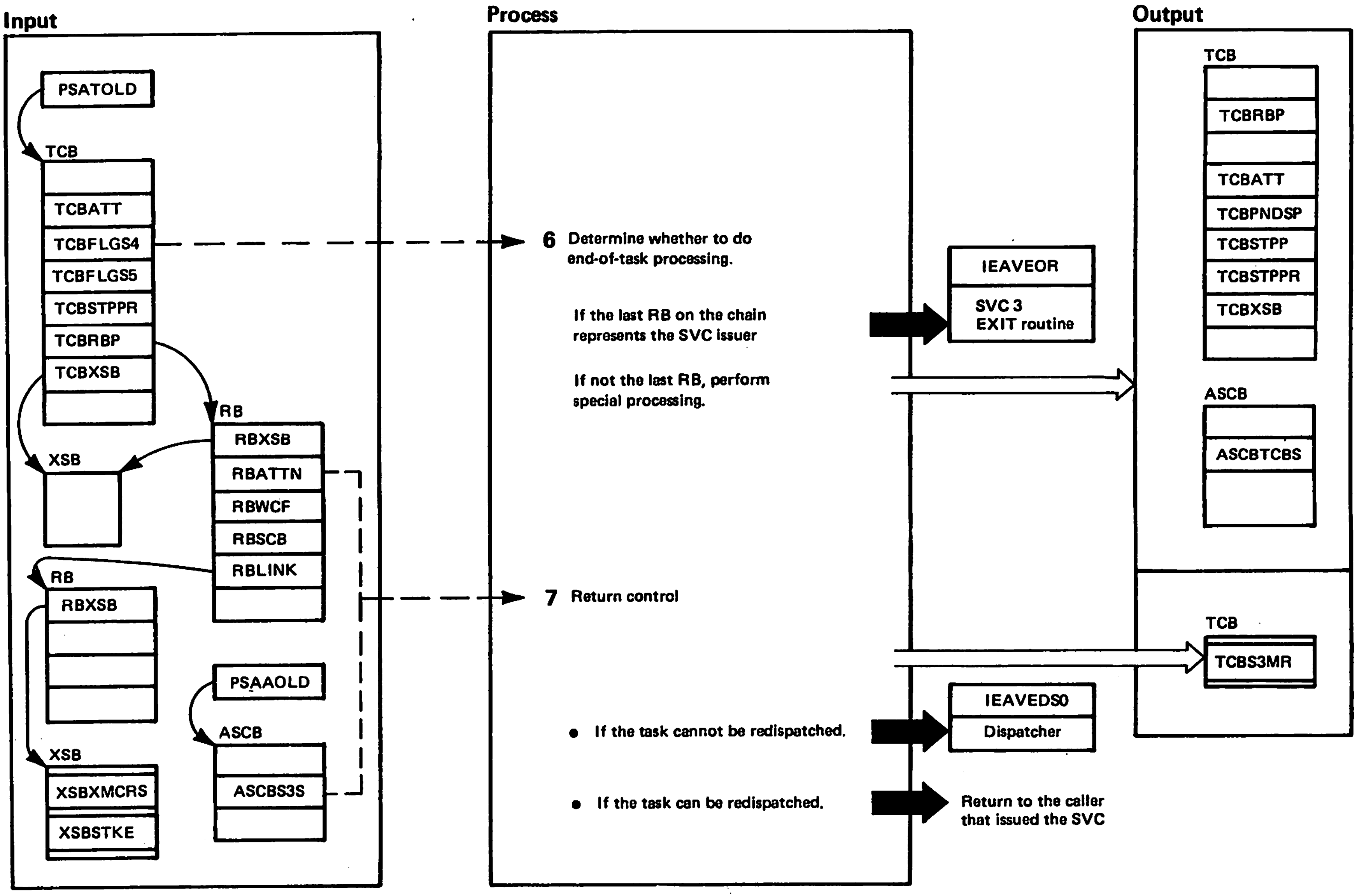
5 Release all locks, and determine the SVC type.

- Type 1 SVC
- Types 2, 3, & 4, continue.

Diagram SUP-9. EXIT Prolog Processing (IEAVEEXP) (Part 2 of 6)

Extended Description	Module	Label
EXIT prolog performs the exiting procedure for SVCs, including EXIT (SVC 3). The exiting SVC routine can provide information in registers 0, 1, and 15. EXIT prolog returns these registers to the SVC caller.		
1 EXIT prolog indicates the SVC issuer cannot be redispatched by setting the force dispatch switch in a register and goes to step 3. Routines that cannot be redispatched after EXIT prolog processing have EXIT prolog, when it finishes processing, pass control to the dispatcher.	IEAVEEXP	IEAVEXP1
2 EXIT prolog indicates that the SVC issuer can be redispatched after processing.		IEAVEXPR
3 EXIT prolog disables the processor for I/O and external interrupts and checks that the SVC exit is in home mode (that is, the exit has current addressability to the SVC issuer's TCB, RB, and XSB). If the exit is not in home mode, EXIT prolog establishes home mode by issuing a CMSET SET macro to the home address space.		
4 If the pointer to the PCLINK stack contains anything except 0, the cross memorv state has not been cleaned up correctly. EXIT prolog issues a PCLINK UNSTACK instruction to purge the PCLINK STACK.		
5 EXIT prolog releases all of the locks held by the caller of EXIT prolog and then determines the SVC type. If the SVC is type 1 (ASCBTYP1 bit equals 1), processing continues at step 7.		GOTYP1

Diagram SUP-9. EXIT Prolog Processing (IEAVEEXP) (Part 3 of 6)



6 Determine whether to do end-of-task processing.

If the last RB on the chain represents the SVC issuer

If not the last RB, perform special processing.

7 Return control

- If the task cannot be redispatched.
- If the task can be redispatched.

Return to the caller that issued the SVC

Diagram SUP-9. EXIT Prolog Processing (IEAVEEXP) (Part 4 of 6)

Extended Description	Module	Label
6 If the last RB on the RB chain represents the SVC issuer, for type 2, 3, and 4 SVCs, the EXIT prolog routine gives control to EXIT to perform end-of-task processing.		GOTOSVC3

EXIT prolog performs special processing for RBs other than the last:

Operation	Fields Read	Fields Modified
a) For type 1 SVCs, resets the type 1 switch.		ASCBFLG1 (ASCBTYP1 bit)
b) Completes STATUS processing for the RB unless other RBs indicate that stops cannot be done.	TCBATT TCBSTPPR	TCBATT TCBSTPPR TCBSTPP TCBPNDSP
c) Gives control to IEAVSETS at entry point IEAVESSS to perform stop SYNCH processing.		
d) If the task becomes nondispatchable, decreases the count of ready tasks.	RBLINK RBWCF TCBFLGS4 TCBFLGS5	ASCBTCBS
e) Dequeues the RB and XSB and marks the RB inactive.	RBLINK RBXSB	TCBRBP TCBXSB
f) Purges any SCBs by giving control to IEAVTSBP.	RBSCB	
g) Moves registers 2-14 from the RB into the TCB.		TCBGRS
h) Depending on how they were obtained, returns dynamic RBs and XSBs to the SVRB pool (RBNOCELL=1) or issues a FREEMAIN to free them (RBNOCELL=0).	RBFDYN RBNOCELL	

Extended Description	Module	Label
7 The EXIT prolog analyzes the task dispatchability state to determine where to pass control.		

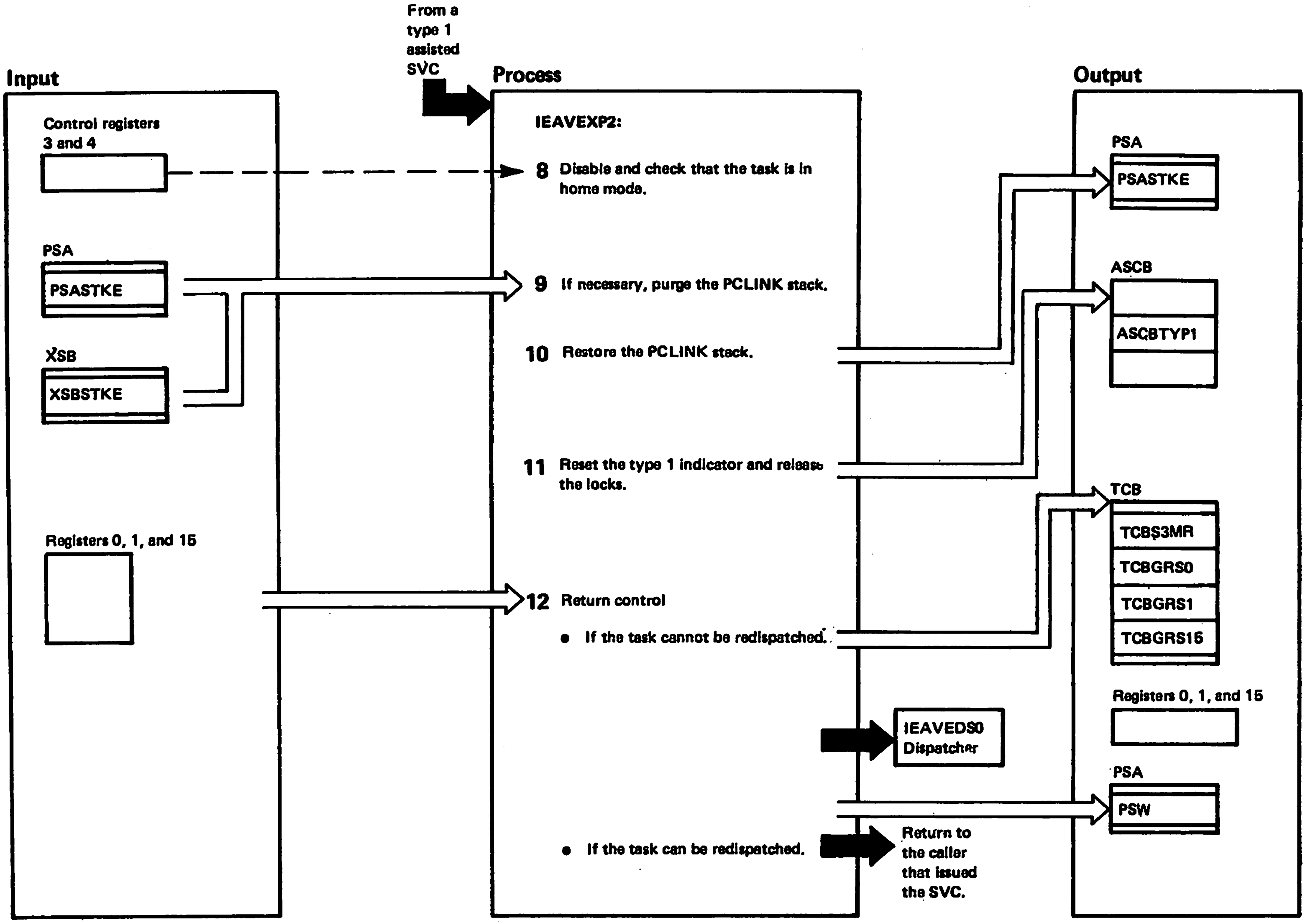
If both the stage 3 exit effector flag (ASCBS3S) and the TCB attention flag (TCBATT) are set, the attention exit must execute before the SVC issuer is given control. (The ASCBS3S indicates that an IQE has been queued and that the stage 3 exit effector is to be invoked to schedule an attention exit; the TCBATT flag indicates the task is not to have an attention exit scheduled on it by the stage 3 exit effector.) When the attention exit must execute before the SVC issuer is given control, EXIT prolog sets the TCBS3MR flag to make the task non-dispatchable, and gives control to the dispatcher at entry point IEA0DS1.

If the task is non-dispatchable for any of the following reasons, EXIT prolog also gives control to the dispatcher at entry point IEA0DS1:

- The ASCBS3S flag is set, but the TCBATT flag is not, indicating that an attention exit must be dispatched before the task.
- The force dispatcher switch is set, indicating that the task cannot be redispatched.
- The non-dispatchability flags (TCBFLGS4 and TCBFLGS5) are nonzero.
- The wait count (RBWCF) in the top RB is nonzero, indicating that the task is in a wait state.

If none of the above conditions are met, the task can be redispatched. EXIT prolog makes a trace entry; if necessary, resets the cross memory state to what it was when the SVC was issued by issuing a CMSET RESET macro; and returns to the caller that issued the SVC.

Diagram SUP-9. EXIT Prolog Processing (IEAVEEXP) (Part 5 of 6)



IEAVEXP2:

8 Disable and check that the task is in home mode.

9 If necessary, purge the PCLINK stack.

10 Restore the PCLINK stack.

11 Reset the type 1 indicator and release the locks.

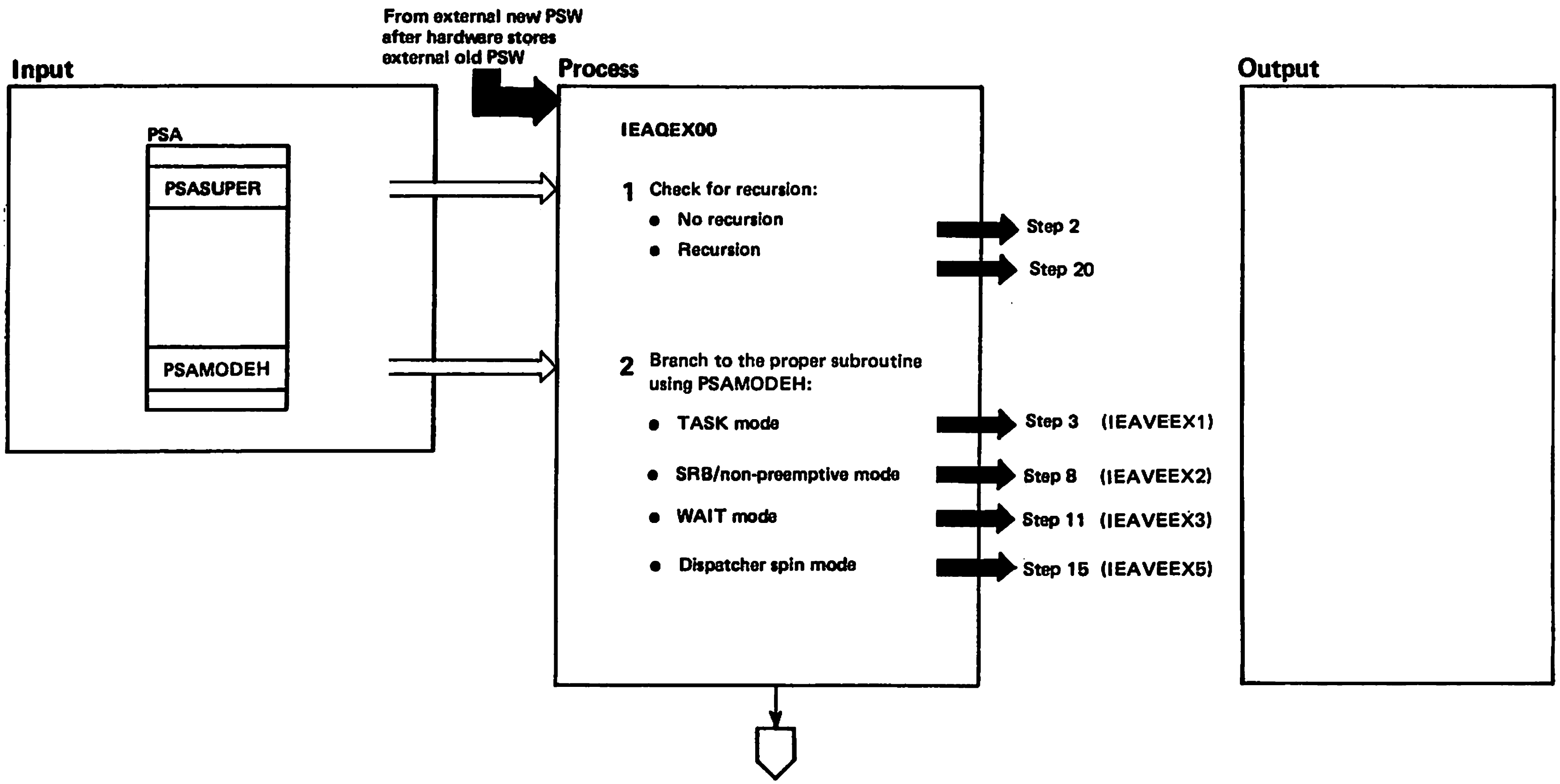
12 Return control

- If the task cannot be redispatched.
- If the task can be redispatched.

Diagram SUP-9. EXIT Prolog Processing (IEAVEEXP) (Part 6 of 6)

Extended Description	Module	Label
8 EXIT prolog disables the processor for I/O and external interrupts and checks that the SVC exit is in home mode (that is, the exit has current addressability to the SVC issuer's TCB, RB, and XSB). If the exit is not in home mode, EXIT prolog establishes home mode by issuing a CMSET SET macro to the home address space.		IEAVEXP2
9 If the current PCLINK stack is not empty, EXIT prolog issues a PCLINK UNSTACK macro with the PURGE option to clear the PCLINK stack.		
10 EXIT prolog restores the PCLINK stack for the issuer of the SVC by placing a pointer to the PCLINK stack in the PSASTKE.		
11 EXIT prolog resets the type 1 SVC indicator (ASCBTYP1) and releases the locks it currently holds. If only the local lock is held, EXIT prolog uses SETLOCKI to release it.		
12 If the task is non-dispatchable for any of the following reasons, EXIT prolog gives control to the dispatcher. Before exiting to the dispatcher, EXIT prolog saves the exiting SVC's registers 0, 1, and 15. • The stage 3 exit effector flag (ASCBS3S) is set. • The non-dispatchability flags (TCBFLGS4 and TCBFLGS5) are set. • The wait count (RBWCF) in the top RB is nonzero, indicating the task is in a wait state. Otherwise, EXIT prolog returns to the issuer of the SVC. Before exiting, EXIT prolog does the following: • Builds a PSW in the PSA. • Traces the SVC return. • Flushes the normal FRR stack. • Restores the SASID and PKM to the values current before the SVC was issued. • Restores the original contents of registers 2-14 from the TCB.		

Diagram SUP-10. External First Level Interrupt Handler (IEAVEEXT) (Part 1 of 16)



IEAQEX00

1 Check for recursion:

- No recursion → Step 2
- Recursion → Step 20

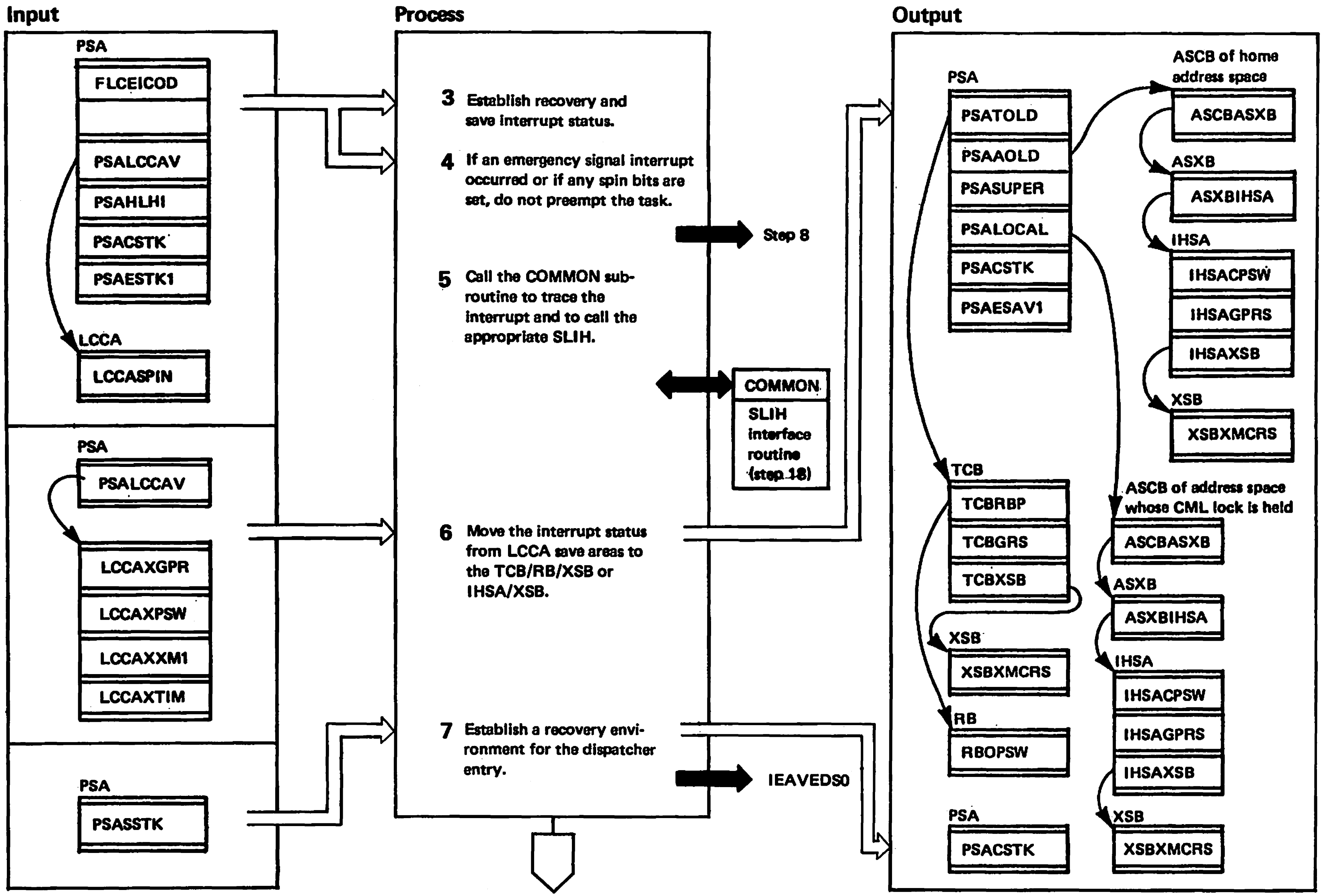
2 Branch to the proper subroutine using PSAMODEH:

- TASK mode → Step 3 (IEAVEEX1)
- SRB/non-preemptive mode → Step 8 (IEAVEEX2)
- WAIT mode → Step 11 (IEAVEEX3)
- Dispatcher spin mode → Step 15 (IEAVEEX5)

Diagram SUP-10. External First Level Interrupt Handler (IEAVEEXT) (Part 2 of 16)

Extended Description	Module	Label
The external FLIH gets control from the external new PSW. The FLIH uses the system mode indicator to determine how to save interrupt status. IEAVEEXT calls a subroutine to trace the interrupt and route control to the appropriate SLIH. The FLIH also supports recursive interrupts if an interrupt occurs while an external SLIH is in control.		
1 The external FLIH checks a super bit (PSAEXT bit in PSASUPER) to determine if this is a recursive entry. If it is, processing continues at step 15.	IEAVEEXT	IEAQEX00
2 The system mode at the time of the interrupt determines which subroutine in the external FLIH should handle the interrupt. IEAVEEXT uses the PSAMODEH as an index to reference the proper routine.		

Diagram SUP-10. External First Level Interrupt Handler (IEAVEEXT) (Part 3 of 16)



3 Establish recovery and save interrupt status.

4 If an emergency signal interrupt occurred or if any spin bits are set, do not preempt the task.

5 Call the COMMON subroutine to trace the interrupt and to call the appropriate SLIH.

6 Move the interrupt status from LCCA save areas to the TCB/RB/XSB or IHSA/XSB.

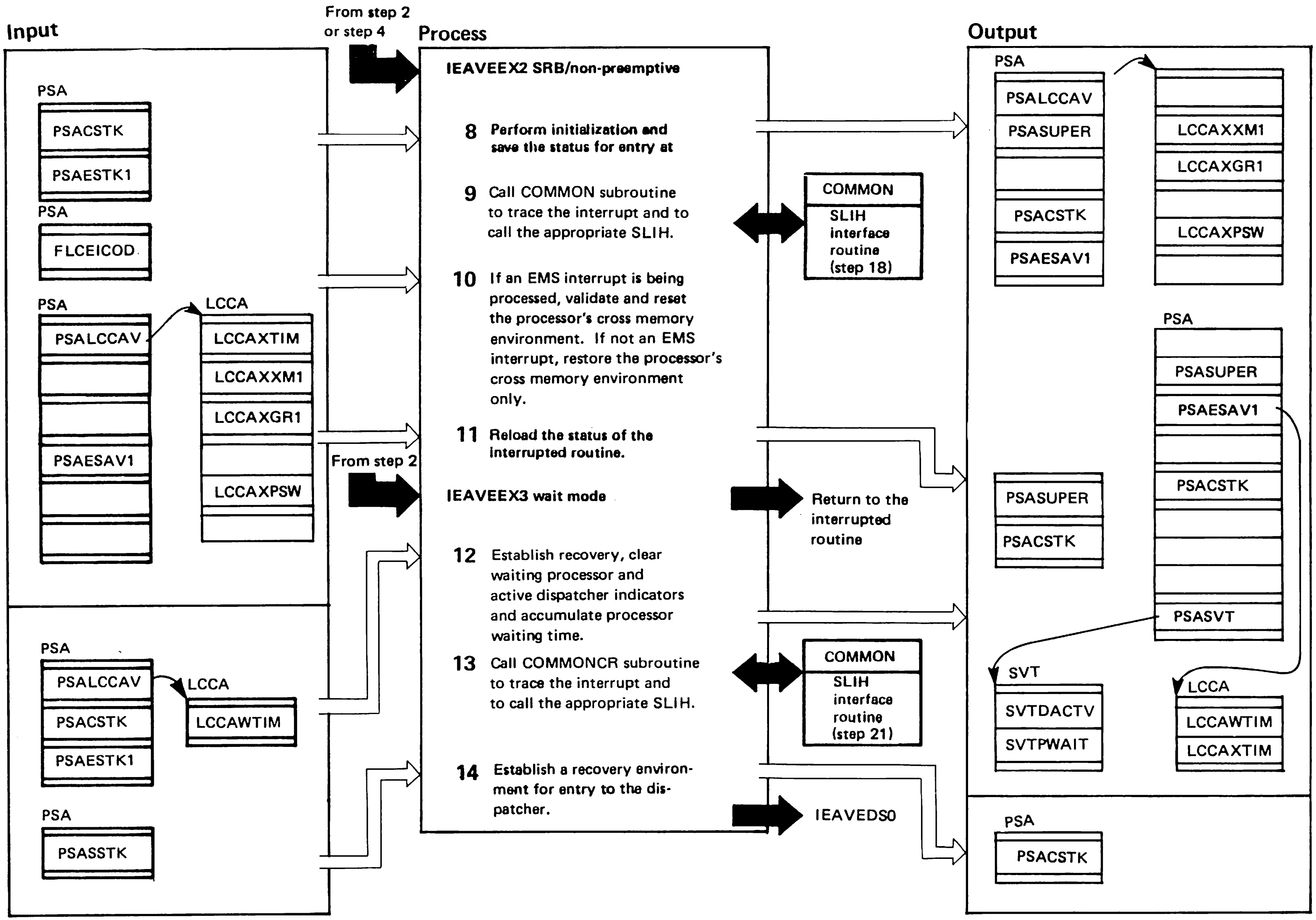
7 Establish a recovery environment for the dispatcher entry.

Diagram SUP-10. External First Level Interrupt Handler (IEAVEEXT) (Part 4 of 16)

Extended Description	Module	Label
The external FLIH mainline gives control to IEAVEEX1 when the system is in task mode.		
3 IEAVEEX1:		
● Saves the current FRR stack pointer (PSACSTK) in the PSAESAV1 field. ● Sets the current FRR stack pointer to the external FLIH stack (PSAESTK1). ● Sets the external FLIH super bit in PSASUPER. ● Saves the CPU timer in LCCAXTIM ● Saves cross memory control registers 3 and 4 in the LCCAXXM1 field. ● Saves the interrupt registers in LCCAXGR1. ● Saves the interrupt PSW in LCCAXPSW.	IEAVEEXT	IEAVEEX1
4 If an emergency signal interrupt occurred or if any spin bits are on, the task is not pre-empted. Instead, processing continues at step 8.		
5 IEAVEEX1 calls the COMMON subroutine which traces the interrupt via system trace and GTF and calls the appropriate SLIH. After the SLIH completes, it returns control here.		
6 If the home address space is not already the primary address space, IEAVEEX1 issues a CMSET SET macro to make it so. This allows IEAVEEX1 to store into the TCB/RB/XSB or IHSA/XSB.		
IEAVEEX1 saves the status of preemptible work as follows.		
If the interrupted task is unlocked, IEAVEEX1: ● Saves the interrupted task's registers in TCBGRS from LCCAXGPR ● Saves the PSW in RBOPSW from LCCAXPSW. ● Copies the interrupted cross memory control registers 3 and 4 from the LCCAXXM1 field into the XSBXMCRS field.		

Extended Description	Module	Label
If the interrupted task holds the LOCAL lock, IEAVEEX1 saves status in the IHSA and XSB of the home address space. It: ● Saves the registers in IHSAGPRS from LCCAXGPR field. ● Saves the PSW in IHSACPW from LCCAXPSW field. ● Copies the interrupted cross memory control registers 3 and 4 from the LCCAXXM1 field into the XSBXMCRS field. ● If the interrupted task holds the CML lock, IEAVEEX1 saves status in the IHSA and XSB of the address space whose CML lock is held. It: – Issues a CMSET SSARTO macro to establish secondary addressability to the address space whose CML lock is held. – Saves the interrupted task's registers in the IHSAGPRS field from LCCAXGPR. – Saves the PSW in the IHSACPSW field from LCCAXPSW field. – Copies the interrupted cross memory control registers 3 and 4 from the LCCAXXM1 field to the XSBXMCRS field.		
7 IEAVEEX1 sets the current FRR stack pointer to the super stack (PSASSTK) and branches to the dispatcher at entry point IEAPDS7, IEAPDS7A, or IEAPDS7C.		

Diagram SUP-10. External First Level Interrupt Handler (IEAVEEXT) (Part 5 of 16)



IEAVEEX2 SRB/non-preemptive

8 Perform initialization and save the status for entry at

9 Call COMMON subroutine to trace the interrupt and to call the appropriate SLIH.

10 If an EMS interrupt is being processed, validate and reset the processor's cross memory environment. If not an EMS interrupt, restore the processor's cross memory environment only.

11 Reload the status of the interrupted routine.

IEAVEEX3 wait mode

12 Establish recovery, clear waiting processor and active dispatcher indicators and accumulate processor waiting time.

13 Call COMMONCR subroutine to trace the interrupt and to call the appropriate SLIH.

14 Establish a recovery environment for entry to the dispatcher.

Diagram SUP-10. External First Level Interrupt Handler (IEAVEEXT) (Part 6 of 16)

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

IEAVEEX2 receives control from the external FLIH mainline when the system is in SRB mode, or from IEAVEEX1qwhen an interrupted task is not to be preempted.

8 IEAVEEX2: IEAVEEXT IEAVEEX2

- Saves the current FRR stack pointer (PSACSTK) in the PSAESAV1 field.
- Sets the current stack pointer to the external FLIH stack (PSAESTK1).
- Sets the external FLIH super bit in PSASUPER.
- Saves the processor timer in LCCAXTIM.
- Saves the interrupted cross memory control registers 3 and 4 in the LCCAXXM1 field.
- Saves the registers in LCCAXGR1
- Saves the PSW in LCCAXPSW

9 IEAVEEX2 calls the COMMON subroutine, which traces the interrupt via system trace, and GTF, and calls the appropriate SLIH. After the SLIH completes, it returns control here.

10 If an EMS interrupt is being processed, IEAVEEX2 issues a CMSET RESET CHKAUTH=(YES) macro. The macro processor (IEAVECMS) checks whether the primary and secondary ASIDs can be accessed using cross memory instructions and whether they can access each other. If they can, IEAVECMS reloads cross memory control registers 1, 3, 4, 5, and 7. If the cross memory state is invalid, IEAVECMS issues abend X'058' with the appropriate reason code.

IEAVEEX2 issues the CMSET macro in case the bind break routine (IEAVEBBR) caused the EMS interrupt in order to validate and reset the processor's cross memory environment. See the IEAVEBBR diagram for details on when IEAVEBBR issues an EMS. See the IEAVECMS diagram for a description of CMSET RESET processing.

If the interrupt was not an EMS interrupt, IEAVEEX2 issues a CMSET RESET, CHKAUTH=NO macro. This restores the cross memory controls to the state they were in when the interrupt occurred.

11 IEAVEEX2 reloads the status of the interrupted routine and returns control to the interrupted routine.

The external FLIH mainline gives IEAVEEX3 control when the system is in wait mode.

12 IEAVEEX3: IEAVEEXT IEAVEEX3

- Saves the current FRR stack pointer (PSACSTK) in the PSAESAV1 field.
- Sets the current stack pointer to the external stack 1 (PSAESTK1).
- Sets the external FLIH super bit in PSASUPER.
- Saves the processor timer in PSACPUT.
- Accumulates wait time in LCCAWTIM.
- Clears SVTDACTV and SVTPWAIT.

13 IEAVEEX3 calls the COMMONCR subroutine, which traces the interrupt via GTF and calls the appropriate SLIH. After the SLIH completes, it returns control here.

14 IEAVEEX3 sets the current FRR stack pointer to the super stack (PSASSTK) and branches to the dispatcher at entry point IEAPDS7B.

Diagram SUP-10. External First Level Interrupt Handler (IEAVEEXT) (Part 7 of 16)

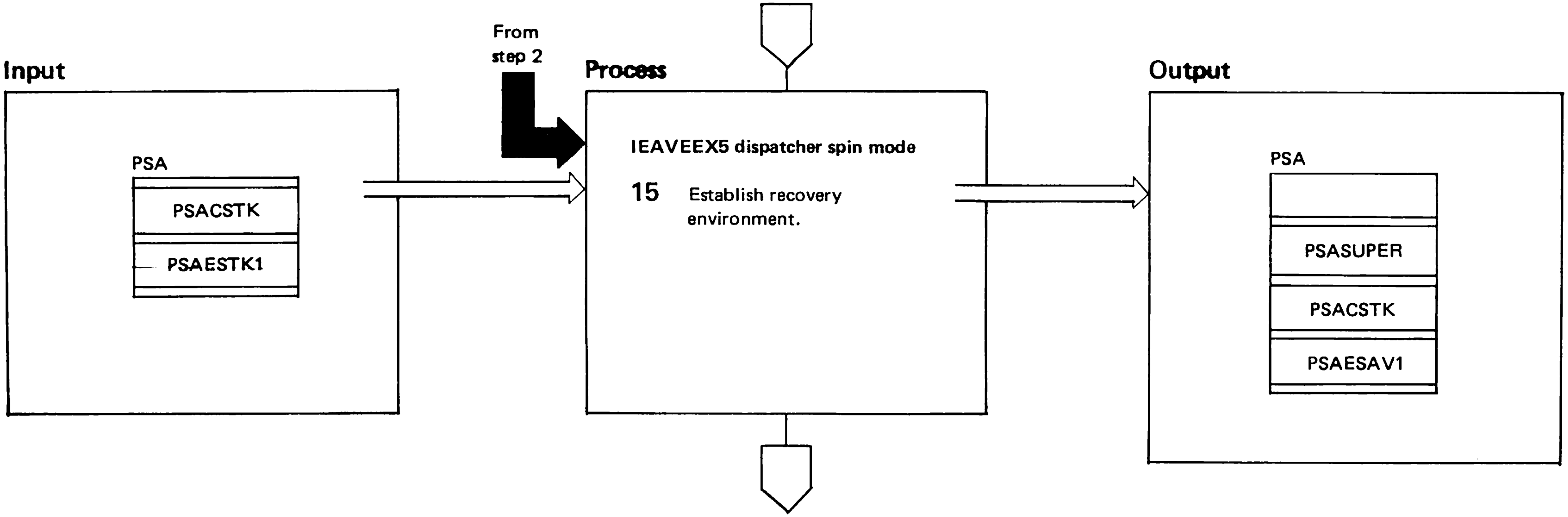
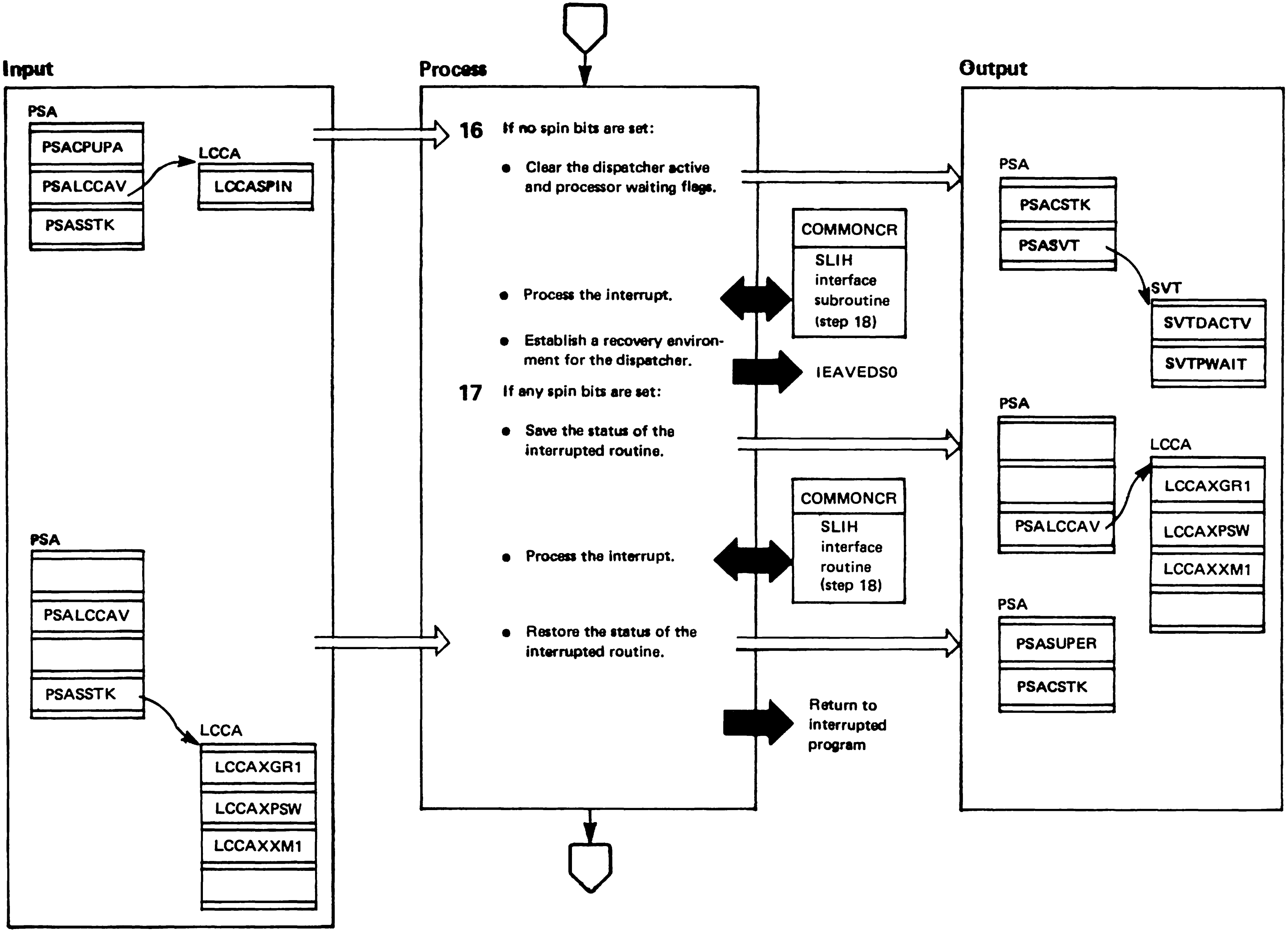


Diagram SUP-10. External First Level Interrupt Handler (IEAVEEXT) (Part 8 of 16)

Extended Description	Module	Label
The external FLIH mainline gives IEAVEEX5 control when an interrupt occurs while the dispatcher is making a recursive scan of the dispatching queues before entering a wait state, or while the dispatcher is spinning on an intersect flag waiting to proceed.		
15 IEAVEEX5:	IEAVEEXT	IEAVEEX5
● Saves the current FRR stack pointer in the PSAESAV1 field. ● Sets the external FLIH super bit in PSASUPER.		

Diagram SUP-10. External First Level Interrupt Handler (IEAVEEXT) (Part 9 of 16)



16 If no spin bits are set:

- Clear the dispatcher active and processor waiting flags.
- Process the interrupt.
- Establish a recovery environment for the dispatcher.

17 If any spin bits are set:

- Save the status of the interrupted routine.
- Process the interrupt.
- Restore the status of the interrupted routine.

COMMONCR
SLIH interface subroutine (step 18)
IEAVEDS0
COMMONCR
SLIH interface routine (step 18)
Return to interrupted program

Diagram SUP-10. External First Level Interrupt Handler (IEAVEEXT) (Part 10 of 16)

Extended Description	Module	Label

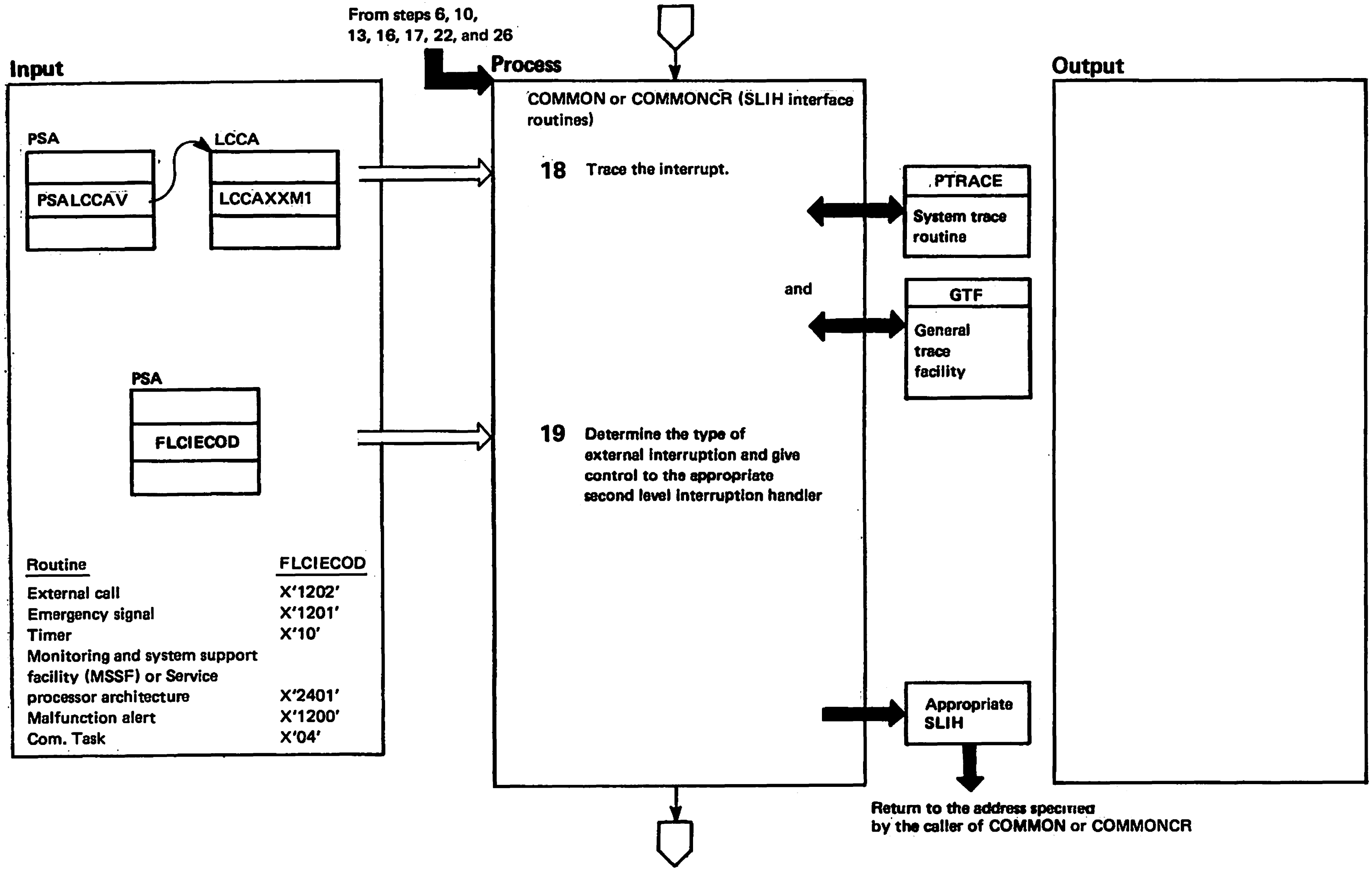
16 When no spin bits are set, IEAVEEX5 preempts the interrupted routine. It:

- Clears the dispatcher active and processor waiting flags (the SVTDACTV and SVTPWAIT fields, respectively).
- Branches to the COMMONCR subroutine (step 18) which traces the interrupt and routes control to the appropriate SLIH.
- Makes the super stack current for the dispatcher.
- Branches to the dispatcher at entry point IEAPDS7B.

17 When spin bits are set, instead of preempting the interrupted routine, IEAVEEX5 returns control to it after the interrupt has been processed. IEAVEEX5:

- Saves the interrupted routine's registers in the LCCAXGR1 field.
- Saves the interrupted routine's PSW in the LCCAXPSW field.
- Saves the interrupted routine's cross memory control registers 3 and 4 in the LCCAXXM1 field.
- Branches to the COMMONCR subroutine (step 18), which traces the interrupt and routes control to the appropriate SLIH.
- After the SLIH returns control, restores the status of the interrupted routine, and returns to it.

Diagram SUP-10. External First Level Interrupt Handler (IEAVEEXT) (Part 11 of 16)



COMMON or COMMONCR (SLIH interface routines)

18 Trace the interrupt.

19 Determine the type of external interruption and give control to the appropriate second level interruption handler

Return to the address specified by the caller of COMMON or COMMONCR

Routine	FLCIECOD
External call	X'1202'
Emergency signal	X'1201'
Timer	X'10'
Monitoring and system support facility (MSSF) or Service processor architecture	X'2401'
Malfunction alert	X'1200'
Com. Task	X'04'

Diagram SUP-10. External First Level Interrupt Handler (IEAVEEXT) (Part 12 of 16)

18 The COMMON and COMMONCR routines trace the external interrupt by invoking the system trace and the generalized trace facility (GIF). IEAVEEXT obtains cross memory status from the LCCAXXM1 in COMMON and from the current control registers 3 and 4 in COMMONCR. The routines are otherwise the same.

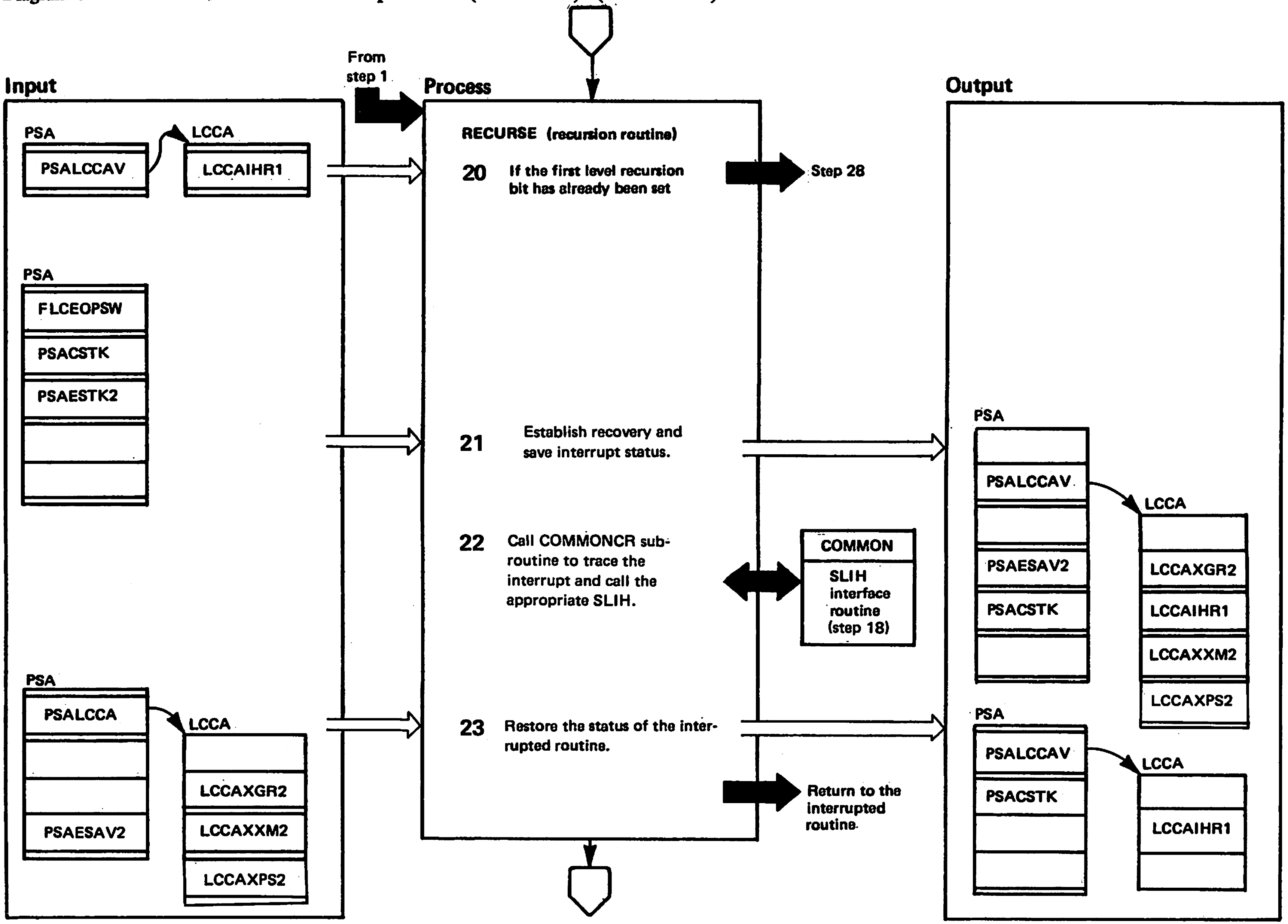
IEAVEEXT	COMMON COMMONCR

19 The external FLIH determines what type of external interrupt occurred and routes control to the appropriate SLIH.

- External call (IEAVEXS) — Occurs after a user issues an external call SIGP (signal processor) instruction.
- Emergency signal (IEAVEES) — Occurs after a user issues an emergency signal SIGP instruction.
- Timer (module IEAVRTI0, entry point IEA0TI00) — Occurs when a selected timer interval expires or when a TOD synch check occurs.
- Monitoring and system support facility (MSSF) or Service processor architecture (IEAVMFIH) — Occurs after the service processor signals the completion of a software-requested service.
- Malfunction alert (IGFPXMFA) — Occurs if another processor fails.
- Communications task (IEEBC1PE) — Occurs when the operator presses the external interrupt key on the operator's console.

When the SLIH completes, it returns control to the caller of COMMON or COMMONCR.

Diagram SUP-10. External First Level Interrupt Handler (IEAVEEXT) (Part 13 of 16)



RECURSE (recursion routine)

20 If the first level recursion bit has already been set

21 Establish recovery and save interrupt status.

22 Call COMMONCR subroutine to trace the interrupt and call the appropriate SLIH.

23 Restore the status of the interrupted routine.

Diagram SUP-10. External First Level Interrupt Handler (IEAVEEXT) (Part 14 of 16)

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

The RECURSE routine receives control when an external interruption occurs while an external SLIH is active. Two levels of recursion are allowed.

20 IEAVEEXT determines if this is the first or second recursion. If this is the second recursion, continues processing at step 25. IEAVEEXT RECURSE

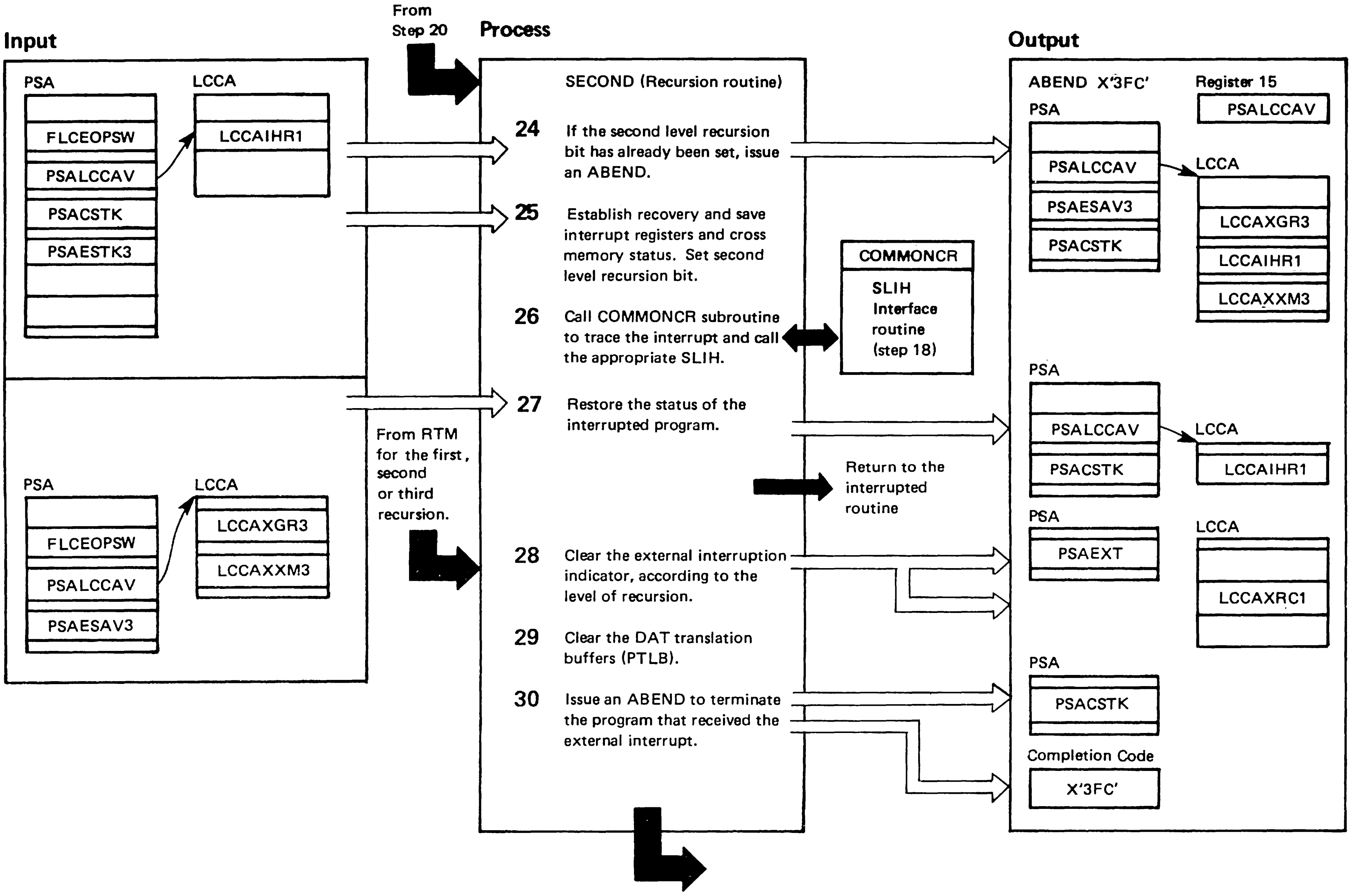
21 For the first level recursion, IEAVEEXT:

- Saves the current FRR stack pointer (PSACSTK) in PSAESAV2 field.
- Sets the current FRR stack pointer to the external stack 2 (PSAESTK2)
- Saves the registers in LCCAXGR2.
- Saves the old PSW in LCCAXPS2.
- Saves the cross memory control registers 3 and 4 in LCCAXXM2.
- Sets the recursion flag in LCCAIHR1.

22 RECURSE calls the COMMONCR subroutine to system and GTF trace the interrupt and to call the appropriate SLIH. After the SLIH completes, control is returned to here.

23 IEAVEEXT restores the status of the interrupted routine and returns control to it.

Diagram SUP-10. External First Level Interrupt Handler (IEAVEEXT) (Part 15 of 16)



SECOND (Recursion routine)

24 If the second level recursion bit has already been set, issue an ABEND.

25 Establish recovery and save interrupt registers and cross memory status. Set second level recursion bit.

26 Call COMMONCR subroutine to trace the interrupt and call the appropriate SLIH.

27 Restore the status of the interrupted program.

28 Clear the external interruption indicator, according to the level of recursion.

29 Clear the DAT translation buffers (PTLB).

30 Issue an ABEND to terminate the program that received the external interrupt.

Diagram SUP-10. External First Level Interrupt Handler (IEAVEEXT) (Part 16 of 16)

Extended Description	Module	Label
If an external interrupt occurs while IEAVEEXT is processing a previous recursive external interrupt, IEAVEEXT gives control to the SECOND subroutine.	IEAVEEXT	SECOND

24 If the second level recursion indicator (LCCAIHR1) has already been set, IEAVEEXT gives control to the THIRD subroutine to issue ABEND X'3FC' with reason code 0.

25 For the second level recursion, SECOND:

- Saves the current FRR stack pointer (PSACSTK) in the PSAESAV3 field.
- Sets the current FRR stack pointer (PSACSTK) to the external stack 3 (PSAESTK3).
- Saves the registers in LCCAXGR3.
- Saves cross memory control registers 3 and 4 in the LCCAXXM3 field.
- Sets the second level recursion indicator (the LCCAIHR1 flag).

26 SECOND calls the COMMONCR subroutine to trace the interrupt via system trace and GTF and to call the appropriate SLIH. After the SLIH completes, it returns control here.

27 After the SLIH returns, SECOND restores the status of the interrupted routine and returns control to the interrupted routine.

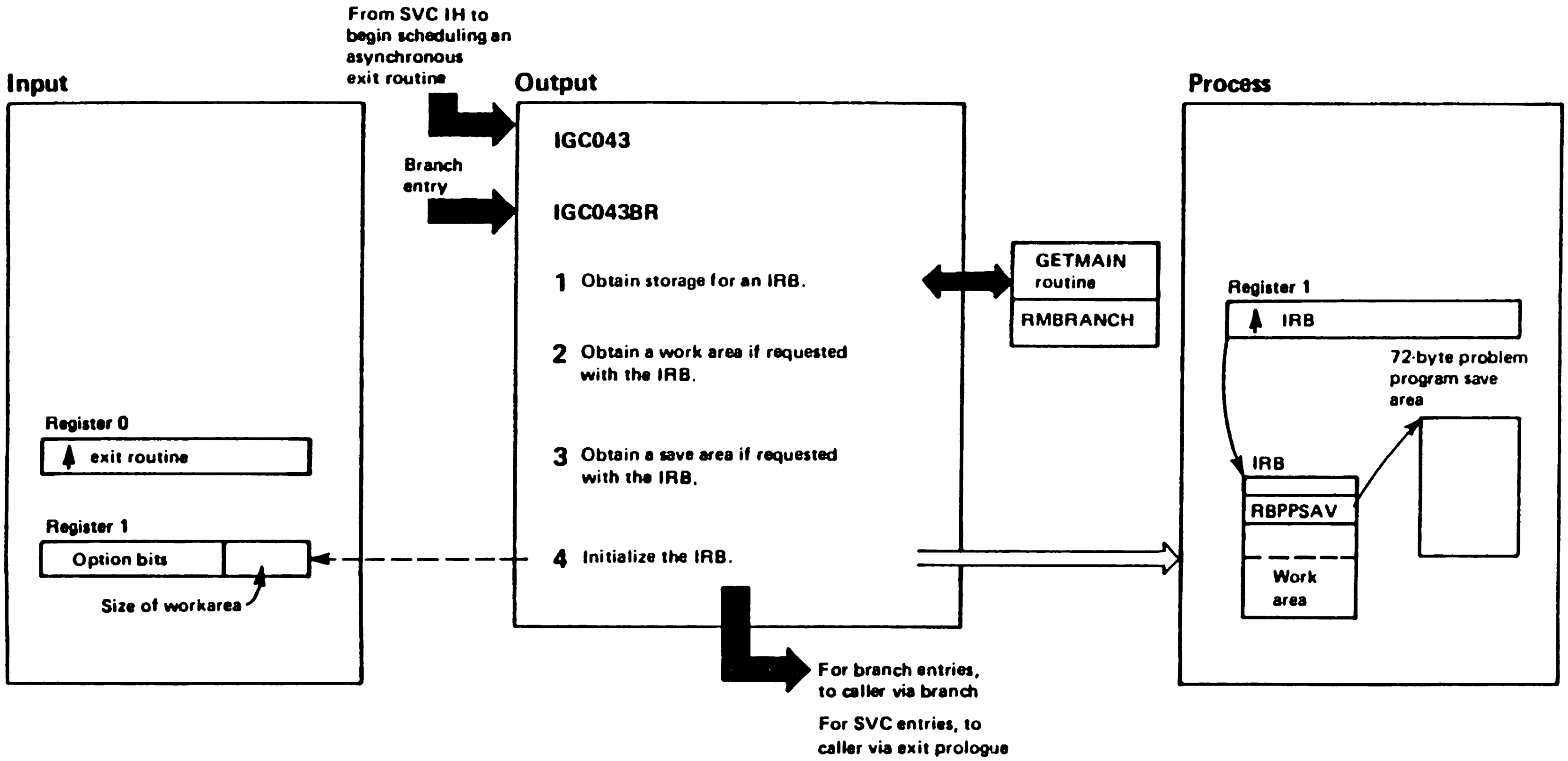
28 The external FLIH has three retry routines which the supervisor FRR (IEAVESPR) will retry to. The retry routine is chosen based upon whichever FRR stack is the current stack when the interrupt occurs. These routines clear various external FLIH recursion indicators, restore the previous (saved) FRR stack, and ABEND the program which received the interrupt.

RETRY Routine	Clears Indicator	Restores FRR Stack From
IEAVEE1R	PSAEXT	PSAESAV1
IEAVEE2R	LCCAXRC1	PSAESAV2
IEAVEE3R	LCCAXRC2	PSAESAV3

29 All three retry routines issue a PTLB instruction to ensure that there is no invalid information in the DAT translation buffers.

30 All three retry routines issue an ABEND macro with an X'3FC' completion code to terminate the program which was running when the external interrupt occurred.

Diagram SUP-11. Stage 1 Exit Effector (IEAVEF00) (Part 1 of 2)


From SVC IH to begin scheduling an asynchronous exit routine
Input
Output
Process
Branch entry
IGC043
IGC043BR
GETMAIN routine
RMBRANCH
Register 0
exit routine
Register 1
Option bits
Size of workarea
Register 1
IRB
72-byte problem program save area
IRB
RBPPSAV
Work area
For branch entries, to caller via branch
For SVC entries, to caller via exit prologue

1 Obtain storage for an IRB.

2 Obtain a work area if requested with the IRB.

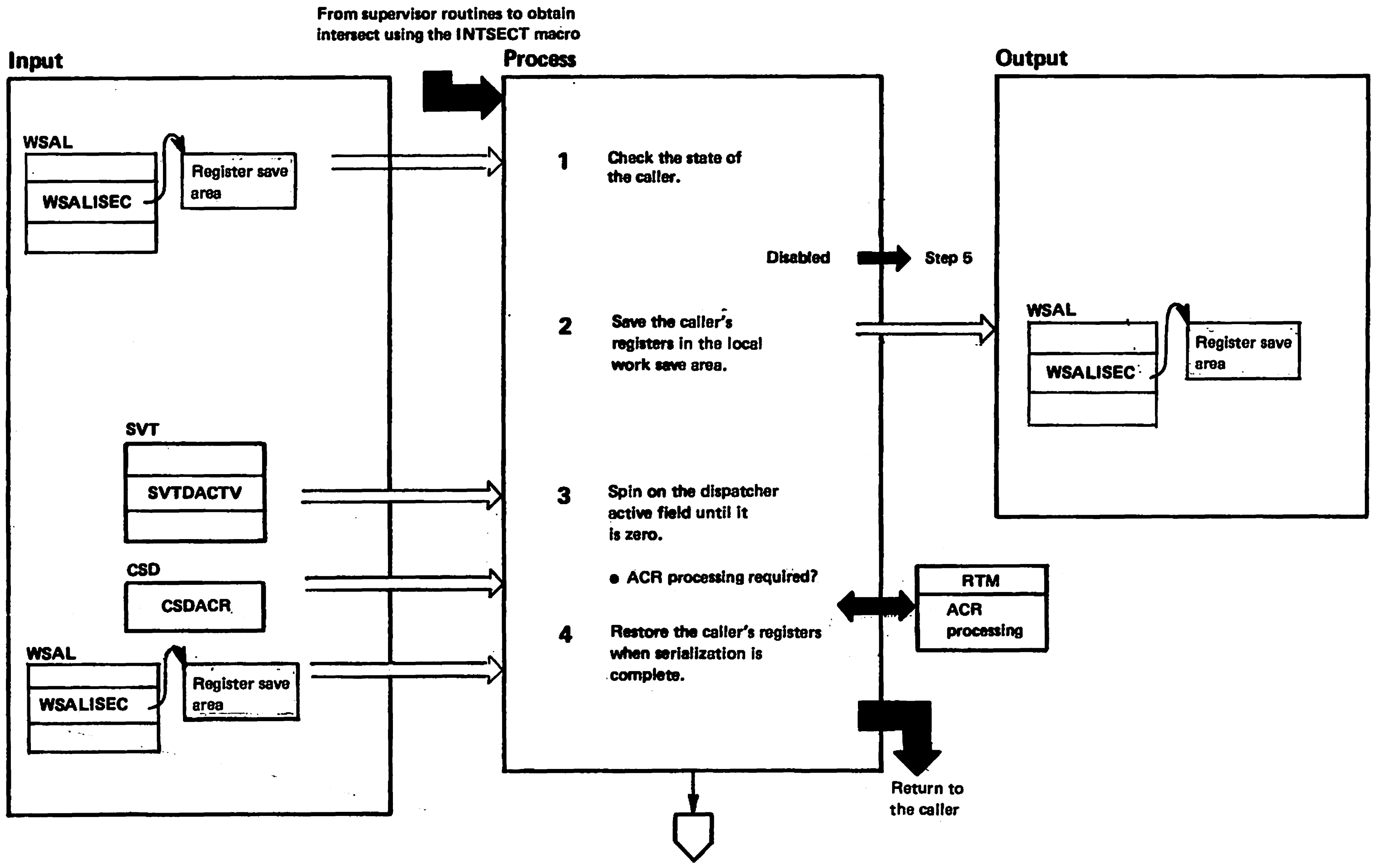
3 Obtain a save area if requested with the IRB.

4 Initialize the IRB.

Diagram SUP-11. Stage Exit Effector (IEAVEF00) (Part 2 of 2)

Extended Description	Module	Label
The stage 1 exit effector is called by supervisor or data management routines. Its purpose is to create and initialize, according to input parameters, an IRB (interruption request block) to control a user exit routine whose future use is requested by the caller.		
1 The stage 1 exit effector calls GETMAIN to obtain storage for the IRB from LSQA, subpool 253.	IEAVEF00	IGC043 IGC043BR
2 The caller may request a work area to be appended to the IRB. This work area will be released when the IRB is freed.		
3 Stage 1 exit effector obtains storage for the save area from the problem program's subpool 0, if requested.		
4 The information placed in the IRB during initialization includes the save area address, the entry point address and addressing mode of the user exit routine, the size of the RB, the PSW to be loaded to start execution of the asynchronous exit routine, and bits indicating whether the IRB should be freed by EXIT.		

Diagram SUP-12. INTERSECT Processing (IEAVEINT) (Part 1 of 6)



1 Check the state of the caller.

2 Save the caller's registers in the local work save area.

3 Spin on the dispatcher active field until it is zero.

- ACR processing required?

4 Restore the caller's registers when serialization is complete.

Diagram SUP-12. INTERSECT Processing (IEAVEINT) (Part 2 of 6)

Extended Description	Module	Label
The intersect function provides a means to serialize processing with the dispatcher. The INTSECT macro provides the interface to this function. If an invoker of the INTSECT macro was assembled with pre-MVS/XA macro libraries, the INTSECT macro passes control to the intersect module to ensure that the dispatcher has exited on all processors. Otherwise, the INTSECT macro makes an inline check of the dispatcher active field (SVTDACTV). If the dispatcher active field is non-zero, the INTSECT macro passes control to the intersect module which will wait for the dispatcher to exit each online processor.		
Two levels of intersect are provided:		
• Local		
The routine that requests the intersect must hold the local lock. The requesting routine then issues the INTSECT macro, which turns on the requestor's intersect bit in the local intersect field (ASCBSRQ). If the invoker was assembled with MVS/XA macros, the INTSECT macro checks the dispatcher active field (SVTDACTV) for zeros. If SVTDACTV is zero, indicating that the dispatcher is not active, the requesting routine continues processing. If SVTDACTV is non-zero, the macro branches to the intersect module at entry point IEAVEINL to perform the required serialization. If the invoker was assembled with pre-MVS/XA macro libraries, the INTSECT macro passes control to the intersect module at entry point IEAVEINT. Note that the pre - MVS/XA 4-byte SVTDACTV field is initialized to all X'FF's. This forces control to be passed to the service routine at IEAVEINT, which checks the new 16-byte SVTDACTV field.		
• Global		
The routine requesting the global intersect must hold the dispatcher lock. The macro turns on the requestor's intersect bit in the global intersect field (SVTDSREQ). If the invoker was assembled with MVS/XA macro libraries the INTSECT macro checks the dispatcher active field (SVTDACTV) for zeros. If SVTDACTV is zero, indicating that the dispatcher is not active, the requesting routine continues processing. If SVTDACTV is non-zero, the INTSECT macro branches to the intersect module (IEAVEINT) at entry point IEAVEING to perform the required serialization. If the invoker was assembled with pre - MVS/XA macros, the INTSECT macro passes control to the intersect module at entry point IEAVEINT.		
1 If the function was entered at the entry point IEAVEINT and the dispatcher active field is zero, IEAVEINT returns to the caller. IEAVEINT determines the state of the caller (either enabled or disabled). If disabled, the module continues processing at step 5.	IEAVEINT	IEAVEINT IEAVEINL IEAVEING
2 IEAVEINT saves the caller's registers 0 through 14 in the intersect local work save area. If the caller is enabled, IEAVEINT assumes that the local lock is held since the dispatcher lock is a disabled lock. The local lock serializes the local work save areas.		
3 The spin on the dispatcher active field (SVTDACTV) ensures serialization with the dispatcher in the following way. IEAVEINT examines the SVTDACTV one byte at a time. When one byte becomes zero, indicating the corresponding processor's dispatcher is not active, IEAVEINT examines the next byte. After all 16 bytes have become zero, proper serialization has been obtained and IEAVEINT returns to the module that issued the INTSECT macro.	IEAVEINT	SPINENBL
In order to ensure that the dispatcher does not interfere with an intersecting routine, the dispatcher checks the local intersect word (ASCBSRQ) for zero before searching the TCB queue in the address space. A nonzero ASCBSRQ indicates that the requesting routine owns the local intersect and that the dispatcher should not change anything in the address space. Therefore, if the ASCBSRQ is nonzero, the dispatcher will look at the next address space.		
4 The serialization process is now complete. The registers are reloaded from the local work save area, and control is passed to the caller.		

Diagram SUP-12. INTERSECT Processing (IEAVEINT) (Part 3 of 6)

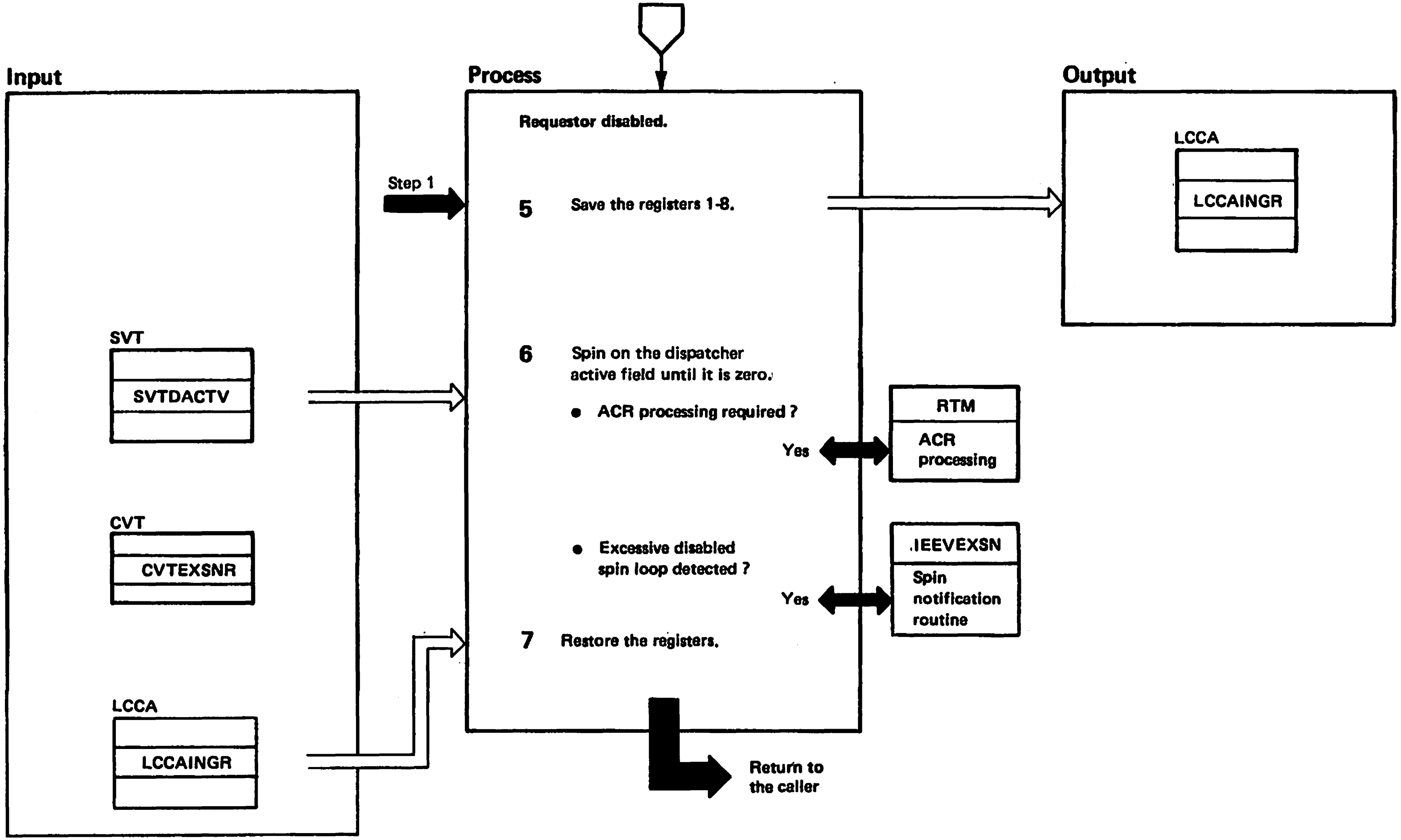
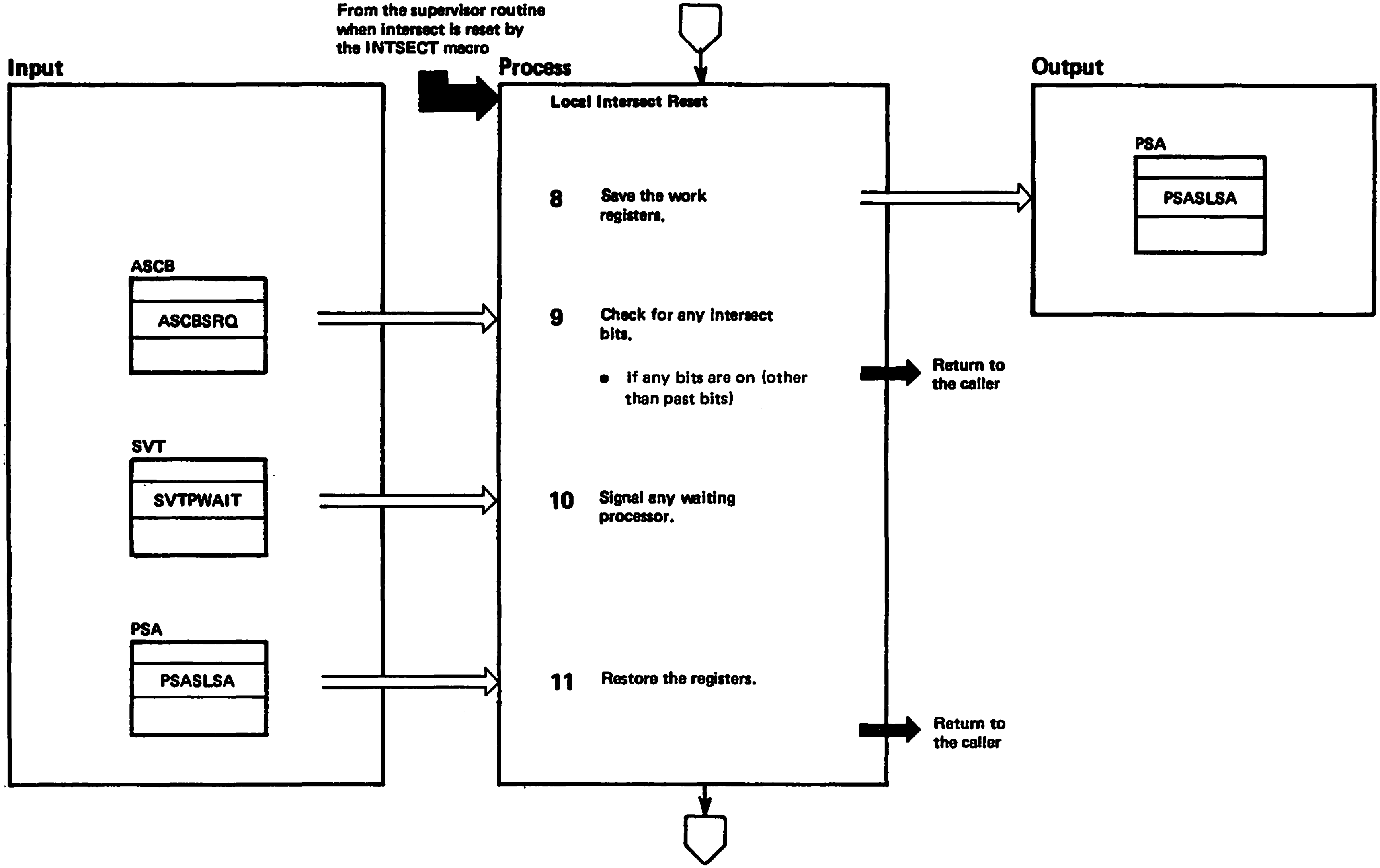

Input
Process
Output
Requestor disabled.
Step 1
5 Save the registers 1-8.
LCCA
LCCAINGR
SVT
SVTDACTV
6 Spin on the dispatcher active field until it is zero.
● ACR processing required ?
Yes
RTM
ACR processing
CVT
CVTEXSNR
● Excessive disabled spin loop detected ?
Yes
.IEEVEXSN
Spin notification routine
7 Restore the registers.
LCCA
LCCAINGR
Return to the caller

Diagram SUP-12. INTERSECT Processing (IEAVEINT) (Part 4 of 6)

Extended Description	Module	Label
Requestor Disabled		
5 IEAVEINT saves the caller's registers 1 through 8 in the LCCA intersect save area and establishes IEAVINTR as the functional recovery routine for the disabled spin processing described below. (See "Recovery Processing" at the end of this extended description for a description of IEAVINTR).	IEAVEINT	SPINDSBL
6 IEAVEINT examines the dispatcher active field (SVTDACTV) one byte at a time until the byte is zero. Since this is a disabled spin, the dispatcher issues the WINDOW macro to accept an emergency signal (EMS) or malfunction alert (MFA) and determines if ACR processing is required. If an ACR condition has occurred, IEAVEINT gives up control to RTM so that work on the dead processor can run. If the time spent in this disabled spin loop becomes greater than a global constant, IEAVEINT calls IEEVEXSN, the spin notification routine, to give the operator the option of initiating ACR.		
7 When all bytes of the dispatcher active field (SVTDACTV) have been tested and become zero, serialization is complete. IEAVEINT removes IEAVINTR from the FRR stack, reloads registers 1 through 8 from the LCCA intersect save area, and returns control to the caller.		

Diagram SUP-12. INTERSECT Processing (IEAVEINT) (Part 5 of 6)



Local Intersect Reset

8 Save the work registers.

9 Check for any intersect bits.

- If any bits are on (other than past bits)

10 Signal any waiting processor.

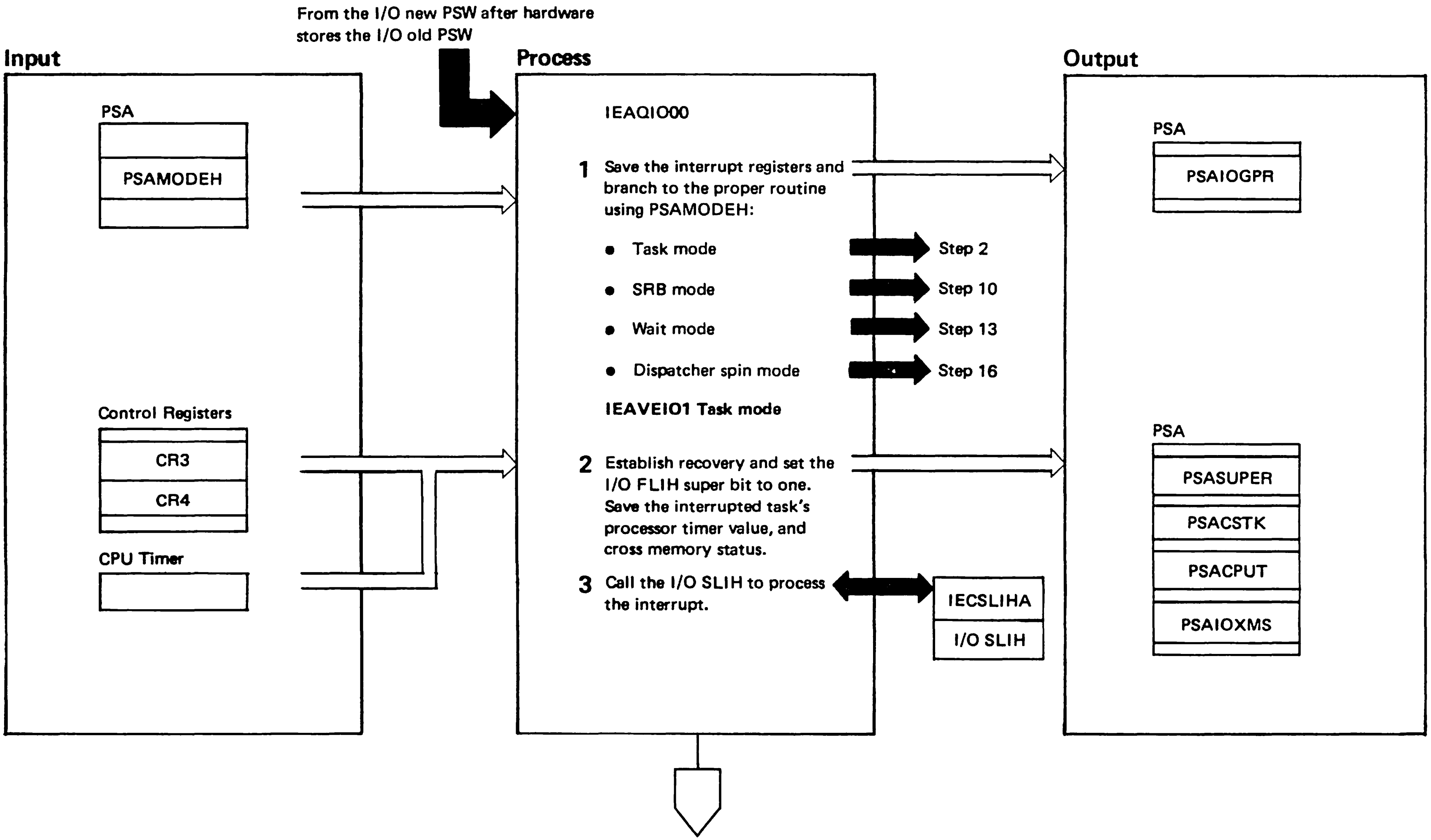
11 Restore the registers.

Diagram SUP-12. INTERSECT Processing (IEAVEINT) (Part 6 of 6)

Extended Description	Module	Label
Local Intersect Reset		
The intersect module is entered using the INTSECT macro on a local intersect reset when the post bits are on in the local intersect word (ASCBSRQ). The dispatcher turns on the post bits before dispatching the wait task when the local intersect is not available for an address space that might have work ready.		
Note: Post bits exist in the local intersect word (ASCBSRQ) and are set by the dispatcher to request notification when the local intersect becomes available.		
8 After disabling the processor, IEAVEINT saves the work registers in the PSA single level save area.		IEAVEINR
9 IEAVEINT examines the local intersect word (ASCBSRQ), ignoring|the post bits. If any othr intersect bits are on, the routine returns to the caller.		
10 If only the post bits are on, IEAVEINT signals waiting processors to enter the dispatcher to process any work that might now be ready. IEAVEINT uses the SVTPWAIT field to determine which processors are waiting. It uses a SIGP instruction to signal them.		
11 IEAVEINT restores the registers from the PSA single level save area, and returns control to the caller.		

Extended Description	Module	Label
Recovery Processing		
When an error occurs while IEAVEINT is in a disabled spin loop waiting for the dispatcher active field to go to zero, RTM gives control to IEAVINTR. IEAVINTR saves (in the variable recording area of the SDWA) the current-locks-held string (PSACLHS), the global intersect word (SVTDSREQ), the dispatcher active field (SVTDACTV), and the local intersect word (ASCBSRQ). If a restart interrupt caused the entry, IEAVINTR checks whether IEAVEINT was in a valid spin loop at the time of the interrupt. If it was, IEAVINTR issues a SETRP macro to return the ID of the processor that is keeping IEAVEINT in the spin (the processor with the non-zero dispatcher active byte). The macro also requests that the error be recorded in the SDWA. IEAVINTR then returns to RTM, and RTM signals the processor causing the spin via SIGP restart.	IEAVEINT	IEAVINTR
If IEAVEINT is not in a valid spin loop, IEAVINTR issues an SETRP macro requesting an SDUMP, error recording, and percolation.		

Diagram SUP-13. I/O Interrupt Handler (IEAVEIO) (Part 1 of 14)



IEAQIO00

1 Save the interrupt registers and branch to the proper routine using PSAMODEH:

- Task mode
- SRB mode
- Wait mode
- Dispatcher spin mode

IEAVEIO1 Task mode

2 Establish recovery and set the I/O FLIH super bit to one. Save the interrupted task's processor timer value, and cross memory status.

3 Call the I/O SLIH to process the interrupt.

Diagram SUP-13. I/O Interrupt Handler (IEAVEIO) (Part 10 of 14)

Extended Description	Module	Label
13 When the interrupt occurs while the system is in wait mode, the I/O FLIH:	IEAVEIO	IEAVEI03

- Sets the I/O FLIH super bit in the PSASUPER.
- Saves the CPU timer in the PSACPUT.

14 The I/O FLIH accumulates the wait time by calculating the amount of time in wait state and adding it to the LCCAWTIM. It also sets to zero the dispatcher active (SVTDACTV) and processor waiting (SVTPWAIT) fields.

15 The I/O FLIH:

- Sets the return address so that the I/O SLIH will return directly to the dispatcher at entry point IEAPDS7B.
- Passes control to the I/O SLIH.

16 When the interrupt occurs while the system is in dispatcher mode, the I/O FLIH:

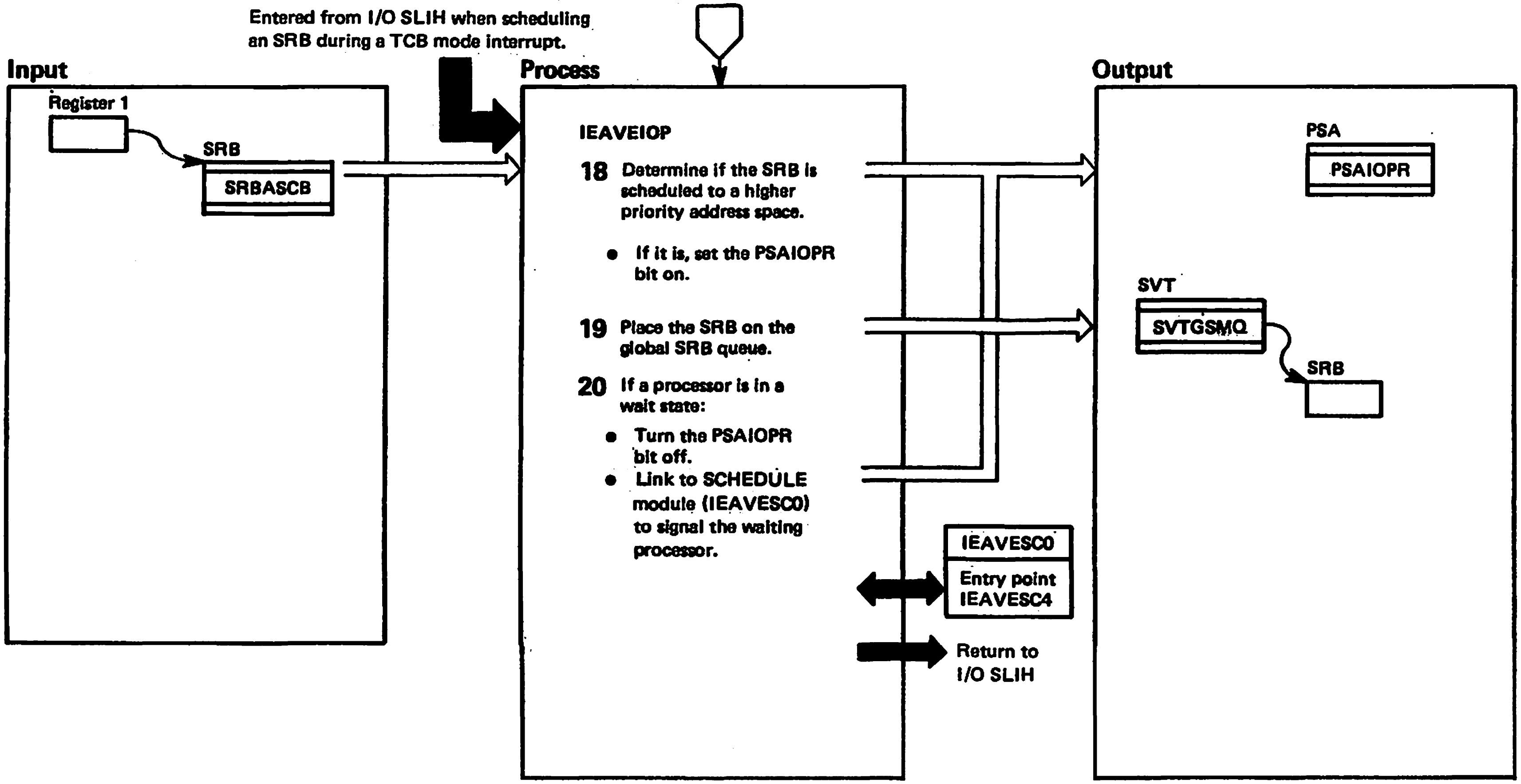
IEAVEI04

- Sets the I/O FLIH super bit in PSASUPER.
- Sets to zero the dispatcher active byte (SVTDACTV) and the processor waiting byte (SVTPWAIT).

17 The I/O FLIH:

- Sets the return address so that the I/O SLIH will return directly to the dispatcher at entry point IEAPDS7B.
- Passes control to the I/O SLIH.

Diagram SUP-13. I/O Interrupt Handler (IEAVEIO) (Part 11 of 14)



IEAVEIOP

18 Determine if the SRB is scheduled to a higher priority address space.

- If it is, set the PSAIOPR bit on.

19 Place the SRB on the global SRB queue.

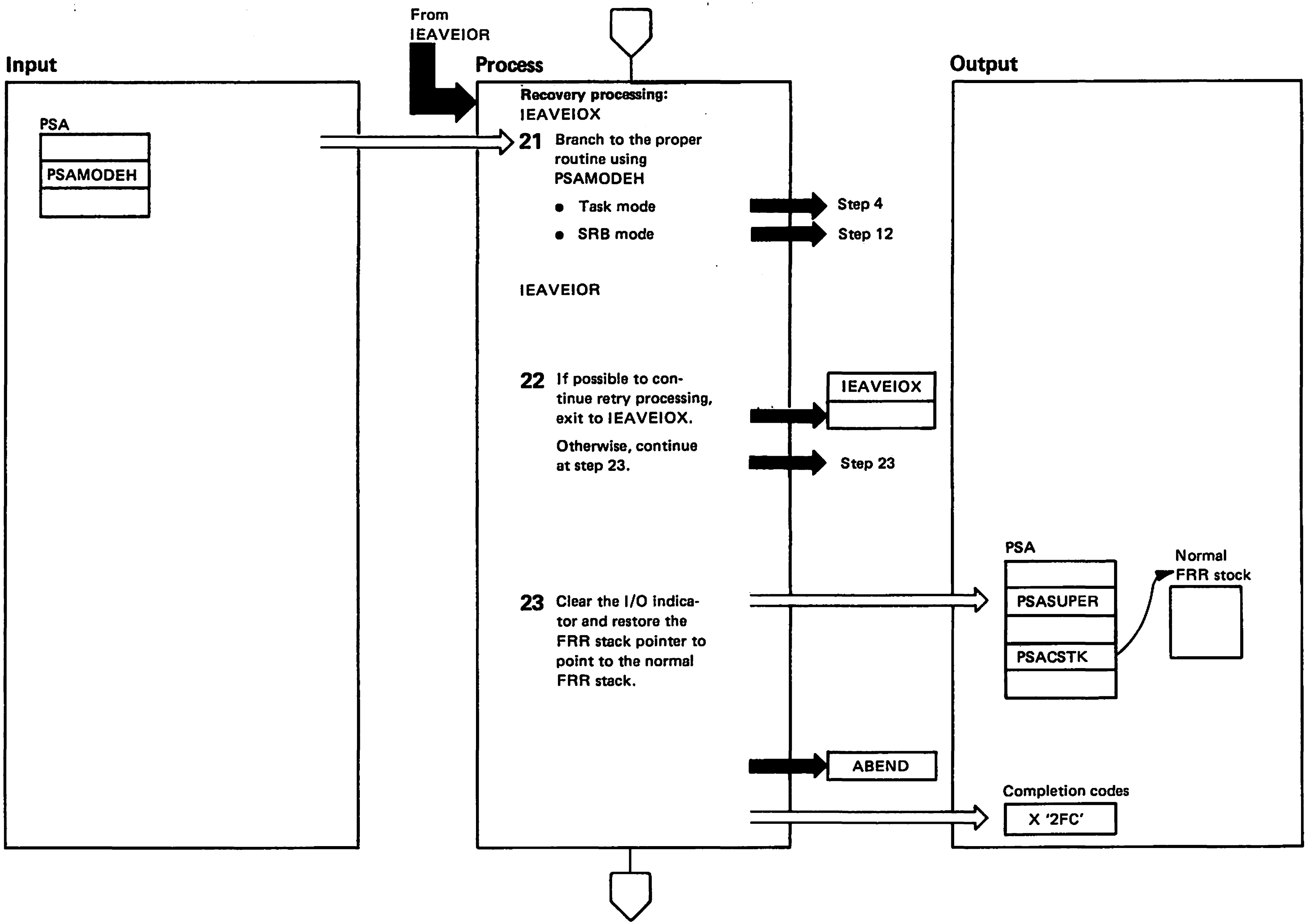
20 If a processor is in a wait state:

- Turn the PSAIOPR bit off.
- Link to SCHEDULE module (IEAVESC0) to signal the waiting processor.

Diagram SUP-13. I/O Interrupt Handler (IEAVEIO) (Part 12 of 14)

Extended Description	Module	Label
18 When the interrupt occurs while the system is in TCB mode, the I/O FLIH determines if an SRB is scheduled to a higher priority address space by comparing the dispatching priorities of the SRBs target address space and the home address space. If the target address space is a higher priority, the I/O FLIH sets the PSAIOPR bit.		IEAVEIOP
If the interrupted task and the task associated with the SRB is for the same address space, the PSAIOPR bit is also set, since the relative priorities of the tasks are unknown.		
19 The I/O FLIH places the SRB on the global SRB queue.		
20 If there is a processor in a wait state, the I/O FLIH resets the PSAIOPR bit to zero and calls the schedule module (IEAVESCO) to signal the waiting processor.	IEAVESCO	IEAVESC4
The I/O FLIH then returns to IOS.		

Diagram SUP-13. I/O Interrupt Handler (IEAVEIO) (Part 13 of 14)



Recovery processing:
IEAVEIOX

21 Branch to the proper routine using PSAMODEH

- Task mode
- SRB mode

IEAVEIOR

22 If possible to continue retry processing, exit to IEAVEIOX.

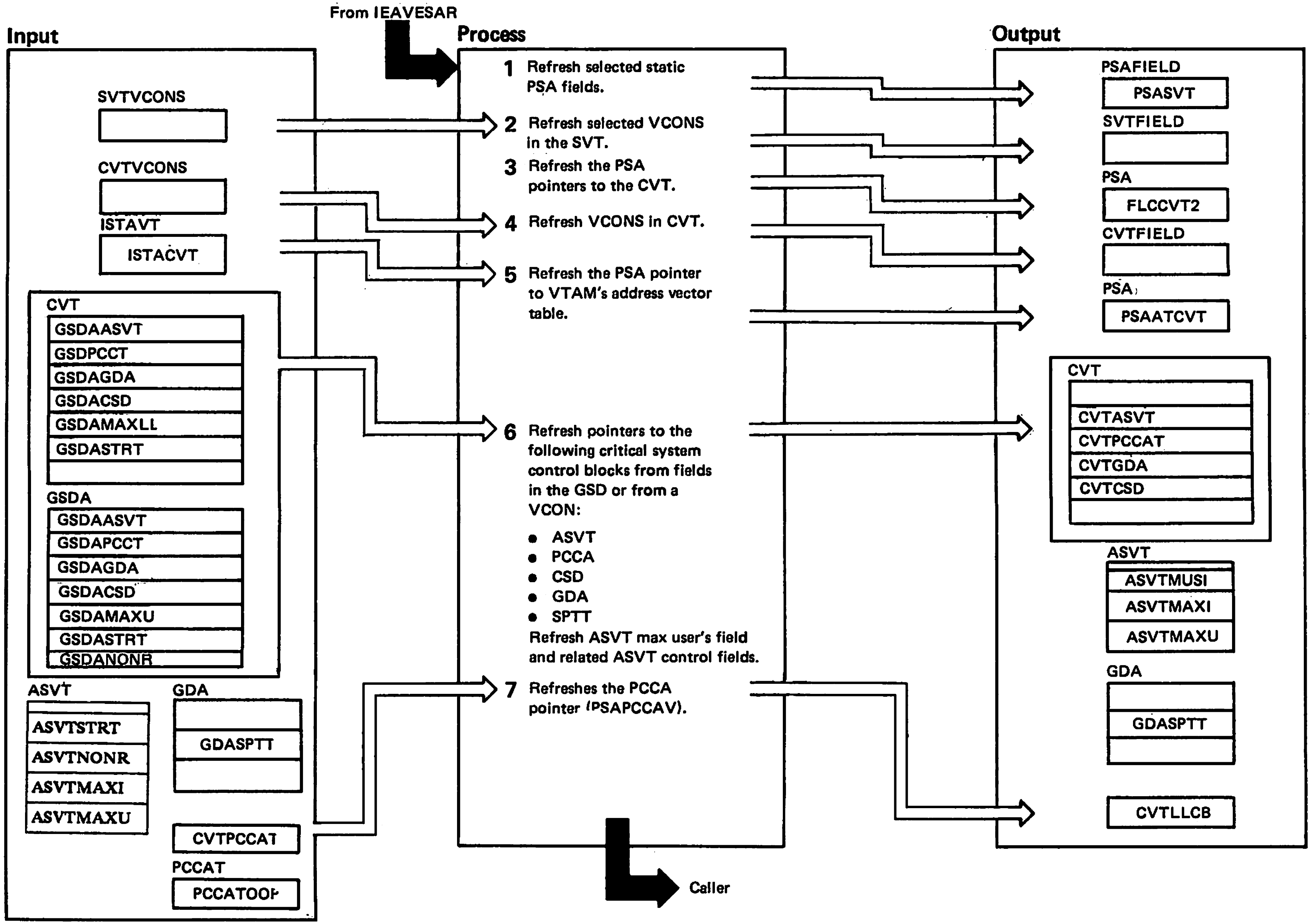
Otherwise, continue at step 23.

23 Clear the I/O indicator and restore the FRR stack pointer to point to the normal FRR stack.

Diagram SUP-13. I/O Interrupt Handler (IEAVEIO) (Part 14 of 14)

Extended Description	Module	Label
Recovery Processing:		
21 Control is received when the I/O FLIH recovery routine (IEAVEIOR) indicates to continue retry processing after an error. The system mode determines which routine is to continue retry processing. IEAVEIO uses PSAMODEH as an index to the internal table IORETTAB to reference the proper routine.		IEAVEIOX
22 If the error occurred when the I/O SLIH was invokea to process a task mode or SRB mode I/O interrupt, resume normal processing by exiting to entry point IEAVEIOX of IEAVIO.		IEAVEIOR
23 The I/O FHIL FRR (IEAVEIOR) clears the I/O FLIH super bit in PSASUPER and points the FRR stacker pointer (PSACSTK) in the PSA to the normal FRR stack. It then ABENDS the interrupted program with a X'2FC' completion code.		

Diagram SUP-14. Low Storage Refresh (IEAVELCR) (Part 1 of 2)



1 Refresh selected static PSA fields.

2 Refresh selected VCONS in the SVT.

3 Refresh the PSA pointers to the CVT.

4 Refresh VCONS in CVT.

5 Refresh the PSA pointer to VTAM's address vector table.

6 Refresh pointers to the following critical system control blocks from fields in the GSD or from a VCON:

- ASVT
- PCCA
- CSD
- GDA
- SPTT

Refresh ASVT max user's field and related ASVT control fields.

7 Refreshes the PCCA pointer (PSAPCCAV).

Diagram SUP-14. Low-Storage Refresh (IEAVELCR) (Part 2 of 2)

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

IEAVELCR checks, and if necessary, refreshes critical fields in the PSA, SVT, CVT, ASVT, and the GDA.

1 Refreshes static fields in the PSA.

2 Refreshes the critical values in the SVT.

3 Refreshes the PSA secondary pointer to the CVT (FLCCVT2).

4 Refreshes the VCONs in the CVT.

5 Refreshes the PSA pointer to VTAM's CVT (PSAATCVT) from VTAM's vector table (AVT).

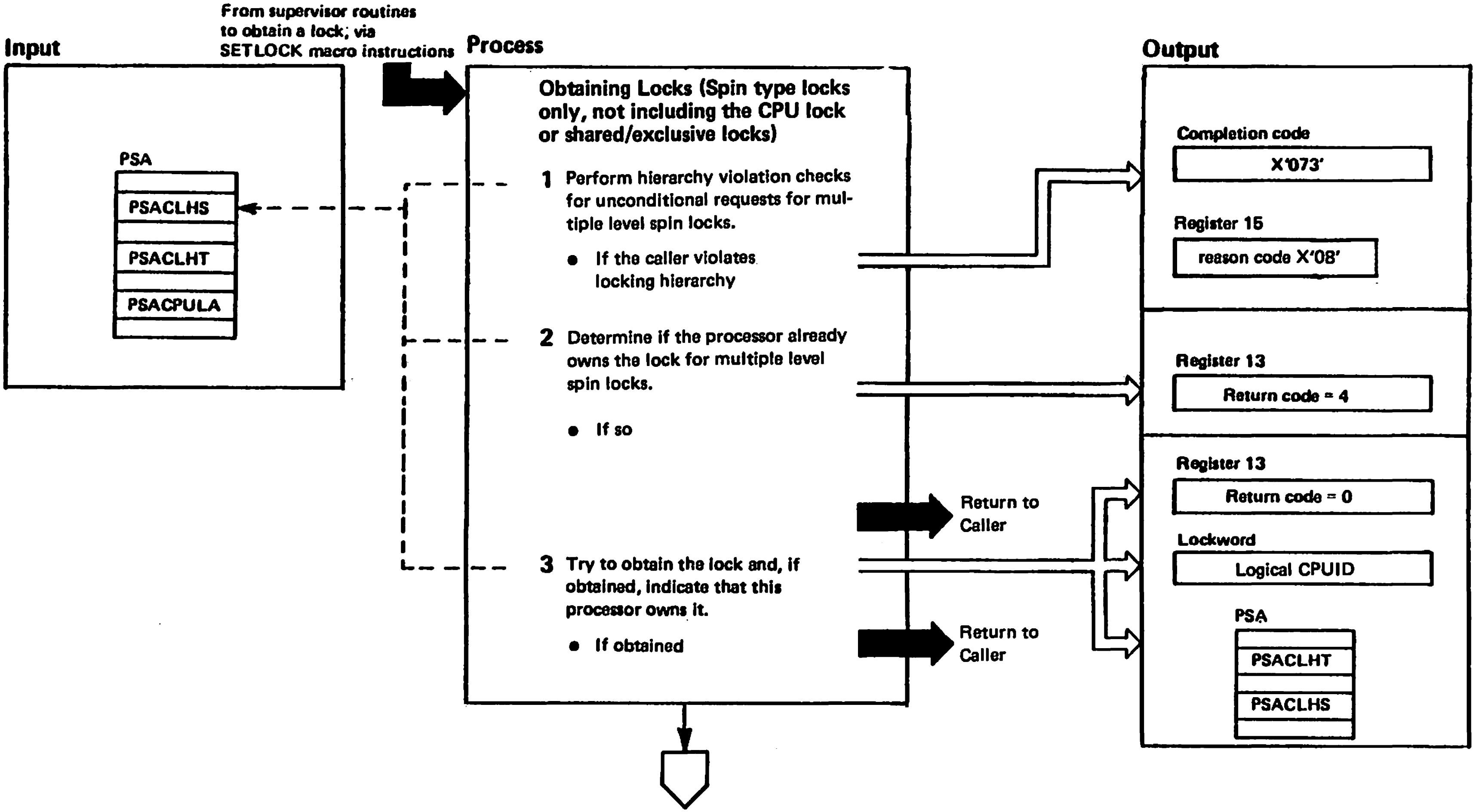
6 Refreshes pointers to critical system control blocks. These control blocks are the ASVT, PCCA, CSD, and the GDA, The ASVT max user's field and related fields are recalculated and refreshed:

- ASVTMAXI - The installation specified max user's count.
- ASVTNONR - The number of ASVT entries reserved to replace non-reusable ASIDs.
- ASVTSTRT - The number of ASVT entries reserved for STARTed/SASI address spaces.
- ASVTMAXU - The installation specified max user's count plus the number of ASVT entries reserved to place non-reusable ASIDs and the number of ASVT entries reserved for STARTed/SASI address spaces.

These pointers and fields are checked against write protected fields in the GSDA. The SPTT (subpool translator table pointer) is refreshed via a VCON.

7 Refreshes the pointer to the PCCA (PSAPCCAV). The alternate path to the PCCA is CVTPCCAT→PCCAT00P→PCCA. If both the PSAPCCAV and the PCCAT00P for this processor fail LRA checks, then this processor is set to a disabled wait with a PSW wait code X'083'.

Diagram SUP-15. Spin Lock Manager Processing (IEAVELK) (Part 1 of 8)



Obtaining Locks (Spin type locks only, not including the CPU lock or shared/exclusive locks)

1 Perform hierarchy violation checks for unconditional requests for multiple level spin locks.

- If the caller violates locking hierarchy

2 Determine if the processor already owns the lock for multiple level spin locks.

- If so

3 Try to obtain the lock and, if obtained, indicate that this processor owns it.

- If obtained

Diagram SUP-15. Spin Lock Manager Processing (IEAVELK) (Part 2 of 8)

Extended Description	Module	Label
The spin lock manager provides the means for a user to obtain locks that serialize the use of a resource. The lock manager provides the following locks:	IEAVELK	

- RSMGL (RSM global lock)
- VSMFIX (VSM fixed subpools lock)
- ASM (auxiliary storage management lock)
- ASMGL (ASM global lock)
- RSMST (RSM steal lock)
- RSMCM (RSM common lock)
- RSMXM (RSM cross memory lock)
- RSMAD (RSM address space lock)
- RSM (RSM lock, shared/exclusive)
- VSMPAG (VSM pageable subpools lock)
- DISP (dispatcher lock)
- SALLOC (space allocation lock)
- IOSYNCH (IOS synchronization lock)
- IOSUCB (IOS unit control block lock)
- SRM (the system resource management lock)
- TRACE (TRACE lock, shared/exclusive)
- CPU (processor lock)

The lock manager both obtains and releases locks. There are two distinct methods of obtaining locks; conditionally and unconditionally. If the lock cannot be obtained for a conditional request, the lock manager immediately returns control to the caller with the appropriate return code in register 13. If an unconditional request for a spin lock cannot be satisfied, the lock manager keeps control until the lock is obtained.

Note:
For non-class spin locks, step 3 is performed prior to steps 1 and 2.

Extended Description	Module	Label
1 If an unconditional request for a multiple level spin lock is presented, the lock manager determines whether the caller has violated the locking hierarchy by:	IEAVELK	

- Unconditionally requesting a lock lower in the hierarchy than a lock it already holds.
- Requesting a class lock when another lock in that class is already held.

The lock manager abnormally terminates callers who violate the hierarchy with a X'073' completion code and a reason code of X'08' in register 15.

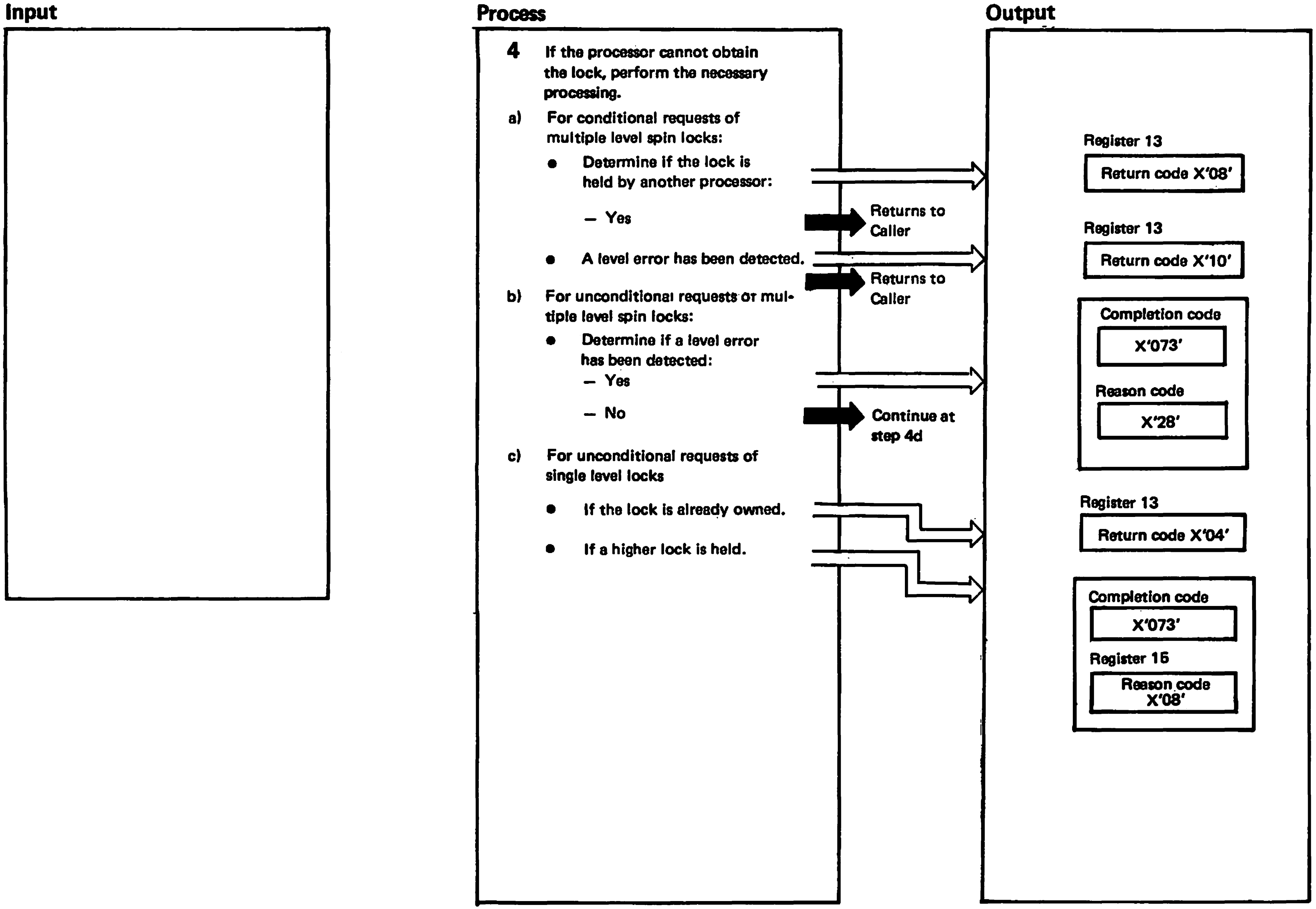
2 The lock manager determines whether this processor already owns the requested lock. If this processor owns it, the lock manager puts a return code of 4 in register 13, and returns control to the caller. Otherwise, processing continues.

3 The lock manager tries to obtain the lock. If the lock is available (the lockword contains 0), the lock manager indicates ownership by:

- Placing into the lockword the logical processor ID.
- Setting the appropriate bit in the processor-locks-held string (PSACLHS)
- For any class lock, storing the address of the lockword into the processor-locks-held table (PSACLHT)

The lock manager then returns to the caller with a zero return code. If the lock is not available, processing continues at step 4.

Diagram SUP-15. Spin Lock Manager Processing (IEAVELK) (Part 3 of 8)



4 If the processor cannot obtain the lock, perform the necessary processing.

a) For conditional requests of multiple level spin locks:

- Determine if the lock is held by another processor:
  - Yes
- A level error has been detected.

b) For unconditional requests of multiple level spin locks:

- Determine if a level error has been detected:
  - Yes
  - No

c) For unconditional requests of single level locks

- If the lock is already owned.
- If a higher lock is held.

Diagram SUP-15. Spin Lock Manager Processing (IEAVELK) (Part 4 of 8)

Extended Description	Module	Label	Extended Description	Module	Label

4

- For conditional requests, IEAVELK sets one of the following return codes:

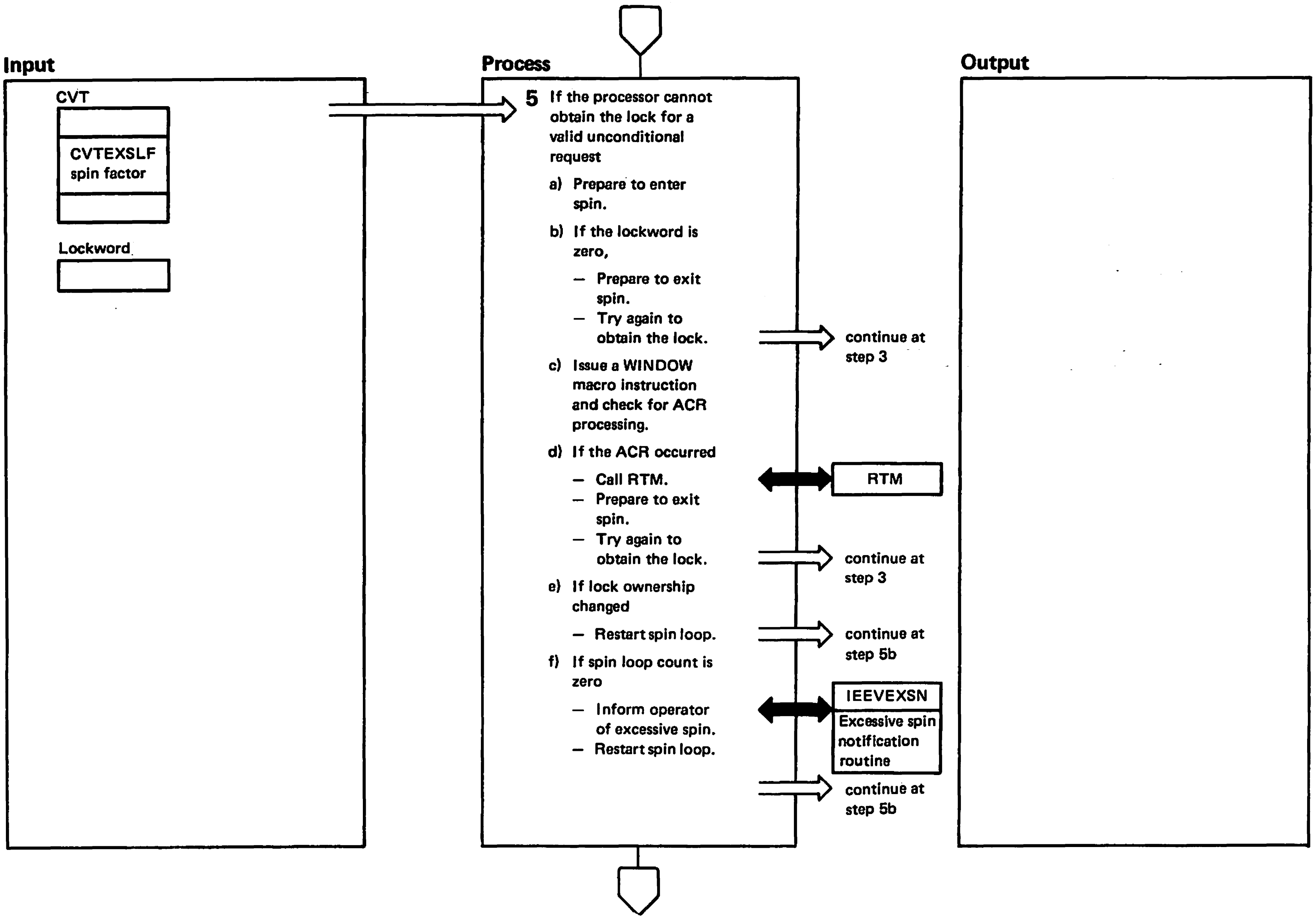
  X '04' – Lock already owned by caller.
  X '08' – Lock held by another processor.
  X'10' – Level error detected. (Level errors are only defined for multiple level locks.)

- For unconditional requests for multiple level spin locks, IEAVELK determines if a level error has occurrod. A level error indicates that:

- The caller is attempting to use the same lockword to represent two distinct locks. This is detected if the PSACLHT slot for the lock requested is zero, the lockword contains the current CPU ID, and another PSACLHT slot contains the user-supplied lockword address.
- The caller is attempting to use two different lockwords for the same lock. This is detected if the PSACLHS indicates the lock is held but the PSACLHT slot contains a lockword address different than the user-supplied lockword address.

If a level error has been detected, the lock manager abnormally terminates the lock request with a completion code of X'073' and a reason code of X '28' in register 15, if the request was for unconditional ownership. A return code of X'10' is generated for level errors detected on conditional requests.

Diagram SUP-15. Spin Lock Manager Processing (IEAVELK) (Part 5 of 8)



5 If the processor cannot obtain the lock for a valid unconditional request

a) Prepare to enter spin.

b) If the lockword is zero,

- Prepare to exit spin.
- Try again to obtain the lock.

continue at step 3

c) Issue a WINDOW macro instruction and check for ACR processing.

d) If the ACR occurred

- Call RTM.
- Prepare to exit spin.
- Try again to obtain the lock.

continue at step 3

e) If lock ownership changed

- Restart spin loop.

continue at step 5b

f) If spin loop count is zero

- Inform operator of excessive spin.
- Restart spin loop.

continue at step 5b

Diagram SUP-15. Spin Lock Manager Processing (IEAVELK) (Part 6 of 8)

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

5 If the processor cannot obtain the lock for a valid unconditional request, the lock manager loops ("spins") to check if the lockword is zero.

a) IEAVELK issues a SETFRR macro to establish ELKFRR as the functional recovery routine (FRR), turns on the lock manager super bit, and calculates the spin loop time out count.

b) The lockword is checked again to see if it is zero. If it is, IEAVELK turns off the super bit, issues the SETFRR macro to delete the FRR, and resumes processing at step 3, where it tries to obtain the lock again.

c) The lock manager issues a WINDOW macro to enable EMS (emergency signal) and MFA (malfunction alert) interruptions. (This is done to prevent deadlock in case of failure on the other processor.)

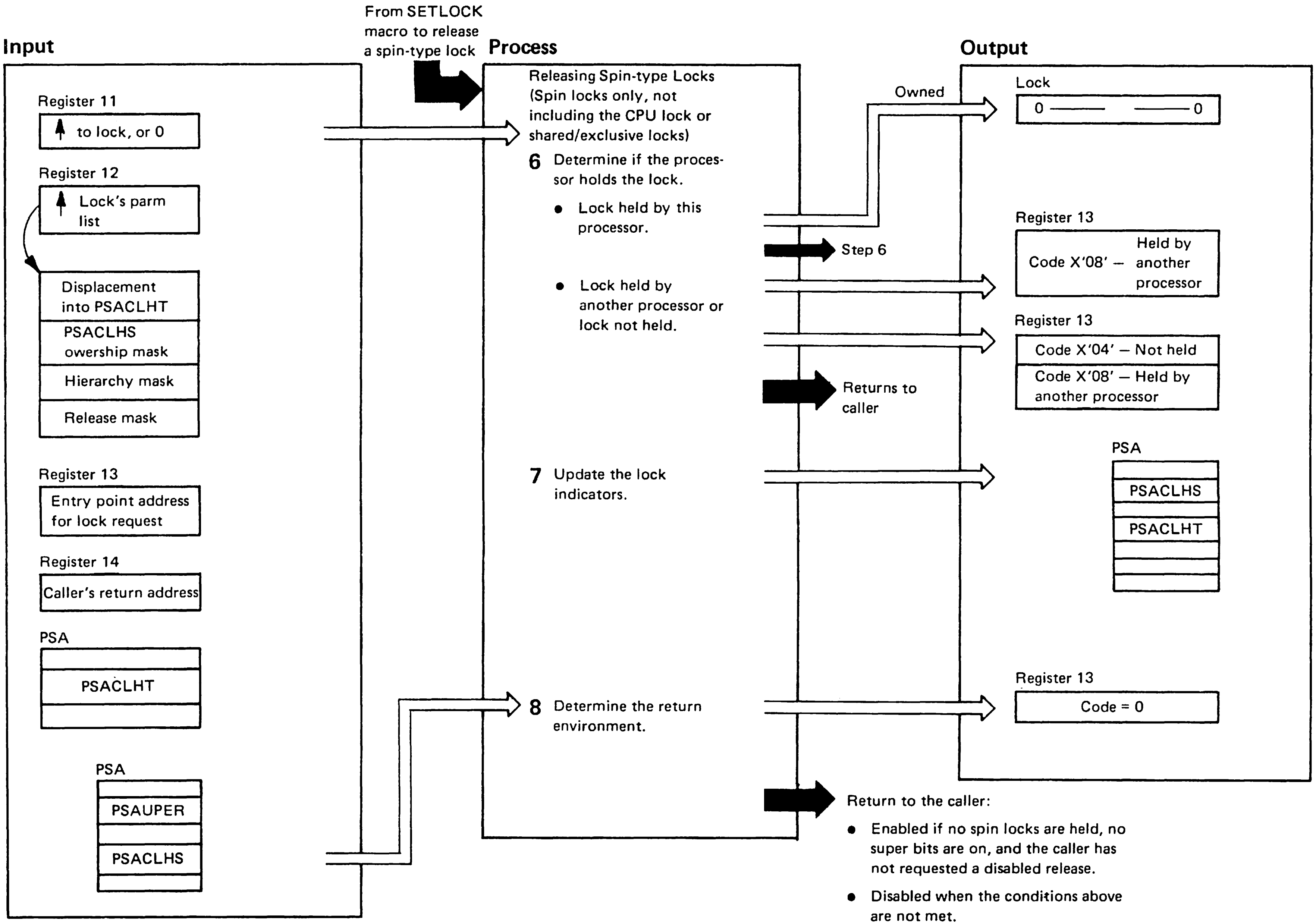
d) The lock manager then determines if an ACR (alternate CPU recovery) condition occurred. If so, it returns control to RTM and turns off the super bit, issues the SETFRR macro to delete the FRR, and resumes processing at step 3, where it tries to obtain the spin lock again.

e) If lock ownership has changed, lock manager restarts the spin at step 5b, resetting the spin loop time count.

f) If the spin loop count is zero, the lock manager calls IEEVEXSN to inform the operator of the excessive spin and restarts the spin loop at step 5a by resetting the spin loop time count.

Otherwise, IEAVELK goes to step 5b to check the lockword and repeats the loop.

Diagram SUP-15. Spin Lock Manager Processing (IEAVELK) (Part 7 of 8)



Releasing Spin-type Locks (Spin locks only, not including the CPU lock or shared/exclusive locks)

6 Determine if the processor holds the lock.

- Lock held by this processor.
- Lock held by another processor or lock not held.

7 Update the lock indicators.

8 Determine the return environment.

Return to the caller:

- Enabled if no spin locks are held, no super bits are on, and the caller has not requested a disabled release.
- Disabled when the conditions above are not met.

Diagram SUP-15. Spin Lock Manager Processing (IEAVELK) (Part 8 of 8)

Extended Description	Module	Label

6 The lock manager releases the locks when the caller issues the SETLOCK macro using the RELEASE operand.

IEAVELK determines if the lock is held by this processor.

- If the processor owns the lockword, the lock manager releases it by setting the lockword to zeros.
- If the lock is not held by this processor, IEAVELK returns to the caller with a return code in register 13. If no processor holds the lock, the return code equals X'04'; if another processor owns the lock, it equals X'08'.

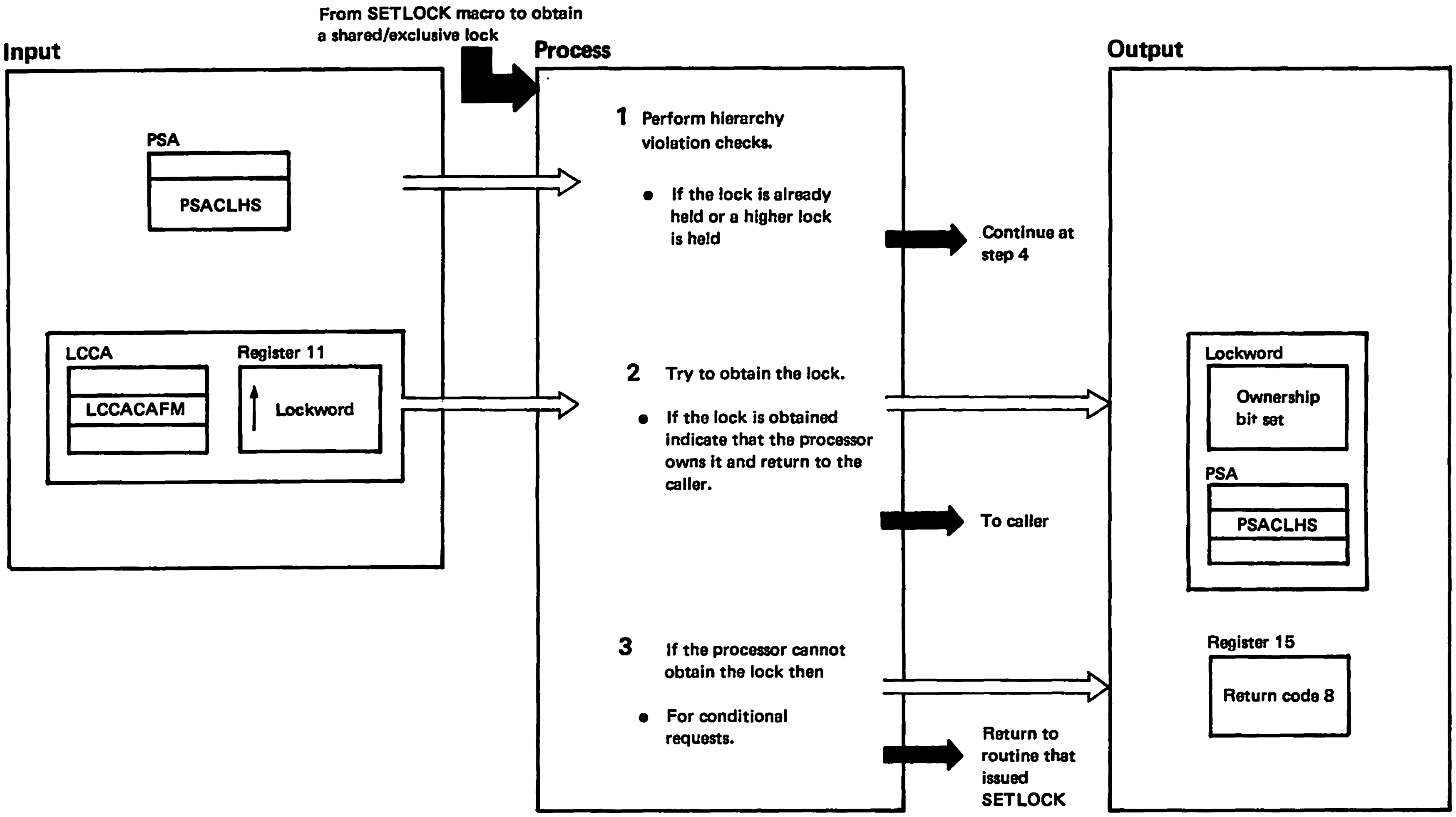
7 IEAVELK updates the lock indicators by clearing the appropriate bit in the locks-held string. If this is a class lock, IEAVELK clears the appropriate entry in the locks-held table.

8 If any one of the following conditions is met, IEAVELK returns to the caller in a disabled state.

- A spin lock is held.
- A super bit is set.
- The caller requested control be returned in a disabled state.

Otherwise, IEAVELK returns in an enabled state. In either case, the return code is zero.

Diagram SUP-16. Obtaining Shared/Exclusive Locks (IEAVELK) (Part 1 of 6)



From SETLOCK macro to obtain a shared/exclusive lock

Input

PSA

PSACLHS

LCCA

LCCACAFM

Register 11

Lockword

Process

1 Perform hierarchy violation checks.

- If the lock is already held or a higher lock is held

Continue at step 4

2 Try to obtain the lock.

- If the lock is obtained indicate that the processor owns it and return to the caller.

To caller

3 If the processor cannot obtain the lock then

- For conditional requests.

Return to routine that issued SETLOCK

Output

Lockword

Ownership bit set

PSA

PSACLHS

Register 15

Return code 8

Diagram SUP-16. Obtaining Shared/Exclusive Locks (IEAVELK) (Part 2 of 6)

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

1 The lock manager determines if the lock requested is already held by the caller or if a higher lock in the locking hierarchy is already held. If so, continue at step 4.

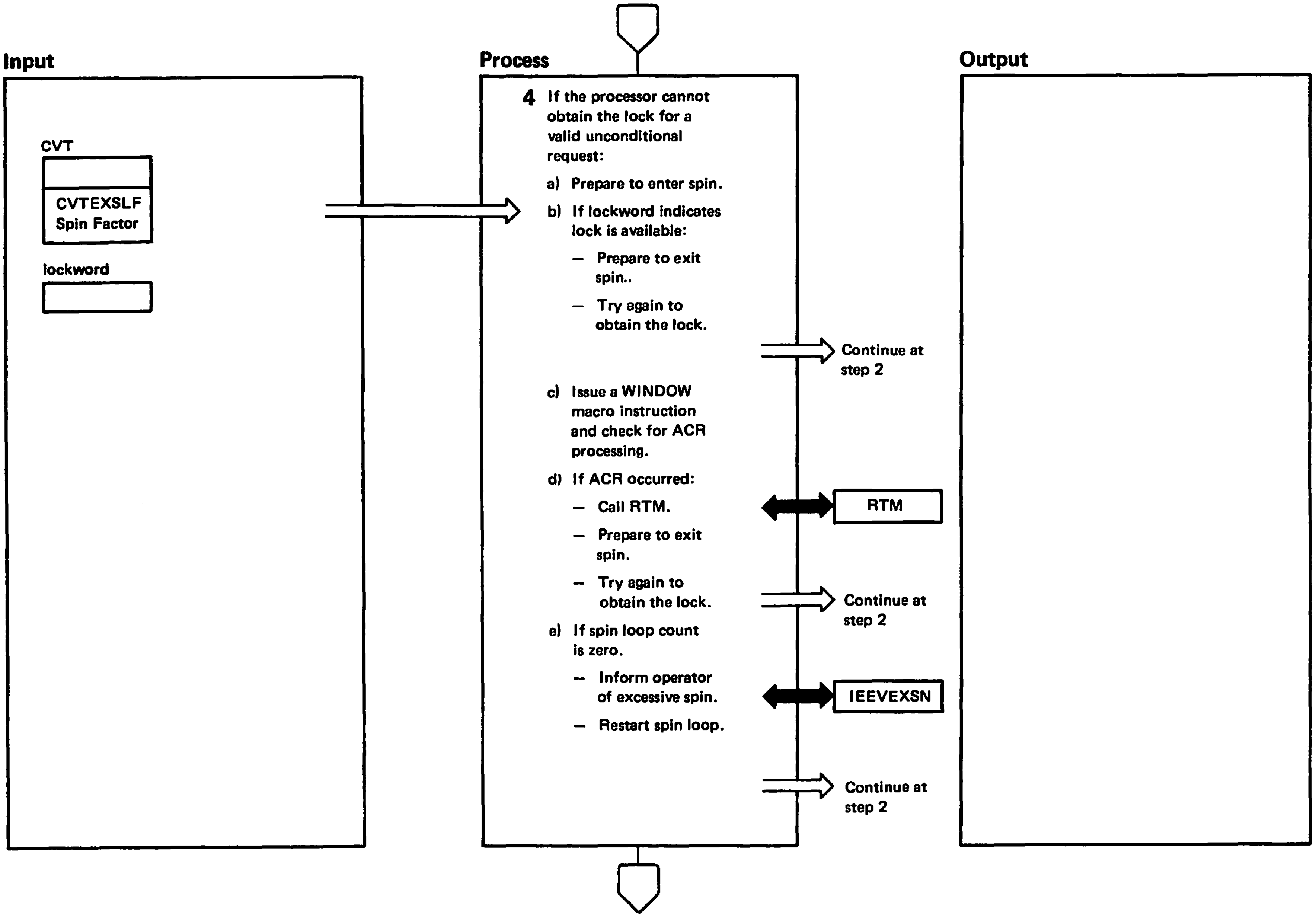
2 The lock manager tries to obtain the lock. If the lockword is available, then: For shared requests, the exclusive ownership and exclusive pending bits 0 and 1 of the lockword must be zero; for exclusive requests, the lockword must be zero. Ownership of the lock is indicated as follows:

- If the lock is obtained shared the lock manager turns on the appropriate bit reflecting the requesting processor. (Bits 16-31 of the lockword represent CPU 0 through F, respectively).

- If the lock is obtained exclusive, the lock manager sets the appropriate bit in the low order two bytes. Bit 0 of the lockword is set to one to indicate exclusive ownership.

3 For conditional requests, IEAVELK returns a return code of 8 in register 13.

Diagram SUP-16. Obtaining Shared/Exclusive Locks (IEAVELK) (Part 3 of 6)



4 If the processor cannot obtain the lock for a valid unconditional request:

a) Prepare to enter spin.

b) If lockword indicates lock is available:

- Prepare to exit spin..
- Try again to obtain the lock.

Continue at step 2

c) Issue a WINDOW macro instruction and check for ACR processing.

d) If ACR occurred:

- Call RTM.
- Prepare to exit spin.
- Try again to obtain the lock.

Continue at step 2

e) If spin loop count is zero.

- Inform operator of excessive spin.
- Restart spin loop.

Continue at step 2

Extended Description	Module	Label

4

- If a shared lock is requested, the lock manager considers the lock available when the exclusive ownership bit and the exclusive pending bit (bits 0 and 1 of the lockword, respectively) become zero.
- If exclusive ownership is requested, the lock manager considers the lock available when either the lockword becomes zero or the only bit on in the lockword is the exclusive pending bit.
- If the processor cannot obtain the lock for a valid unconditional request, the lock manager loops ("spins") to check if the lockword is zero.

a) IEAVELK issues a SETFRR macro to establish ELKFRR as the functions recovery routine (FRR), turns on the lock manager super bit, and calculates the spin loop time-out count .

b) The lockword is checked again to see if it is zero. If it is, IEAVELK turns off the super bit, issues the SETFRR macro to delete the FRR, and resumes processing at step 2, where it tries to obtain the lock again.

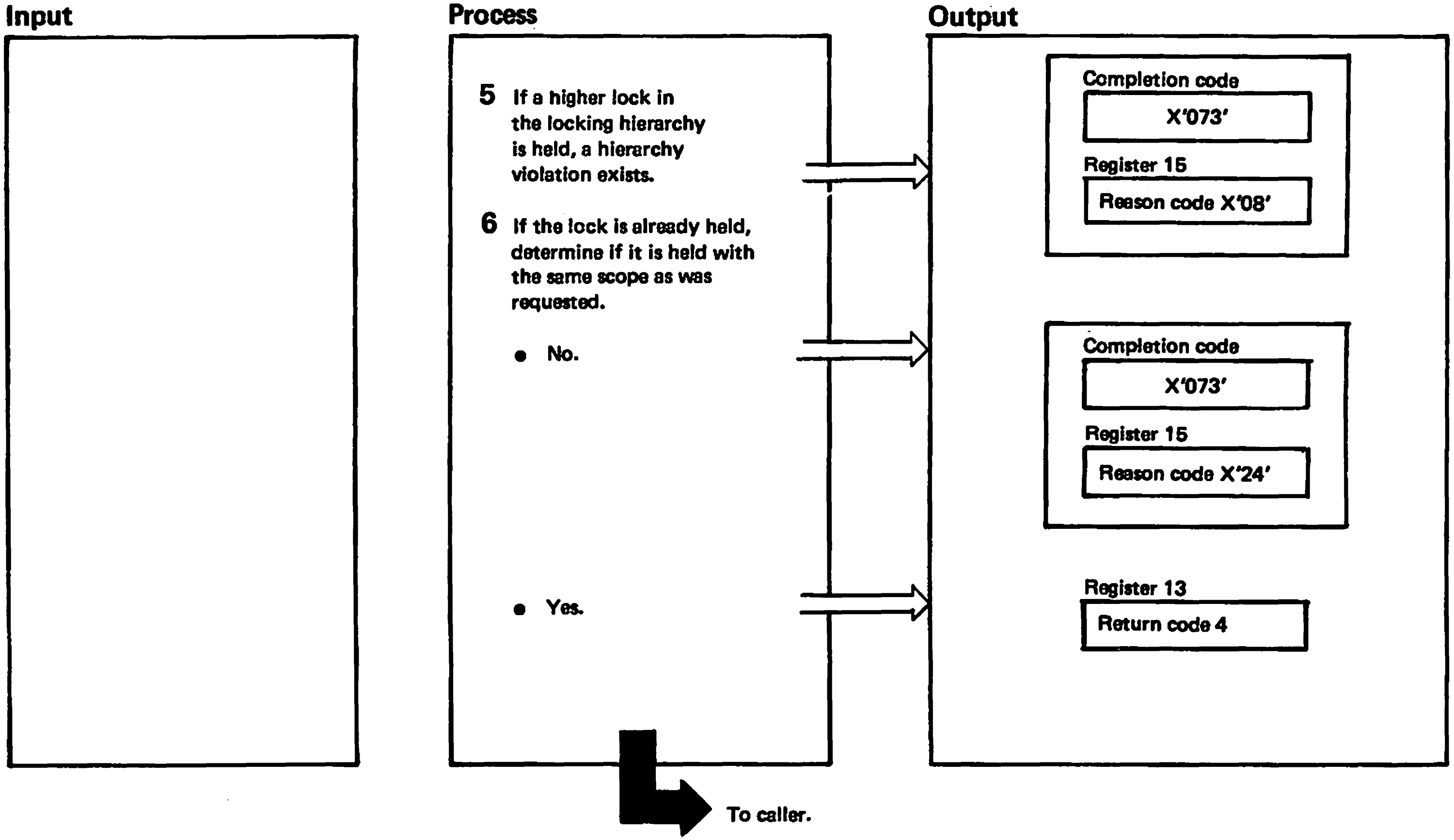
c) The lock manager issues a WINDOW macro to enable EMS (emergency signal) and MFA (malfunction alert) interruptions. (This is done to prevent deadlock in case of failure on the other processor.)

d) The lock manager then determines if an ACR (alternate CPU recovery) condition occurred. If so, returns control to RTM and turns off the super bit. Issues the SETFRR macro to delete the FRR, and resumes processing at step 2, where it tries to obtain the spin lock again.

e) If the spin loop count is zero, the lock manager calls IEEVEXSN to inform the operator of the excessive spin and restarts the spin loop at step 4a by resetting the spin loop time count.

Otherwise, IEAVELK goes to step 4b to check the lockword and repeats the loop.

Diagram SUP-16. Obtaining Shared/Exclusive Locks (IEAVELK) (Part 5 of 6)



Input

Process

5 If a higher lock in the locking hierarchy is held, a hierarchy violation exists.

6 If the lock is already held, determine if it is held with the same scope as was requested.

- No.
- Yes.

To caller.

Output

Completion code X'073'
Register 15 Reason code X'08'

Completion code X'073'
Register 15 Reason code X'24'

Register 13 Return code 4

Diagram SUP-16. Obtaining Shared/Exclusive Locks (IEAVELK) (Part 6 of 6)

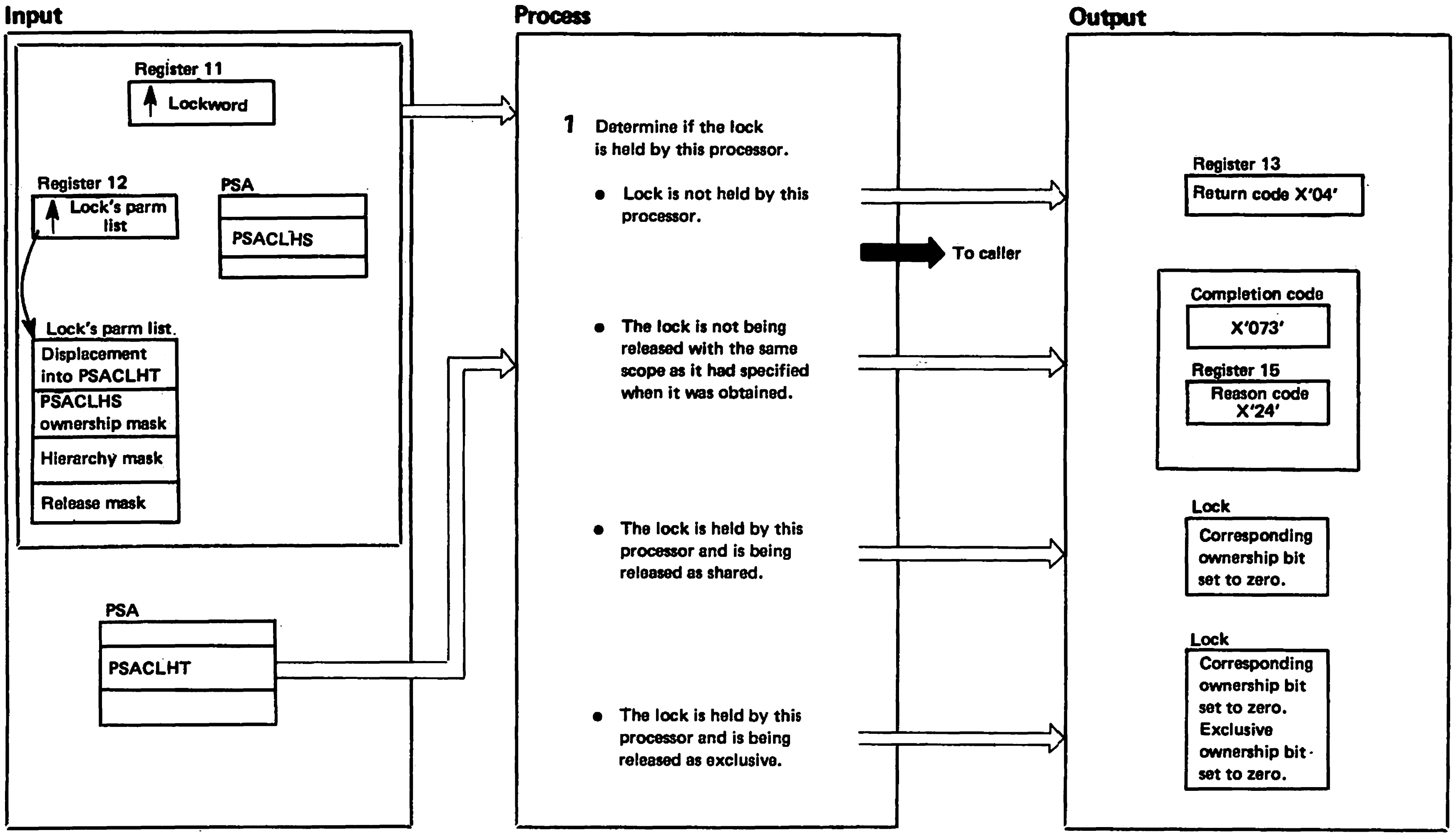
Extended Description **Module** **Label**

5 If a lock held is higher than the one requested, the caller abnormally terminates with a completion code of X'073' and a reason code of X'08' in register 15.

6 The caller owns the lock requested. The lock manager validates the request by checking if the lock is held with the same scope as was requested.

- If the scope is different, abnormally terminates the caller with a completion code of X'073' and a reason code of X'24' in register 15
- If the scope is the same, returns control to the caller with a return code of X'04' in register 13.

Diagram SUP-17. Releasing Shared/Exclusive Locks - Unconditional and Conditional (IEAVELK) (Part 1 of 4)



Input

Register 11

Lockword

Register 12

Lock's parm list

PSA

PSACLHS

Lock's parm list.

Displacement into PSACLHT

PSACLHS ownership mask

Hierarchy mask

Release mask

PSA

PSACLHT

Process

1 Determine if the lock is held by this processor.

- Lock is not held by this processor.

To caller

- The lock is not being released with the same scope as it had specified when it was obtained.

- The lock is held by this processor and is being released as shared.

- The lock is held by this processor and is being released as exclusive.

Output

Register 13

Return code X'04'

Completion code

X'073'

Register 15

Reason code X'24'

Lock

Corresponding ownership bit set to zero.

Lock

Corresponding ownership bit set to zero. Exclusive ownership bit set to zero.

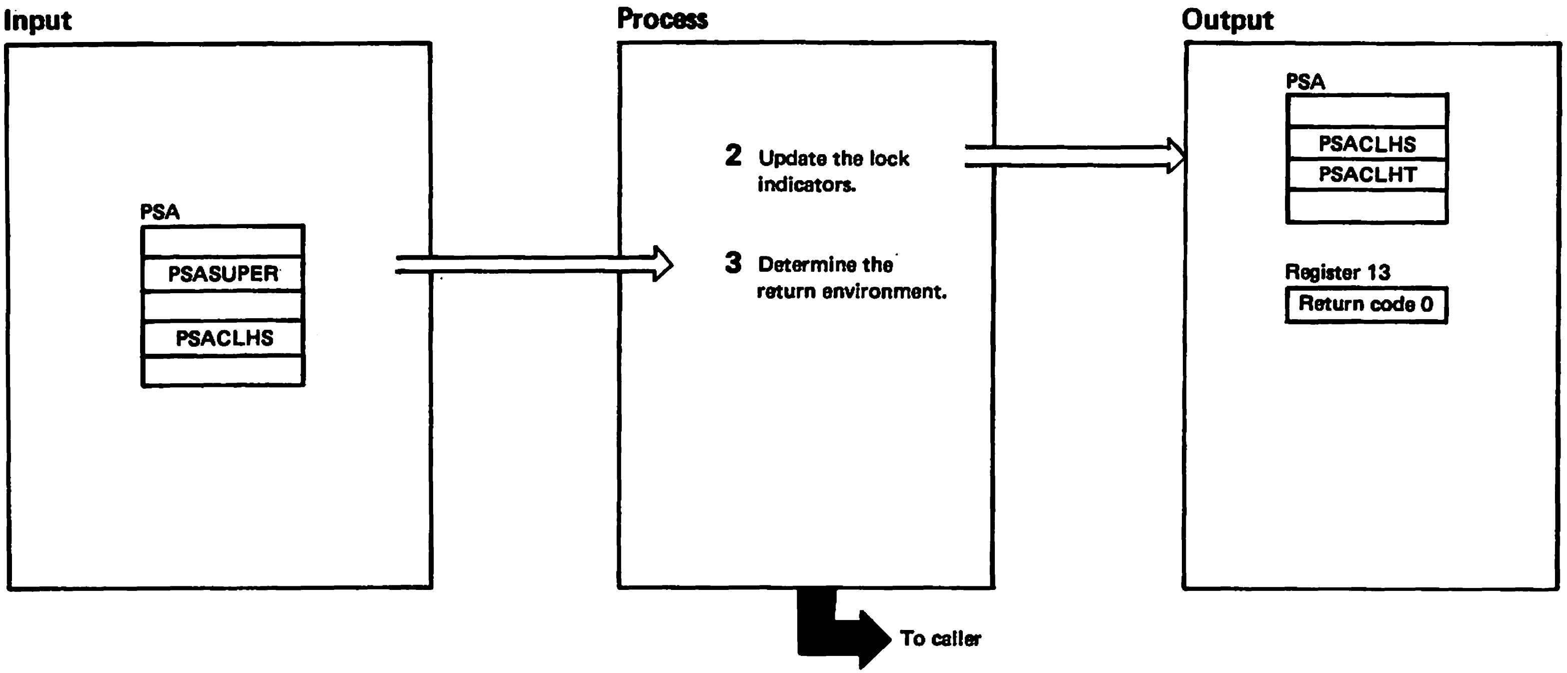
Diagram SUP-17. Releasing Shared/Exclusive Locks - Unconditional and Conditional (IEAVELK) (Part 2 of 4)

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

1 IEAVELK determines if the lock is held by this processor.

- If the lock is not held by this processor, IEAVELK returns to the caller with a return code of X'04' in register 13.
- If the lock is being released with a different scope attribute than was specified when the lock was obtained, the caller abnormally terminates with a completion code of X'073' and a reason code of X'24'.
- If the lock is being released as shared, resets the corresponding ownership bit in bit positions 16 - 31 of the lockword to zero.
- If the lock is being released as exclusive, resets the corresponding ownership bit in bit positions 16 - 31 of the lockword to zero. Additionally, resets the exclusive ownership bit (bit 0 of the lockword) to zero. After releasing the lock as exclusive, the only bit in the lockword that could still be on is the exclusive pending bit (bit 1 of the lockword).

Diagram SUP-17. Releasing Shared/Exclusive Locks – Unconditional and Conditional (IEAVELK) (Part 3 of 4)



2 Update the lock indicators.

3 Determine the return environment.

Diagram SUP-17. Releasing Shared/Exclusive Locks – Unconditional and Conditional (IEAVELK) (Part 4 of 4)

Extended Description

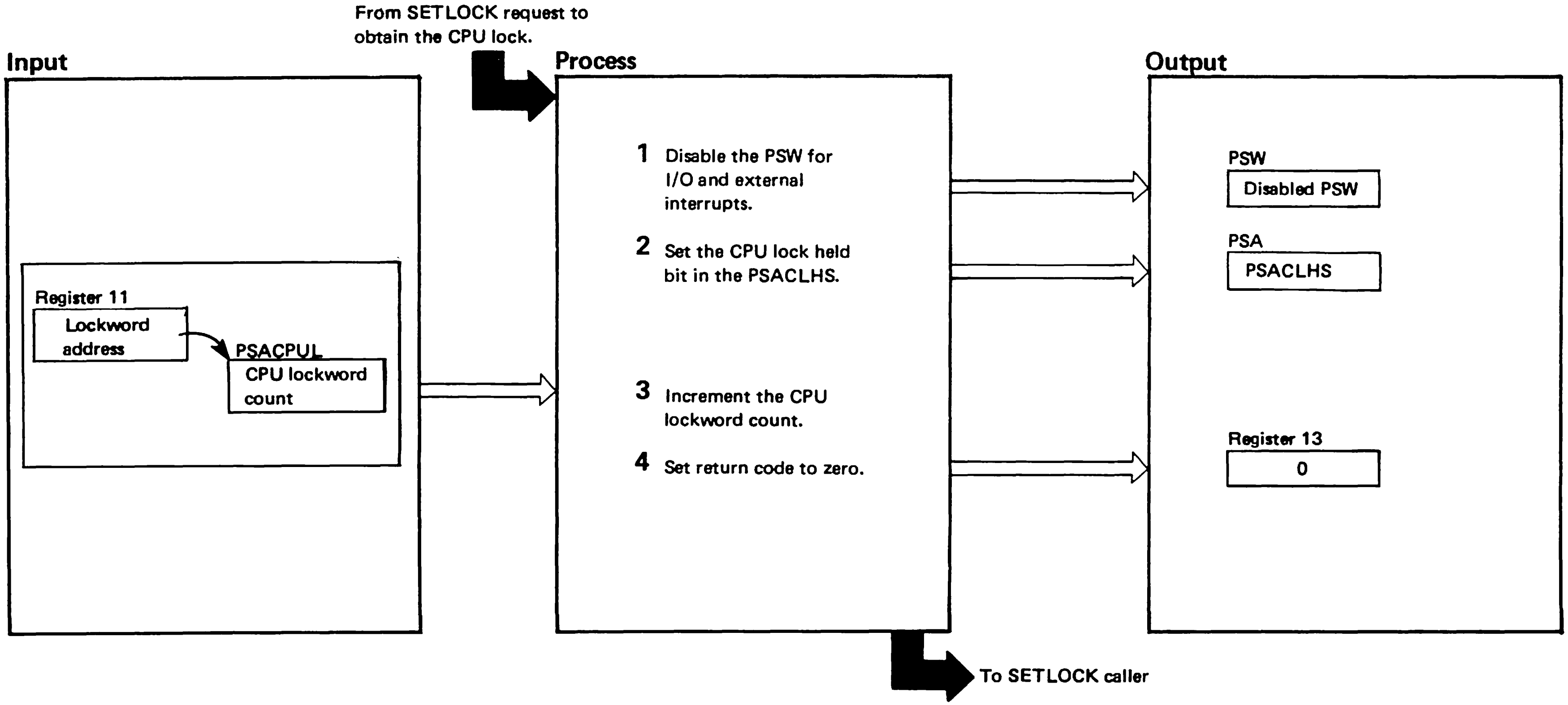
2 IEAVELK updates the lock indicators by clearing the appropriate bit in the current locks held string. If the lock was released as exclusive, IEAVELK also sets the high order bit of the corresponding current locks held table (PSACLHT) slot to zero to indicate that the lock is no longer held.

3 If any one of the following conditions is met, IEAVELK returns to the caller in a disabled state:

- A spin lock is held.
- A super bit is set.
- The caller requested that control be returned in a disabled state.

Otherwise, IEAVELK returns in an enabled state. In either case, the return code is zero.

Diagram SUP-18. Obtaining CPU Lock (IEAVELK) (Part 1 of 2)



1 Disable the PSW for I/O and external interrupts.

2 Set the CPU lock held bit in the PSACLHS.

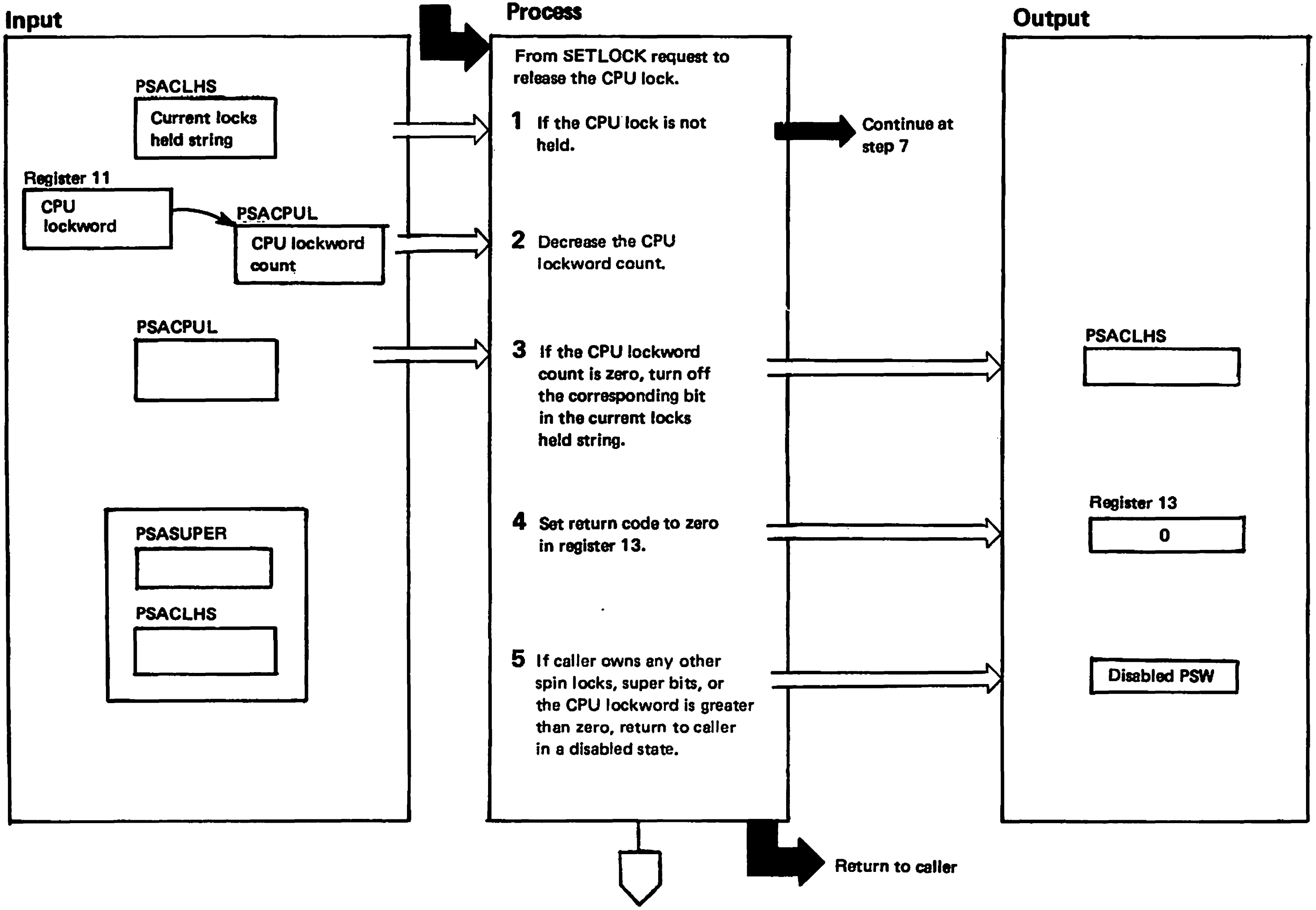
3 Increment the CPU lockword count.

4 Set return code to zero.

Diagram SUP-18. Obtaining CPU Lock (IEAVELK) (Part 2 of 2)

Extended Description	Module	Label
1 IEAVELK disables the PSW for I/O and external interrupts to serialize the CPU lockword, PSACPUL.	IEAVELK	CPUOBT
2 IEAVELK sets bit 0 of the current locks held string, PSACLHS, to one to indicate that the CPU lock is owned on this processor.		
3 IEAVELK increments the CPU lock count in the CPU lockword, PSACPUL, by one.		
4 IEAVELK sets a return code of zero in register 13.		

Diagram SUP-19. Releasing the CPU Lock (IEAVELK) (Part 1 of 4)



From SETLOCK request to release the CPU lock.

1 If the CPU lock is not held.

Continue at step 7

2 Decrease the CPU lockword count.

3 If the CPU lockword count is zero, turn off the corresponding bit in the current locks held string.

4 Set return code to zero in register 13.

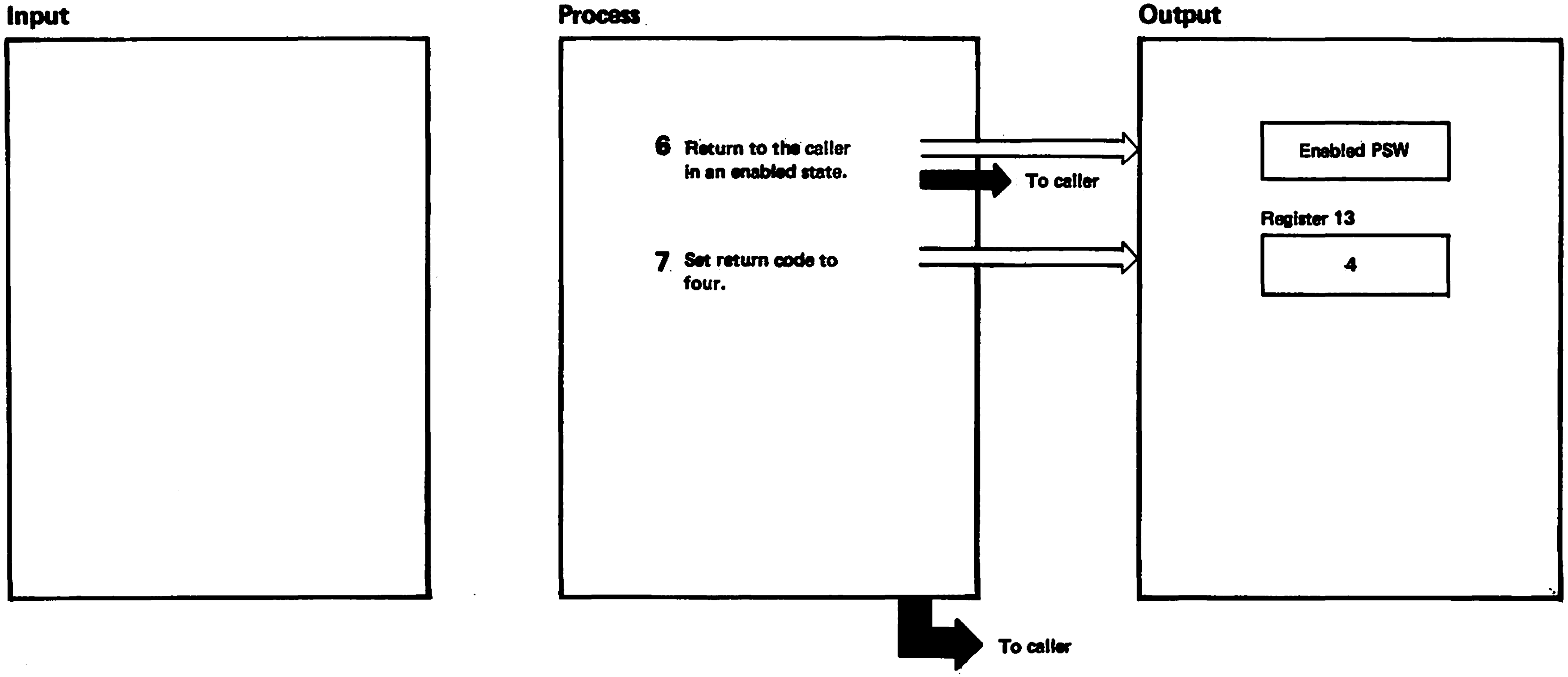
5 If caller owns any other spin locks, super bits, or the CPU lockword is greater than zero, return to caller in a disabled state.

Return to caller

Diagram SUP-19. Releasing the CPU Lock (IEAVELK) (Part 2 of 4)

Extended Description	Module	Label
1 If the CPU lock held bit is off in the current locks held string, PSACLHS, IEAVELK goes to step 7.	IEAVELK	CPUREL
2 IEAVELK decreases the CPU lock count in the CPU lockword, PSACPUL, by 1.		
3 If the count has gone to zero, IEAVELK turns off the CPU lock held bit in the current locks held string, PSACLHS.		
4 IEAVELK sets a return code of zero in register 13.		
5 If any super bits are on in the field PSASUPER or if the caller of this service owns any other spin locks, or the CPU lockword is greater than zero, IEAVELK returns to the caller in a disabled state.		

Diagram SUP-19. Releasing the CPU Lock (IEAVELK) (Part 3 of 4)



6 Return to the caller in an enabled state.

7 Set return code to four.

Diagram SUP-19. Releasing the CPU Lock (IEAVELK) (Part 4 of 4)

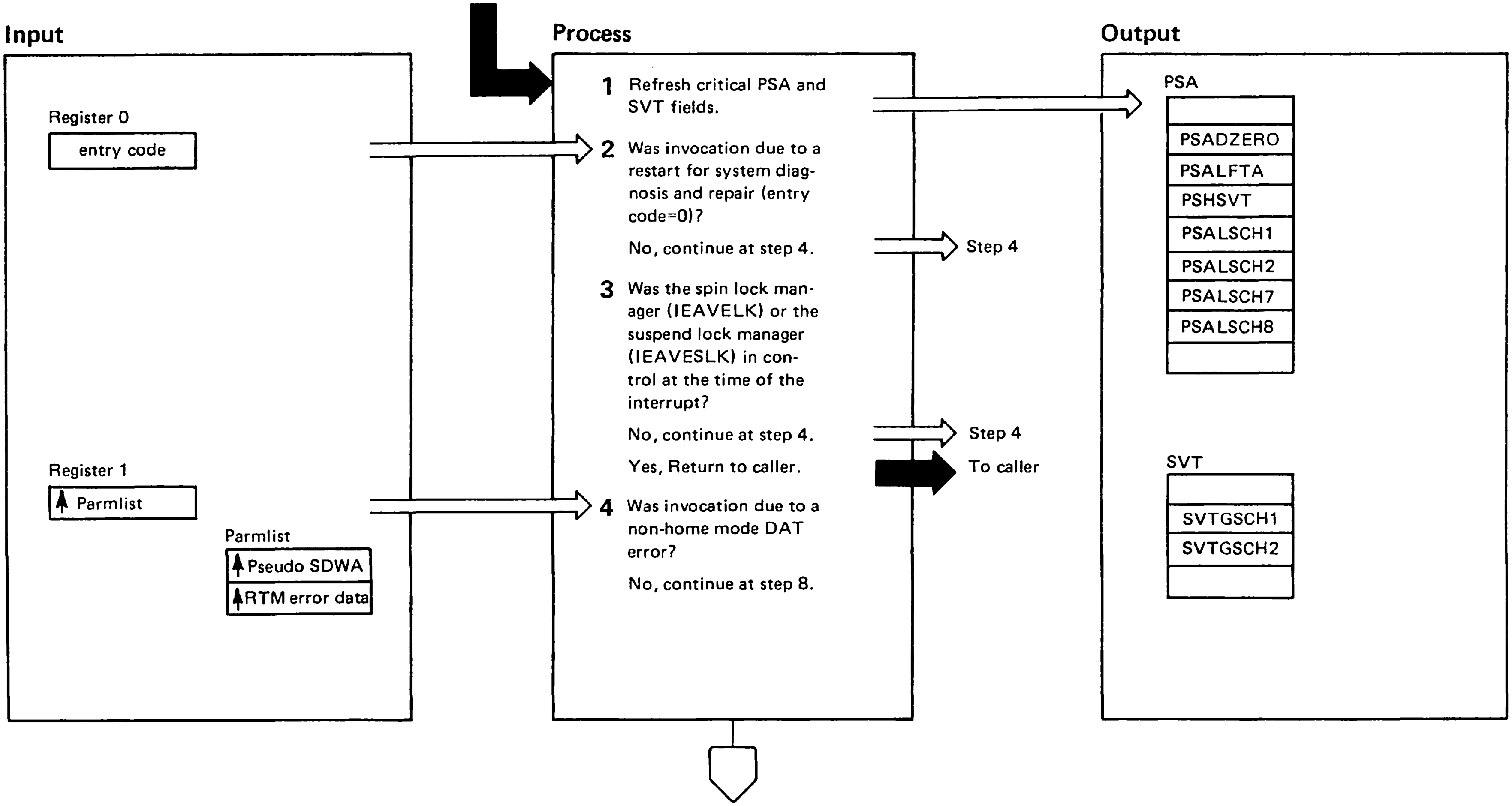
Extended Description	Module	Label
6 IEAVELK returns to the caller in an enabled state.		
7 IEAVELK sets a return code of four in register 13 indicating that the lock to be freed is not owned by the caller.		

Recovery Processing:

Lock recovery processing is described in the diagram "Address Space Verification Processing IEAVELCR, IEAVELKR, and IEAVEVRR".

The FRR routine for spin lock manager, ELKFRR, is an entry point within IEAVELKR.

Diagram SUP-20. Spin Lock Repair Routine (IEAVELKR) (Part 1 of 12)



1 Refresh critical PSA and SVT fields.

2 Was invocation due to a restart for system diagnosis and repair (entry code=0)?

No, continue at step 4.

3 Was the spin lock manager (IEAVELK) or the suspend lock manager (IEAVESLK) in control at the time of the interrupt?

No, continue at step 4.

Yes, Return to caller.

4 Was invocation due to a non-home mode DAT error?

No, continue at step 8.

Diagram SUP-20. Spin Lock Pepair Routine (IEAVELKR) (Part 2 of 12)

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

The spin lock repair routine is called as follows:

- By IEAVESAR (the Supervisor Analysis Router) due to SLIH mode processing by RTM or due to a restart for system diagnosis and repair via the Restart FLIH.
- By IEAVTRT1 due to non-home mode DAT error.

The spin lock repair routine correlates the current locks held string (PSACLHS) to the spin lockword with the assumption that a double error has not occurred. It ensures that a valid PSACLHS exists with respect to the spin lockwords. For spin locks, the lockword contents are always assumed to be correct. There are cases where the spin lock routine alters the PSACLHS, while in others, it may correct a lockword.

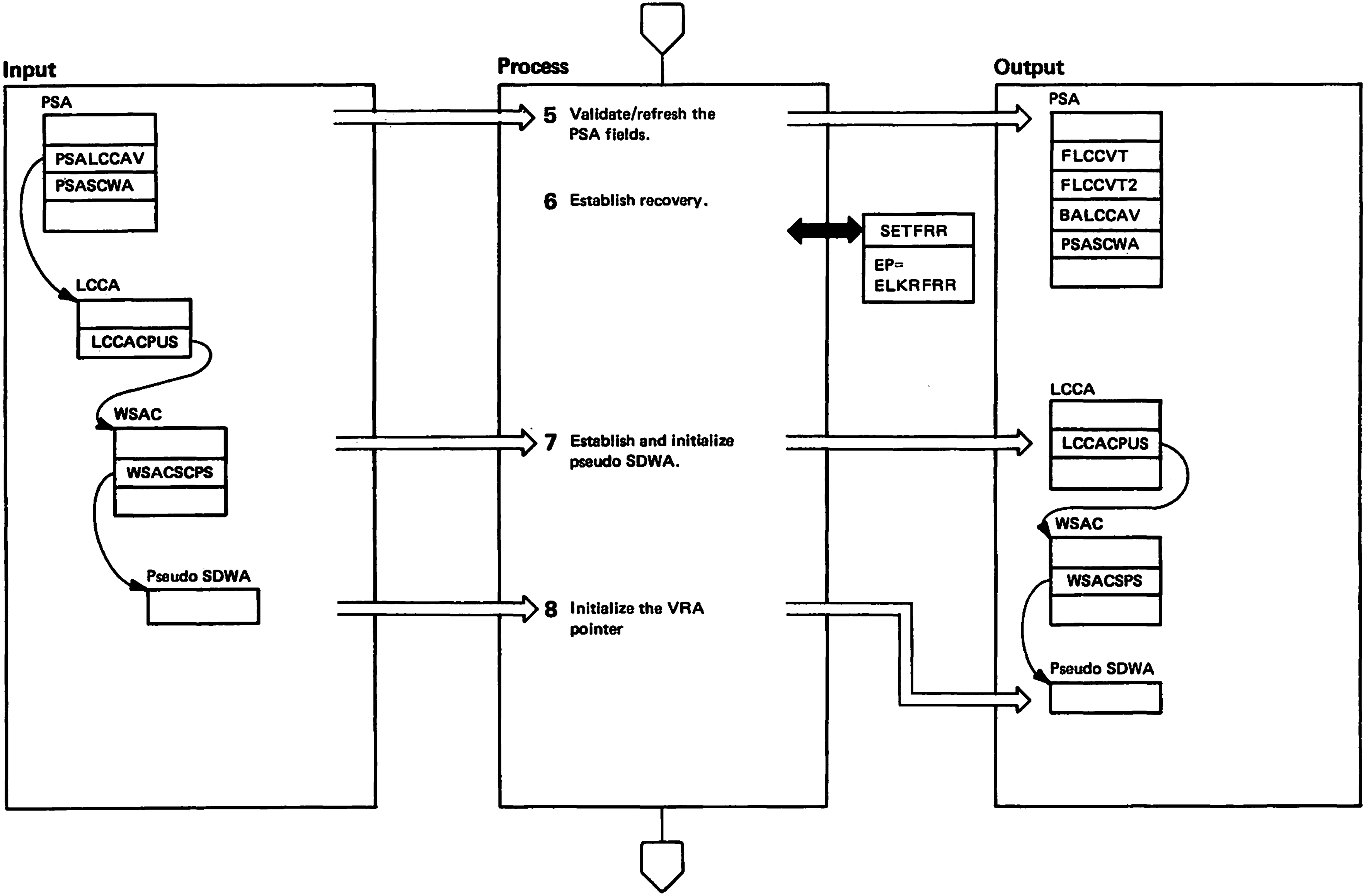
1 Spin lock repair refreshes system critical fields needed to successfully complete lock repair processing.

2 If the entry code in register 0 is zero, then a restart for system diagnosis and repair interrupt occurred. For this type of restart, repair and refresh processing is done, then the interrupted process is resumed at the next sequential instruction.

3 IEAVELKR checks the super bits and compares the interrupt PSW address to the beginning and end of spin lock manager (IEAVELK) and suspend lock manager (IEAVESLK). If either lock manager is executing at the time of interrupt, then IEAVELKR returns to the caller without doing lock repair. (Since the obtain/release of a lock may be partially complete, lock repair could clean up lockword and indicators if allowed to process.)

4 RTM (IEAVTRT1) calls lock repair for non-home mode DAT errors. If the first word of the parmlist is zero, RTM is the direct caller of lock repair. If RTM is the caller, lock repair must do additional setup processing prior to repairing locks.

Diagram SUP-20. Spin Lock Repair Routine (IEAVELKR) (Part 3 of 12)



5 Validate/refresh the PSA fields.

6 Establish recovery.

7 Establish and initialize pseudo SDWA.

8 Initialize the VRA pointer

Diagram SUP-20. Spin Lock Repair Routine (IEAVELKR) (Part 4 of 12)

Extended Description	Module	Label

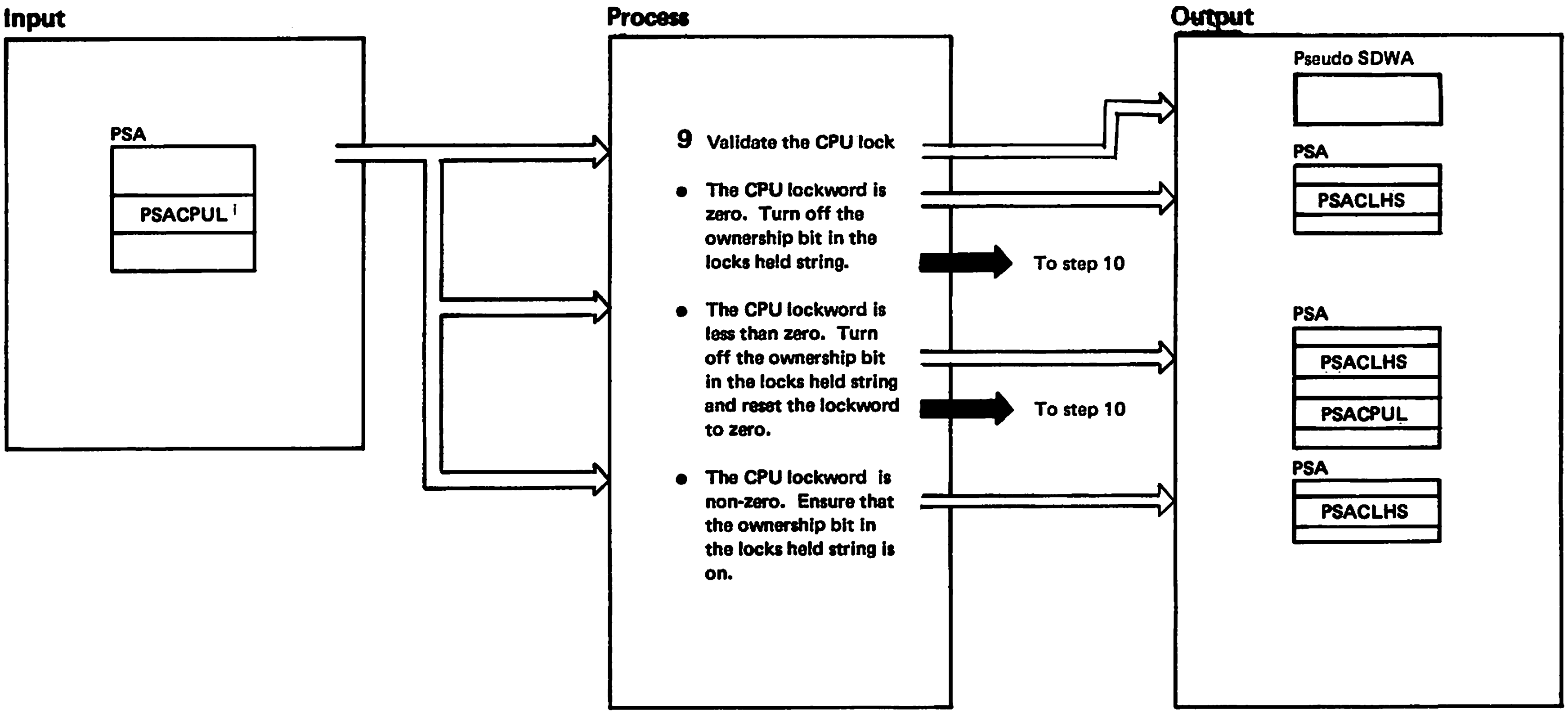
5 Spin lock repair validates and refreshes required additional fields in the PSA. These fields are validated/refreshed if the caller is IEAVESAR. Thus, this processing is only required if called by RTM directly.

6 IEAVELKR establishes recovery for the lock repair function. This recovery is required to clear recursive processing bit (SCWELKRF) that indicates to IEAVESAR that lock repair is processing as a result of direct call by RTM. This presents IEAVESAR from reinvoking lock repair.

7 IEAVELKR establishes addressability to the pseudo SDWA. This area is cleared to zero. The pseudo SDWA is used as an area for logging repair data.

8 IEAVELKR initializes the pointer to the VRA in the pseudo SDWA. The lock repair function uses the VRA area to log information concerning lockword repairs and the current lock held string (PSACLHS).

Diagram SUP-20. Spin Lock Repair Routine (IEAVELKR) (Part 5 of 12)



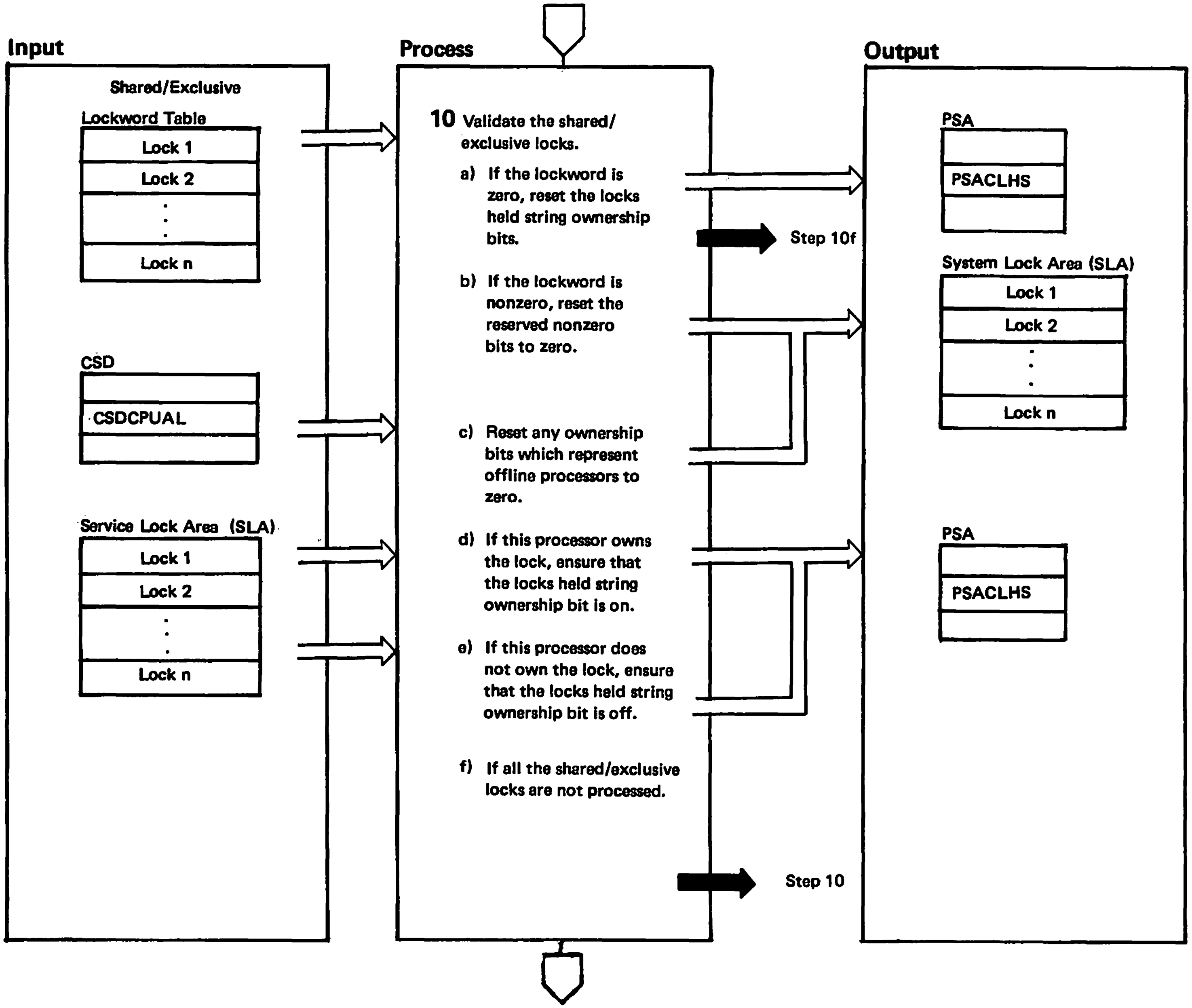
9 Validate the CPU lock

- The CPU lockword is zero. Turn off the ownership bit in the locks held string.
- The CPU lockword is less than zero. Turn off the ownership bit in the locks held string and reset the lockword to zero.
- The CPU lockword is non-zero. Ensure that the ownership bit in the locks held string is on.

Diagram SUP-20. Spin Lock Repair Routine (IEAVELKR) (Part 6 of 12)

Extended Description	Module	Label
9 Validates the CPU lock. ● If an error is found, it logs the repair information in the VRA portion of the pseudo SDWA. ● If the lockword contains a zero count, set the current locks held string ownership bit to zero. ● If the CPU lockword contains a negative value, resets the lockword and the current locks held string ownership bit to zero. ● If the CPU lockword contains a non-zero positive value, sets the current locks held string ownership bit to one.		

Diagram SUP-20. Spin Lock Repair Routine (IEAVELKR) (Part 7 of 12)



10 Validate the shared/exclusive locks.

a) If the lockword is zero, reset the locks held string ownership bits.

b) If the lockword is nonzero, reset the reserved nonzero bits to zero.

c) Reset any ownership bits which represent offline processors to zero.

d) If this processor owns the lock, ensure that the locks held string ownership bit is on.

e) If this processor does not own the lock, ensure that the locks held string ownership bit is off.

f) If all the shared/exclusive locks are not processed.

Diagram SUP-20. Spin Lock Repair Routine (IEAVELKR) (Part 8 of 12)

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

10 Validates the shared/exclusive locks. If an error is found, it logs the repair information in the VRA portion of the pseudo SDWA.

a) If the shared/exclusive lockword contents is zero, resets the ownership bit in the locks held string to zero and continues at step 4e.

b) If any reserved bits are nonzero in the lockword, resets them to zero.

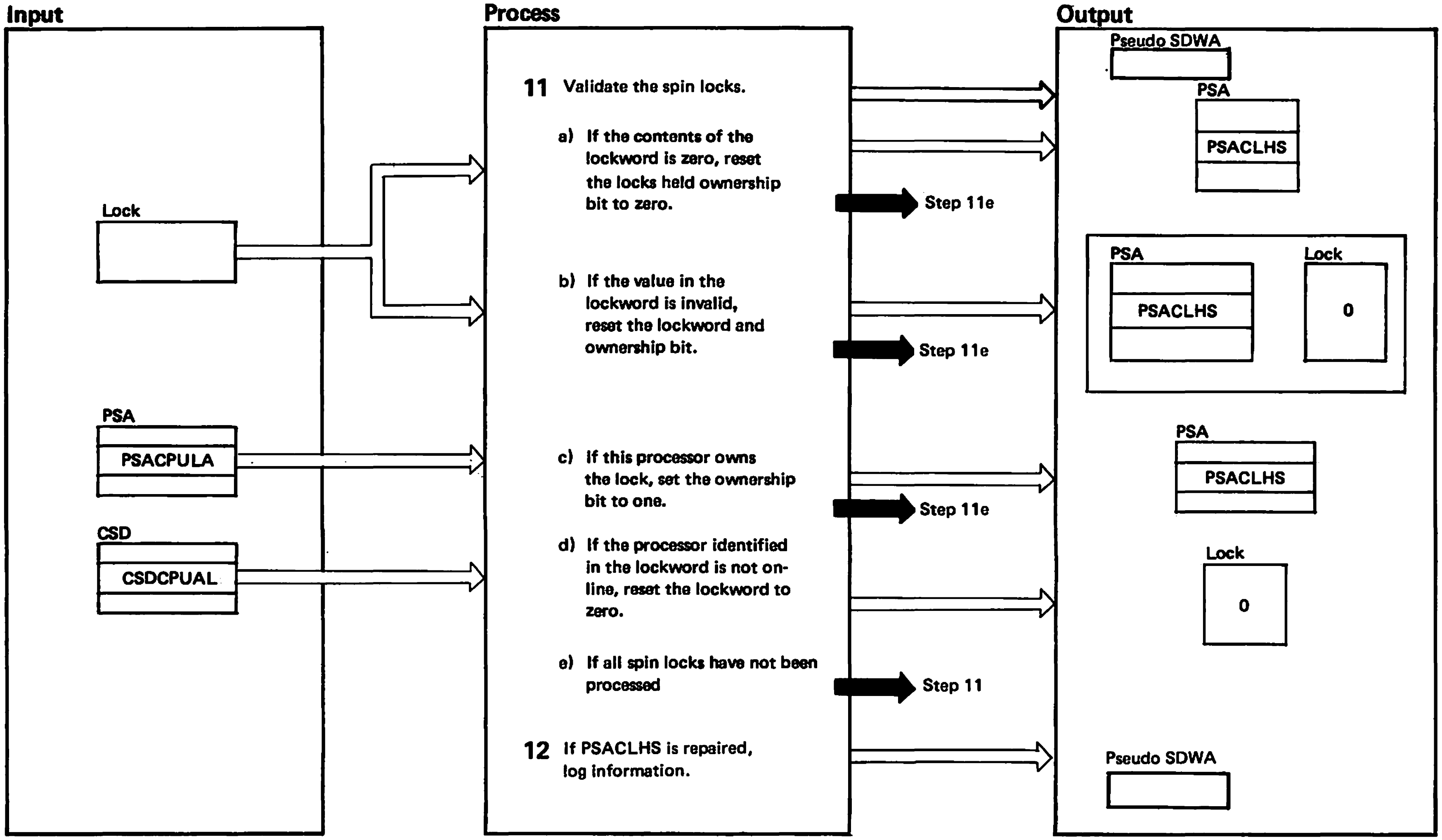
c) Interrogates the CPU alive mask (CSDCPUAL) to determine which processors are currently online. Uses this information to correlate the setting of the ownership bits in the shared/exclusive lockword. If the ownership bit is on in the lockword and it is off in the CPU alive string, resets the ownership bit in the lockword to zero.

d) If this processor's ownership bit is on in the lockword, unconditionally updates the locks held string to reflect ownership.

e) If this processor does not own the lock, sets the locks held string ownership bit to zero.

f) Repeats step 10 until all the shared/exclusive locks are processed.

Diagram SUP-20. Spin Lock Repair Routine (IEAVELKR) (Part 9 of 12)



11 Validate the spin locks.

a) If the contents of the lockword is zero, reset the locks held ownership bit to zero. → Step 11e

b) If the value in the lockword is invalid, reset the lockword and ownership bit. → Step 11e

c) If this processor owns the lock, set the ownership bit to one. → Step 11e

d) If the processor identified in the lockword is not on-line, reset the lockword to zero.

e) If all spin locks have not been processed → Step 11

12 If PSACLHS is repaired, log information.

Diagram SUP-20. Spin Lock Repair Routine (IEAVELKR) (Part 10 of 12)

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

11 Validates single and multiple level spin locks. If an error is found, it logs the repair function in the VRA portion of the pseudo SDWA.

a) If the lockword contents are zero, no processors own the lock. Resets the current locks held string ownership bit on the current processor to zero.

b) The range of valid values in the lockword is 40 through 4F, inclusive. If anything else is found in the lockword, resets the lockword to zero and unconditionally resets the current locks held string ownership bit on the current processor to zero.

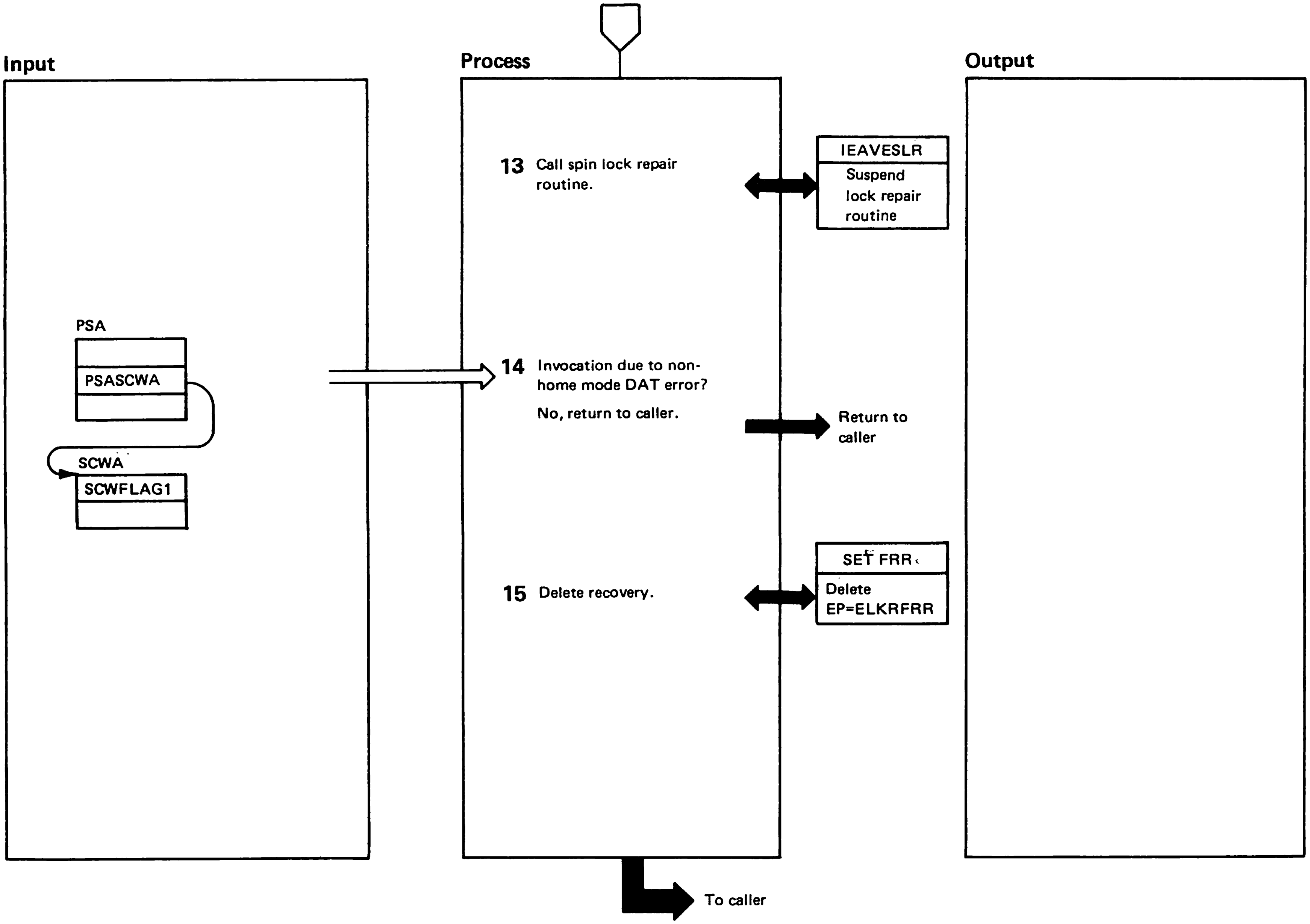
c) If the current processor's CPU ID is found in the lockword being validated, sets the current locks held string ownership bit to one.

d) If the lockword contains a valid nonzero processor ID and does not contain the current processor ID, interrogates the CPU alive mask (CSDCPUAL) to determine which processors are online. Use this information to determine if the processor ID in the lockword represents an online processor. If not, resets the lockword to zero.

e) Repeats the validation process for each spin lock.

12 IEAVELKR saves the contents of PSACLHS before beginning spin lock validation. If this field was updated, it logs the old and new values in the VRA portion of the pseudo SDWA.

Diagram SUP-20. Spin Lock Repair Routine (IEAVELKR) (Part 11 of 12)



13 Call spin lock repair routine.

14 Invocation due to non-home mode DAT error?

No, return to caller.

15 Delete recovery.

Diagram SUP-20. Spin Lock Repair Routine (IEAVELKR) (Part 12 of 12)

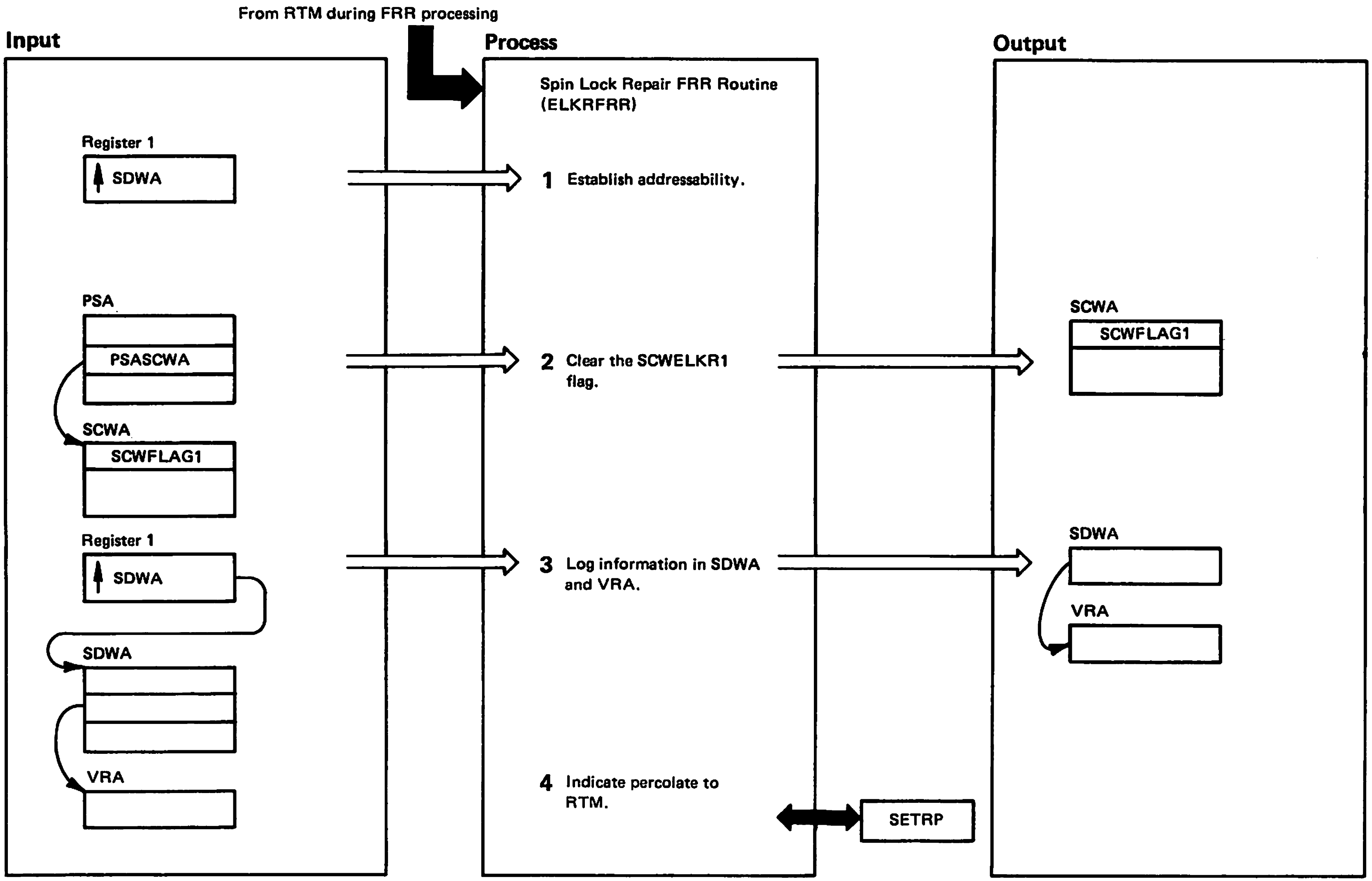
Extended Description **Module** **Label**

13 Validates the suspend lock environment via a call to the suspend lock repair routine (IEAVESLR).

14 Checks bit SCWELKRT. It the bit is not on, then it returns to the caller. If the bit is on, it continues at the next step.

15 Deletes the established recovery if the caller was RTM. Then it returns to the caller.

Diagram SUP-21. Spin Lock Repair Routine (IEAVELKR) (Part 1 of 2)



Spin Lock Repair FRR Routine (ELKRFRR)

1 Establish addressability.

2 Clear the SCWELKR1 flag.

3 Log information in SDWA and VRA.

4 Indicate percolate to RTM.

Diagram SUP-21. Spin Lock Repair Routine (IEAVELKR) (Part 2 of 2)

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

The spin lock repair FRR routine (ELKRFRR) receives control due to an unexpected error occurring during the lock repair function. The primary function of the recovery routine is to clear the bit SCWELKRF. This bit indicates that lock repair was called directly by RTM. This bit is tested by the supervisor analysis router (IEAVESAR) to prevent recursive entry into lock repair. SCWELKRF must be cleared in an error situation in order to allow the router to continue to perform its repair/refresh function.

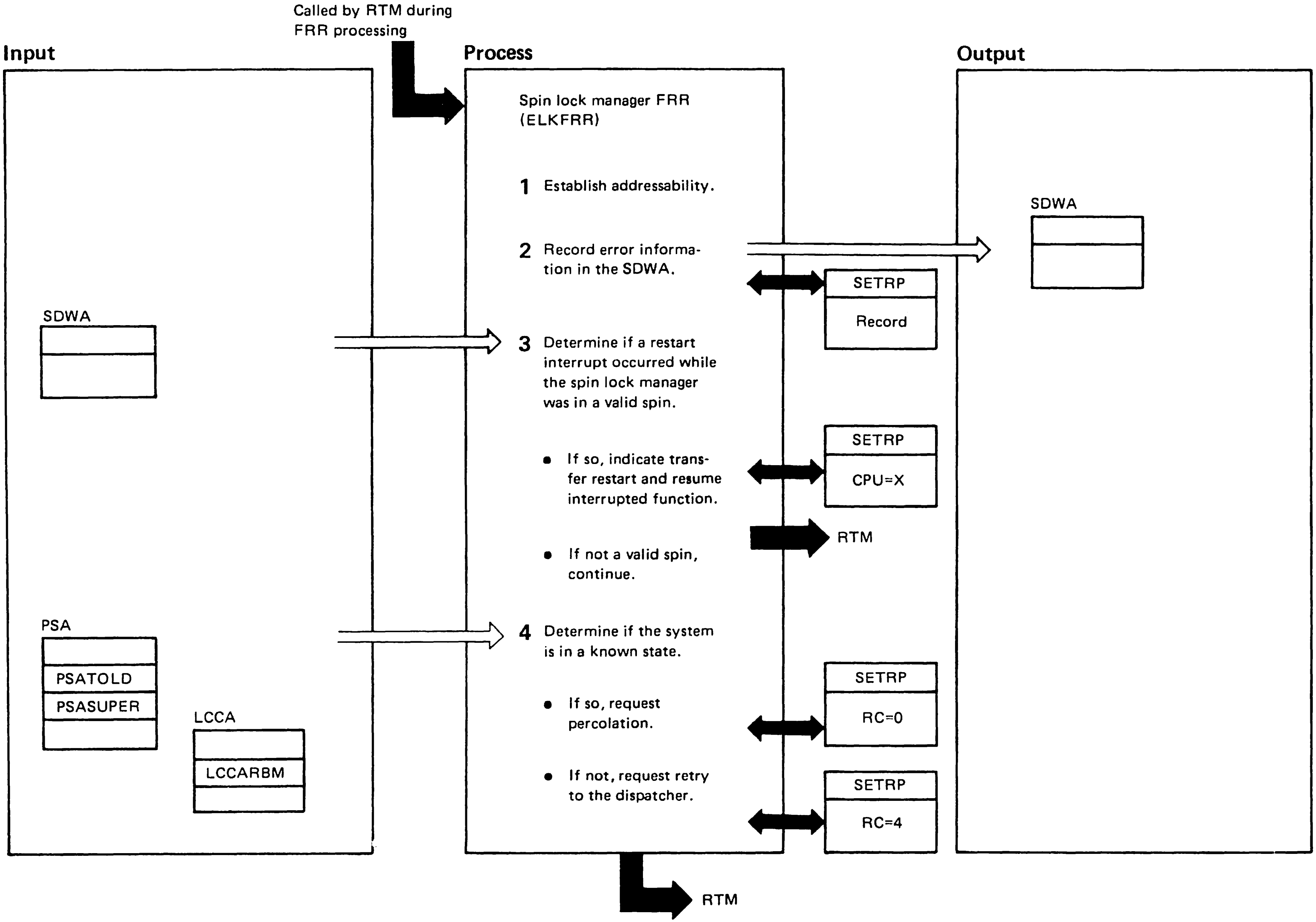
1 Establishes module addressability and establishes addressability to SDWA and VRA.

2 Clears the bit used to indicate repair was invoked directly by RTM due to non-home mode DAT error.

3 Logs diagnostic information in the SDWA and VRA.

4 Uses the SETRP macro to indicate to RTM to continue with termination. Returns to RTM.

Diagram SUP-22. Spin Lock Manager FRR Routine (IEAVELKR) (Part 1 of 2)



Spin lock manager FRR (ELKFRR)

1 Establish addressability.

2 Record error information in the SDWA.

3 Determine if a restart interrupt occurred while the spin lock manager was in a valid spin.

- If so, indicate transfer restart and resume interrupted function.
- If not a valid spin, continue.

4 Determine if the system is in a known state.

- If so, request percolation.
- If not, request retry to the dispatcher.

Diagram SUP-22. Spin Lock Manager FRR Routine (IEAVELKR) (Part 2 of 2)

Extended Description	Module	Label
The spin lock manager FRR routine receives control if an error or restart interrupt occurs and the spin lock manager (IEAVELK) has set an FRR.	IEAVELKR	ELKFRR

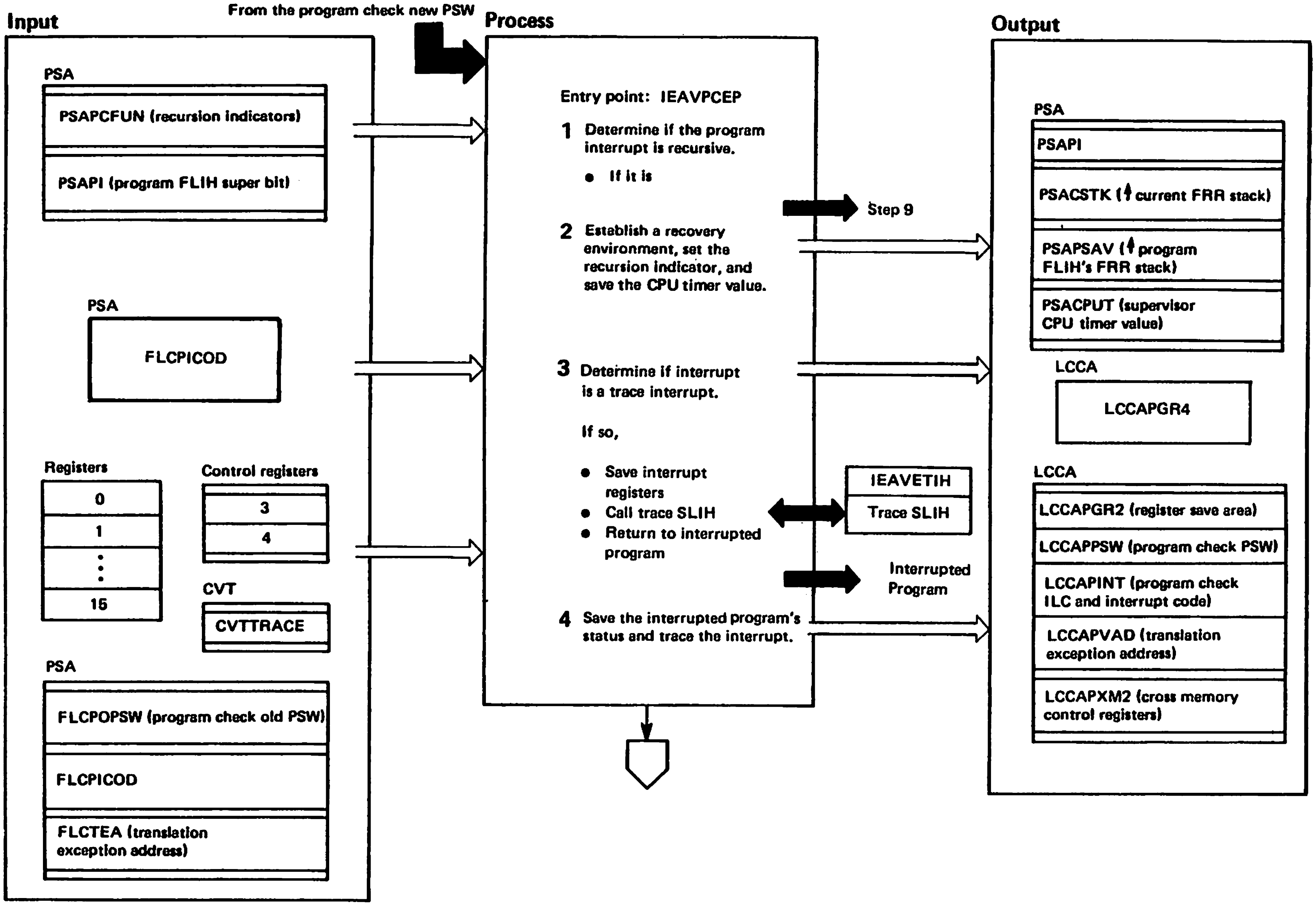
1 ELKFRR establishes module addressability and also establishes addressability to SDWA and VRA.

2 ELKFRR logs information in the SDWA and VRA. It indicates (via SETRP macro) to record the logged information.

3 If the error was a restart interrupt and it occurred while the spin lock manager (IEAVELK) was in a valid spin (LCCASPN=1), ELKFRR issues a SETRP macro to inform RTM of the valid spin and returns to RTM. Otherwise, continue with the next step.

4 ELKFRR determines if the system is in a known state (SRB mode, task mode, or supervisor system mode) by checking the LCCASRBM bit, the PSATOLD field, and the PSASUPER bits. If the system is in a known state, ELKFRR issues a SETRP macro to request percolation to the next FRR. If the system is in an unknown state, ELKFRR issues a SETRP requesting retry at entry point IEAODS in the dispatcher.

Diagram SUP-23. Program Check First Level Interrupt Handler (IEAVEPCO and IEAVEPC) (Part 1 of 14)


Input
From the program check new PSW
Process
Output
PSA
PSAPCFUN (recursion indicators)
PSAPI (program FLIH super bit)
PSA
FLCPICOD
Registers
0
1
15
Control registers
3
4
CVT
CVTTRACE
PSA
FLCPOPSW (program check old PSW)
FLCPICOD
FLCTEA (translation exception address)
Step 9
IEAVETIH
Trace SLIH
Interrupted Program
PSA
PSAPI
PSACSTK (↑ current FRR stack)
PSAPSAV (↑ program FLIH's FRR stack)
PSACPUT (supervisor CPU timer value)
LCCA
LCCAPGR4
LCCA
LCCAPGR2 (register save area)
LCCAPPSW (program check PSW)
LCCAPINT (program check ILC and interrupt code)
LCCAPVAD (translation exception address)
LCCAPXM2 (cross memory control registers)

Entry point: IEAVPCEP

1 Determine if the program interrupt is recursive.

- If it is

2 Establish a recovery environment, set the recursion indicator, and save the CPU timer value.

3 Determine if interrupt is a trace interrupt.

If so,

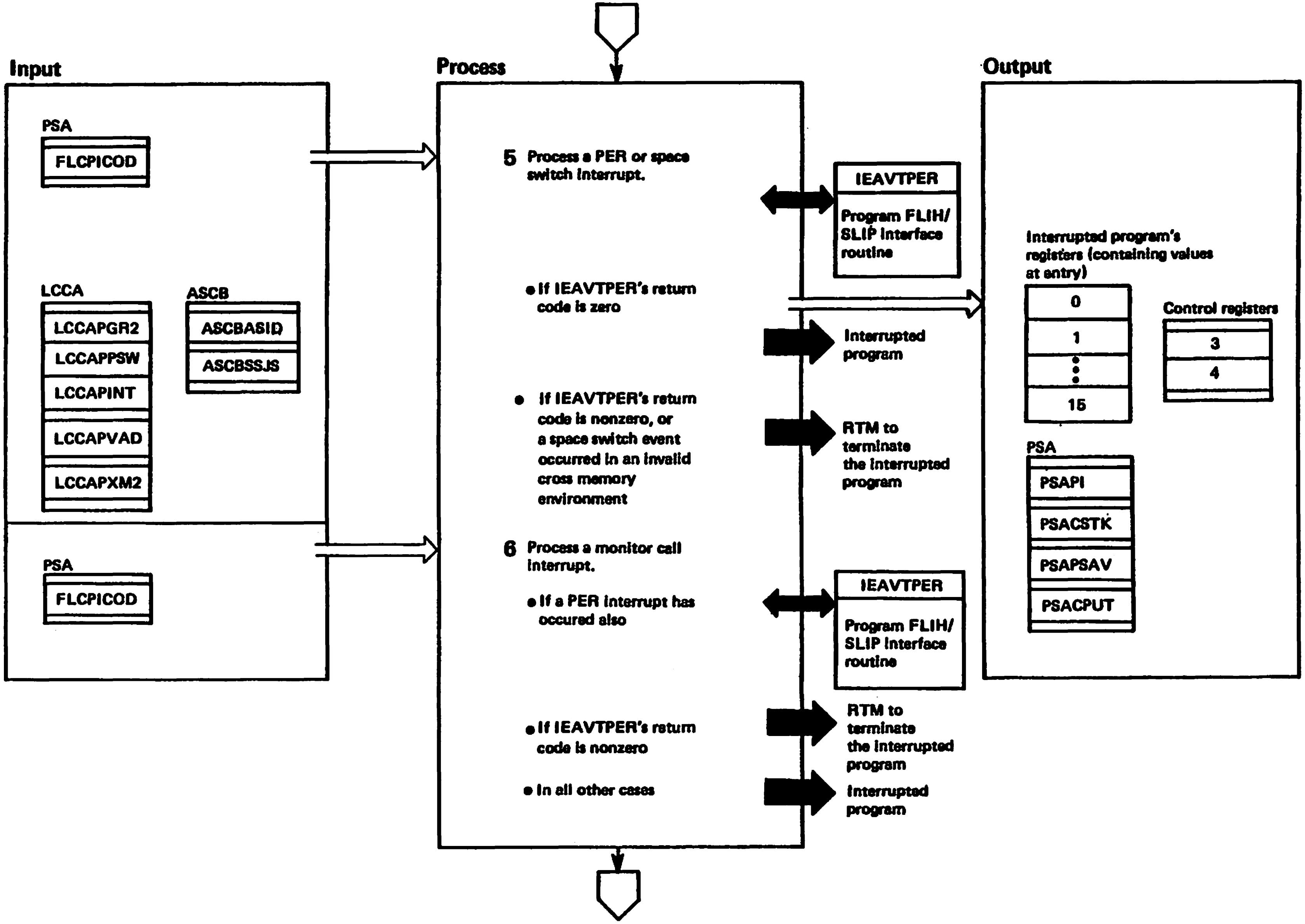
- Save interrupt registers
- Call trace SLIH
- Return to interrupted program

4 Save the interrupted program's status and trace the interrupt.

Diagram SUP-23. Program Check First Level Interrupt Handler (IEAVEPCO and IEAVEPC) (Part 2 of 14)

Extended Description	Module	Label
After a program check occurs, the program check first level interrupt handler (FLIH) receives control from the program check new PSW. The program FLIH determines the program check type and routes control to the appropriate second level interrupt handler to process it. The program FLIH is made up of two modules, IEAVEPCO, which executes with DAT disabled and without virtual addressing, and IEAVEPC, which executes with DAT enabled. While performing the program FLIH functions, the two modules pass control back and forth using an LPSW instruction. This diagram describes the program FLIH as a whole, and notes in the extended description which module is actually performing the function being described. In general, if the program interrupt is recursive, IEAVEPCO handles it. Otherwise, IEAVEPCO establishes a recovery environment, saves the caller's registers and interrupt information, and gives control to IEAVEPC to finish processing the interrupt.		
1 The PSAPCFUN word contains recursion indicators. If it is not zero, or if the program FLIH super bit (PSAPI) is on, the interrupt is recursive. IEAVEPCO branches to label RECURSN to handle it. This processing is described in steps 10-12.	IEAVEPCO	IEAVPCEP
2 To establish recovery, IEAVEPCO saves the current FRR stack address and makes the program FLIH's FRR stack the current one. IEAVEPCO also sets the PSAPI bit and saves the current CPU timer value for calculating job step timing later.		
3 If the program interrupt code indicates a trace interrupt, IEAVEPCO calls the trace SLIH (IEAVETIH) to handle the interrupt. • Saves the interrupted program's registers in LCCAPGR4. • Calls IEAVETIH to handle the interrupt. • After the trace interrupt has been handled, returns control to the interrupted program.		
4 IEAVEPCO saves in the LCCA the interrupted program's registers, old PSW, instruction length code, interrupt code, translation exception address, and cross memory control registers 3 and 4. It then enables DAT and transfers control to the DAT-on section of the program FLIH (IEAVEPC). IEAVEPC traces all interrupts except PER and space switch events. These are not traced until later, after the program FLIH/SLIP interface (IEAVTPER) has been called. The program FLIH traces PER or space switch events then only if IEAVTPER indicates that the interrupted program is to be terminated. When the interrupted program is to be redispatched, the PER or space switch interrupt is not traced.	IEAVEPCO IEAVEPC	 IEAQPK00
IEAVEPC system traces all non-recursive program interrupts, except monitor call interrupts, via the PTRACE macro instruction. All non-recursive program interrupts are GTF traced via the HOOK macro instruction.		

Diagram SUP-23. Program Check First Level Interrupt Handler (IEAVEPCO and IEAVEPC) (Part 3 of 14)



Input
PSA
FLCPICOD
LCCA
LCCAPGR2
LCCAPPSW
LCCAPINT
LCCAPVAD
LCCAPXM2
ASCB
ASCBASID
ASCBSSJS
PSA
FLCPICOD
Process
IEAVTPER
Program FLIH/ SLIP Interface routine
Interrupted program
RTM to terminate the interrupted program
IEAVTPER
Program FLIH/ SLIP Interface routine
RTM to terminate the interrupted program
Interrupted program
Output
Interrupted program's registers (containing values at entry)
0
1
15
Control registers
3
4
PSA
PSAPI
PSACSTK
PSAPSAV
PSACPUT

5 Process a PER or space switch interrupt.

- If IEAVTPER's return code is zero
- If IEAVTPER's return code is nonzero, or a space switch event occurred in an invalid cross memory environment

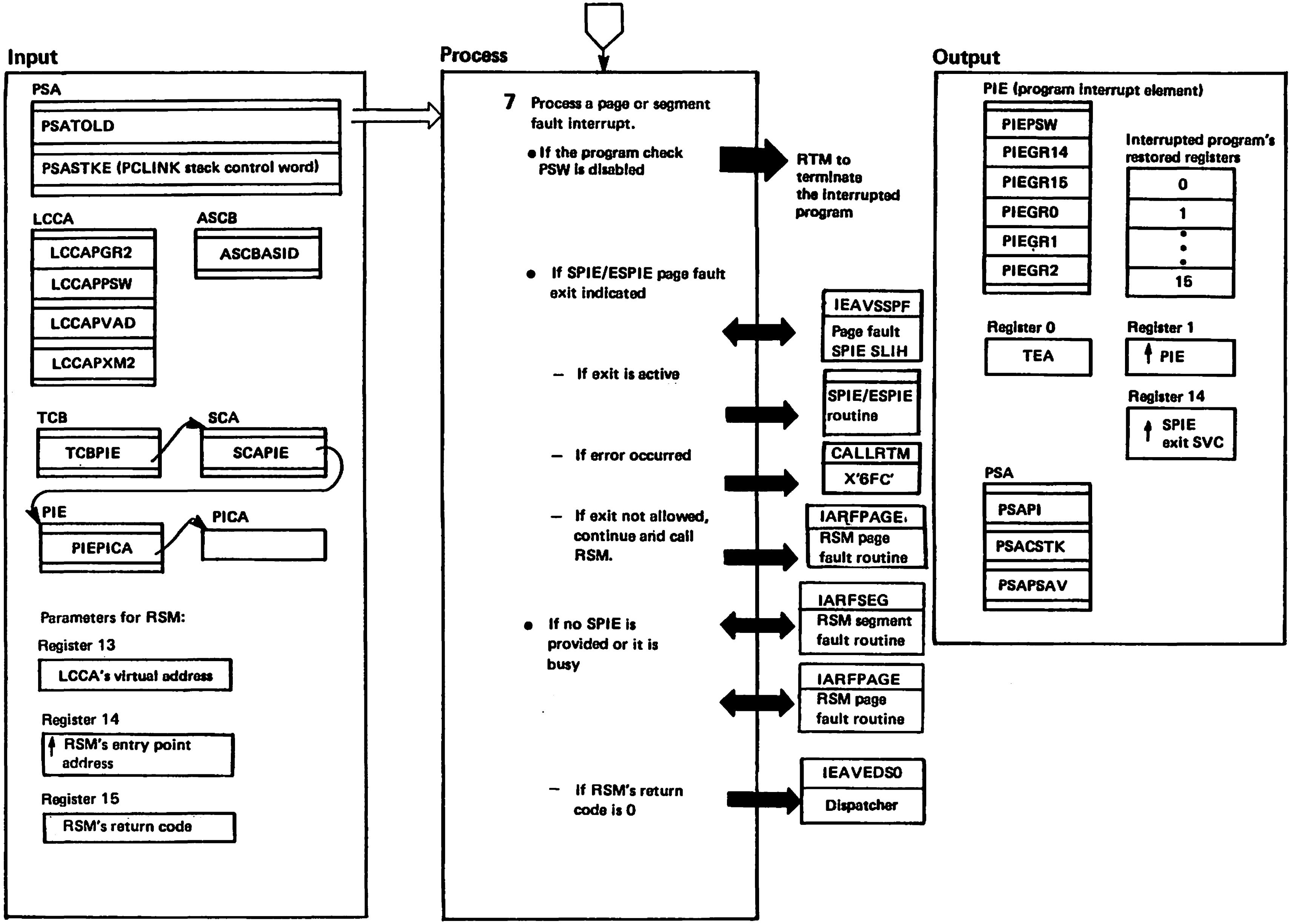
6 Process a monitor call interrupt.

- If a PER interrupt has occured also
- If IEAVTPER's return code is nonzero
- In all other cases

Diagram SUP-23. Program Check First Level Interrupt Handler (IEAVEPCO and IEAVEPC) (Part 4 of 14)

Extended Description	Module	Label
5 If the program interrupt code indicates a PER or space switch interrupt occurred, IEAVEPC calls the program FLIH/SLIP interface module (IEAVTPER). IEAVTPER calls either the space switch handler (IEAVTSSH) or the SLIP action processor (IEAVSTLP) to handle the interrupt. (See the IEAVTPER, IEAVTSSH, and IEAVTSLP diagrams for more information.) After IEAVTPER returns, IEAVEPC determines if the interrupt was a valid PER or space switch interrupt. It is not valid when: • IEAVTPER's return code is nonzero. • The interrupt occurred when a space switch event was attempted in an invalid cross memory environment. Specifically, the interrupt occurred when a routine tried to transfer control (using a PT instruction) to an address space after the job step had been terminated.	IEAVEPC IEAVTPER IEAVEPC	
In either of these cases, IEAVEPC traces the event as described in step 4, and calls RTM to terminate the interrupted unit of work.	IEAVEPC	
After IEAVTPER successfully processes a valid PER or space switch interrupt, IEAVEPC: • Restores the interrupted program's registers and PSW from values saved in the LCCA, and restores the CPU timer from the PSA • Deletes the program FLIH's recovery routine by restoring the FRR stack pointer to make the previous FRR stack current • Clears the PSAPI recursion indicator • Returns to the interrupted program	IEAVEPC	
6 If the program interrupt code indicates a monitor call only (PER is not also indicated), IEAVEPC already processed the interrupt when it called GTF to trace it in step 4. If the interrupt code indicates that a PER interrupt also occurred, IEAVEPC calls IEAVTPER to handle it. See step 5 for a description of how this is done.	IEAVEPC	

Diagram SUP-23. Program Check First Level Interrupt Handler (IEAVEPCO and IEAVEPC) (Part 5 of 14)



Input

PSA
PSATOLD
PSASTKE (PCLINK stack control word)

LCCA
LCCAPGR2
LCCAPPSW
LCCAPVAD
LCCAPXM2

ASCB
ASCBASID

TCB
TCBPIE

SCA
SCAPIE

PIE
PIEPICA

PICA

Parameters for RSM:

Register 13
LCCA's virtual address

Register 14
↑ RSM's entry point address

Register 15
RSM's return code

Process

7 Process a page or segment fault interrupt.

- If the program check PSW is disabled → RTM to terminate the interrupted program
- If SPIE/ESPIE page fault exit indicated ↔ IEAVSSPF Page fault SPIE SLIH
  - If exit is active → SPIE/ESPIE routine
  - If error occurred → CALLRTM X'6FC'
  - If exit not allowed, continue and call RSM. → IARFPAGE, RSM page fault routine
- If no SPIE is provided or it is busy ↔ IARFSEG RSM segment fault routine ↔ IARFPAGE RSM page fault routine
  - If RSM's return code is 0 → IEAVEDSO Dispatcher

Output

PIE (program interrupt element)
PIEPSW
PIEGR14
PIEGR15
PIEGR0
PIEGR1
PIEGR2

Interrupted program's restored registers
0
1
•
•
•
15

Register 0
TEA

Register 1
↑ PIE

Register 14
↑ SPIE exit SVC

PSA
PSAPI
PSACSTK
PSAPSAV

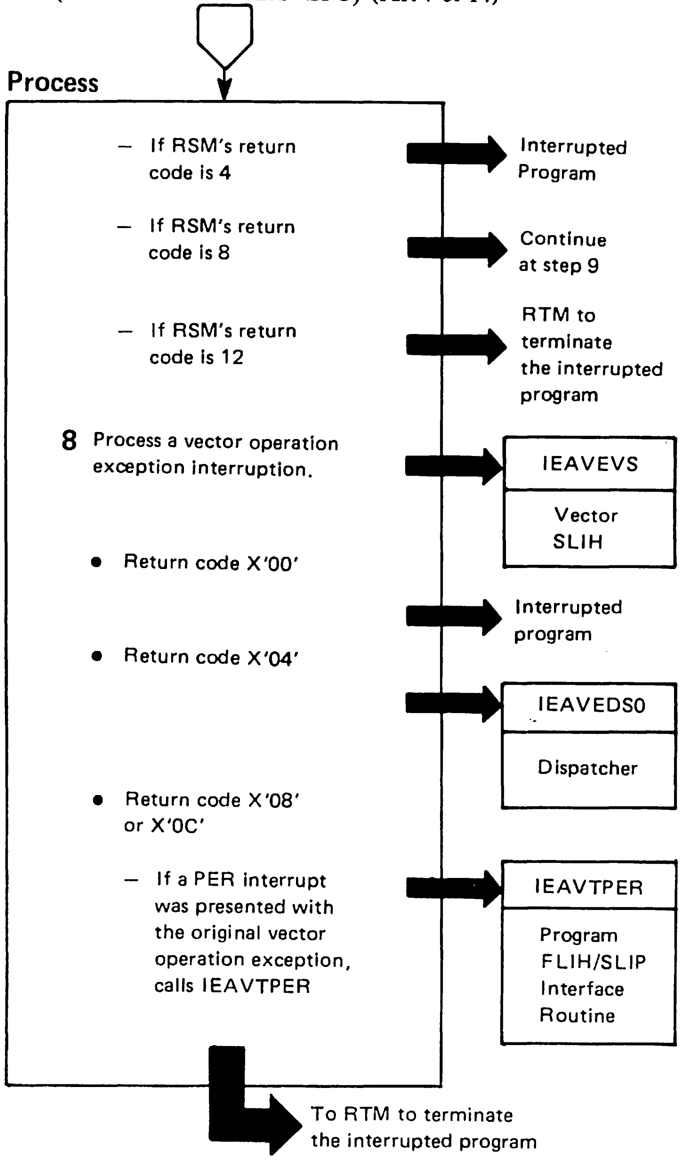
Diagram SUP-23. Program Check First Level Interrupt Handler (IEAVEPCO and IEAVEPC) (Part 6 of 14)

Extended Description	Module	Label
7 When processing a page or segment fault, IEAVEPC first determines if the program check PSW is disabled. If so, the page fault is invalid. IEAVEPC calls RTM to terminate the interrupted program.	IEAVEPC	
If the interrupted program is a task and has a page fault SPIE or ESPIE exit specified, the page fault SPIE/ESPIE routine (IEAVSSPF) is called to handle the exit.		
• If there is no SPIE/ESPIE exit allowed, or if it is busy, IEAVSSPF returns a code of X'04' which causes IEAVEPC to call RSM to handle the page fault.	IEAVESPI	IEAVSSPF
• If there is a SPIE/ESPIE exit, IEAVSSPF returns a code of X'08' which causes IEAVEPC to give control to the exit with the registers and PSW updated by IEAVSSPF.		
• If IEAVSSPF encountered an error referencing the SPIE/ESPIE control blocks, a return code of X'0C' is returned which causes IEAVEPC to call RTM to terminate the task with a code of X'6FC'.		

Extended Description	Module	Label
If a SPIE/ESPIE routine is not provided or if it is busy, IEAVEPC calls the paging supervisor (RSM) at entry point IARFPAGE for page faults and entry point IARFSEG for segment faults. Determines how IEAVGPC exits from page or segment fault processing by using the return code from RSM.	IARFS IARFS	IARFPAGE IARFSEG

Return Code	*Action taken*	Module	Label
0	I/O is required to resolve the page fault. RSM called the supervisor STOP/RESET routine (IEAVESRT) to suspend the interrupted program. (See the IEAVESRT diagram for more information.) IEAVEPC gives control to the dispatcher (IEAVEDS0) at entry point IEAODS1A.	IEAVEDS0	IEAODS1A
4	No paging I/O is required. Either the real storage frame containing the page was reclaimed, or a valid page was referenced for the first time. IEAVEPC returns control to the interrupted program.		
8	The page was not valid. IEAVEPC treats the interrupt as a protection exception and continues processing at the next step.		
12	An internal error occurred in RSM, IEAVEPC calls RTM to terminate the interrupted program with a code of X'028'.		

Diagram SUP-23. Program Check First Level Interrupt Handler (IEAVEPCO and IEAVEPC) (Part 7 of 14)



Process

– If RSM's return code is 4

Interrupted Program

– If RSM's return code is 8

Continue at step 9

– If RSM's return code is 12

RTM to terminate the interrupted program

8 Process a vector operation exception interruption.

IEAVEVS

Vector SLIH

- Return code X'00'

Interrupted program

- Return code X'04'

IEAVEDS0

Dispatcher

- Return code X'08' or X'0C'

  – If a PER interrupt was presented with the original vector operation exception, calls IEAVTPER

IEAVTPER

Program FLIH/SLIP Interface Routine

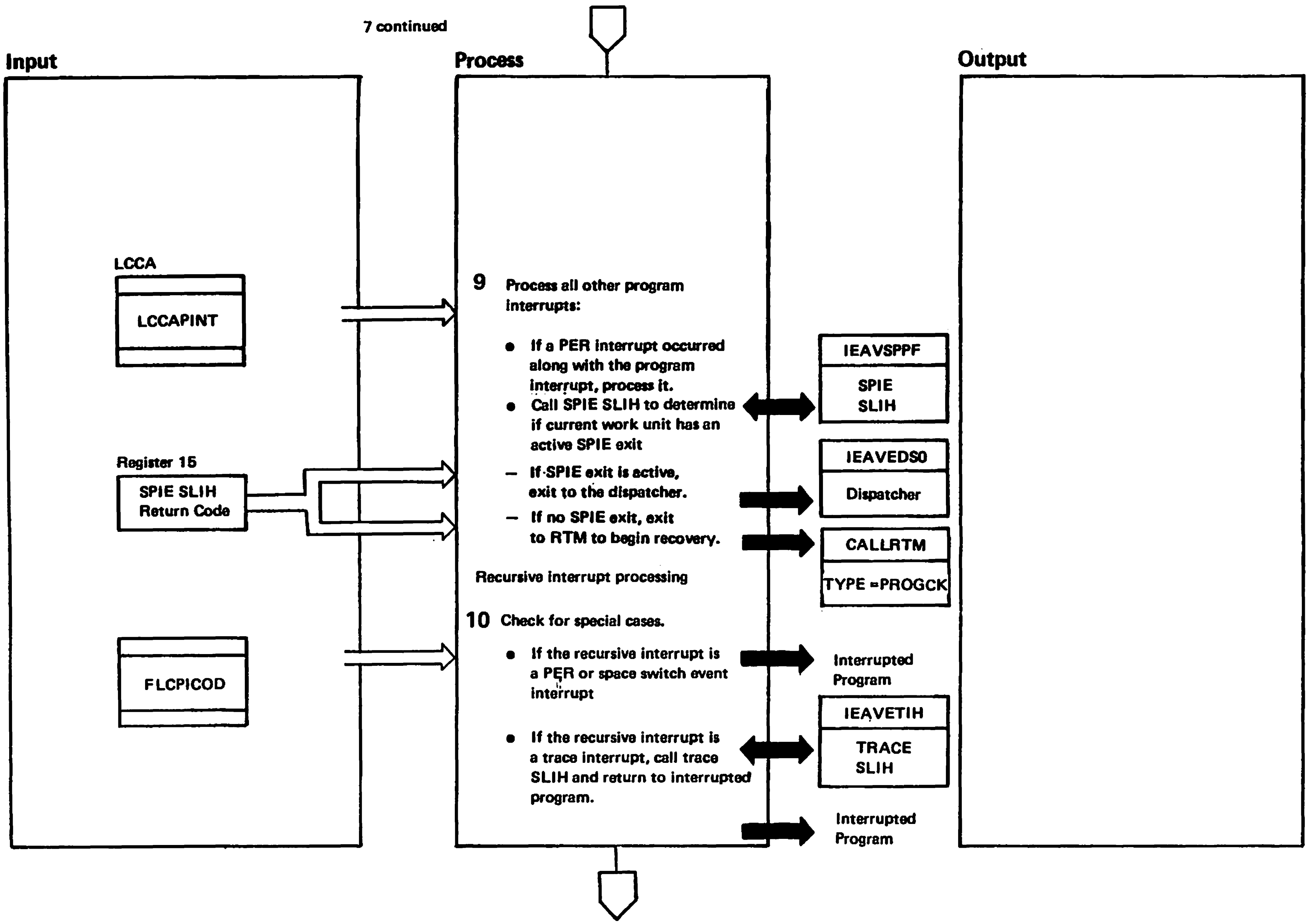
To RTM to terminate the interrupted program

Diagram SUP-23. Program Check First Level Interrupt Handler (IEAVEPCO and IEAVEPC) (Part 8 of 14)

Extended Description	Module	Label
8 When processing a vector operation exception interrupt, IEAVEPC calls the vector second level interrupt handler. IEAVEPC uses the return code in register 15 to determine how to exit from vector operation exception handling.	IEAVEPC	VECEXCP

Return Code	Action Taken	
X'00'	The vector SLIH successfully set up the task's vector environment and indicated that the task should be redispatched immediately. IEAVEPC returns control to the interrupted unit of work.	VSRETURN
X'04'	The vector SLIH successfully set up the vector environment for the task, but determined that either (1) the current processor is not eligible for vector work or (2) the vector SLIH suspended the interrupted unit of work and suspended an SRB to complete the creation of the vector environment. IEAVEPC exits to the dispatcher at entry point IEA0DS1A.	VSEXIT
X'08' X'0C'	The vector SLIH was unsuccessful in setting up the vector environment for the interrupted work unit ● If a PER interrupt was presented with the vector operation exception, IEAVTPER is called. IEAVEPC calls RTM to terminate the interrupted unit of work with a completion code of X'09F'. The associated reason code from the vector SLIH is also passed to RTM.	VSERROR

Diagram SUP-23. Program Check First Level Interrupt Handler (IEAVEPCO and IEAVEPC) (Part 9 of 14)



9 Process all other program interrupts:

- If a PER interrupt occurred along with the program interrupt, process it.
- Call SPIE SLIH to determine if current work unit has an active SPIE exit
  - — If SPIE exit is active, exit to the dispatcher.
  - — If no SPIE exit, exit to RTM to begin recovery.

Recursive interrupt processing

10 Check for special cases.

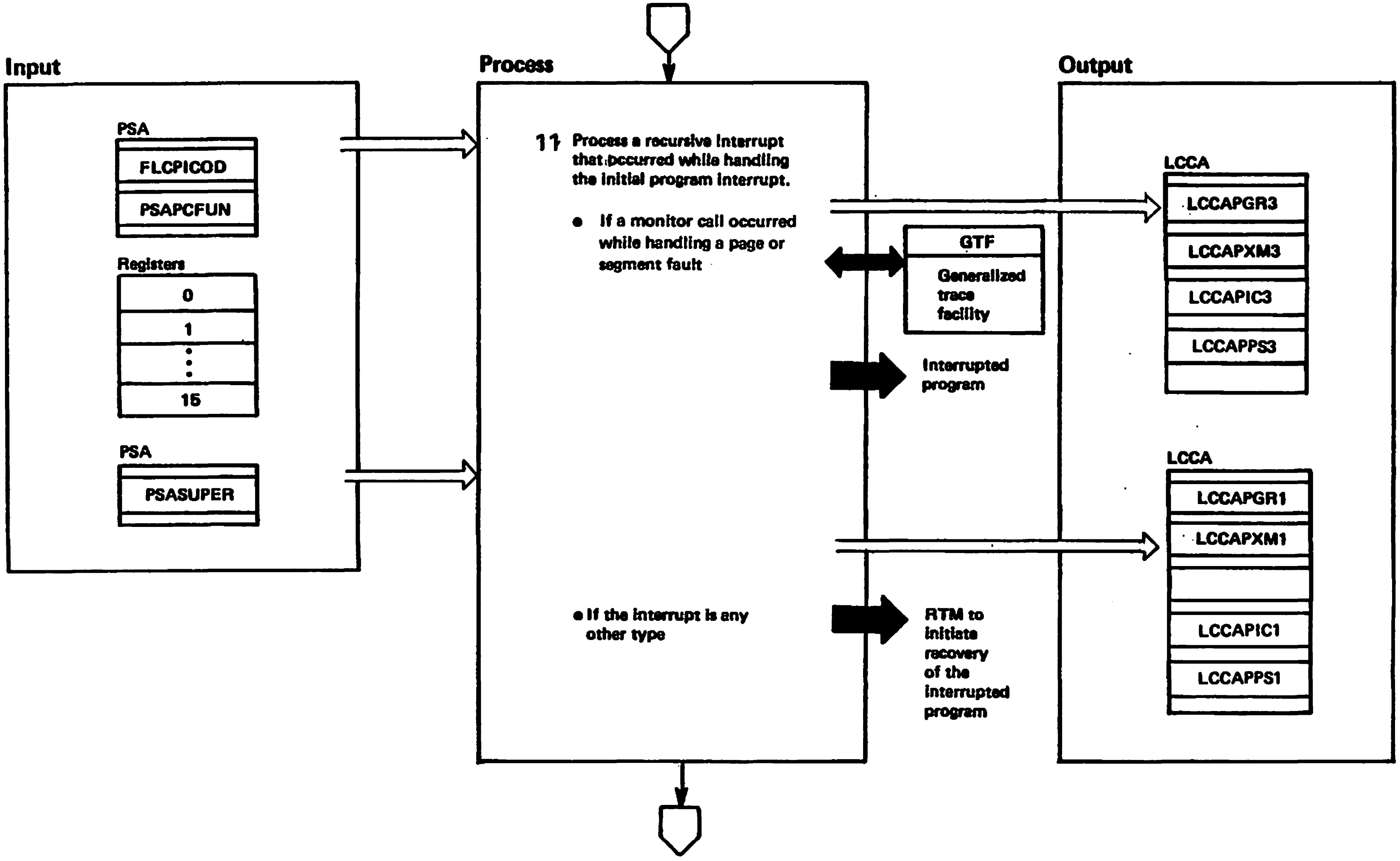
- If the recursive interrupt is a PER or space switch event interrupt
- If the recursive interrupt is a trace interrupt, call trace SLIH and return to interrupted program.

Diagram SUP-23. Program Check First Level Interrupt Handler (IEAVEPCO and IEAVEPC) (Part 10 of 14)

Extended Description	Module	Label
9 If there was a PER interrupt along with the program check, IEAVEPC calls IEAVTPER to handle it. Step 4 describes how this is done. IEAVESPI calls the SPIE SLIH (IEAVSPPF) to determine if an SPIE or ESPIE exit is in effect for the interrupted work unit.	IEAVEPC IEAVTPER IEAVESPI	 IEAVSPPF
• If there is an SPIE/ESPIE exit, IEAVSPPF returns a return code of X '04' which causes IEAVEPC to branch to the dispatcher IEA0DS entry point.	IEAVEGLU	IEA0DS
• If there is no SPIE/ESPIE exit, IEAVSPPF returns X '08' which causes IEAVEPC to exit to RTM via the CALLRTM macro.		
Recursive interrupt processing:		
IEAVEPCO provides six levels of recursive interrupt routines (PROGCK1, PROGCK2, PROGCK3, PROGCK4, DATERR, and LOADWAIT). When an recursive interrupt occurs while a previous one is being handled, IEAVEPCO gives control to the next higher level routine. IEAVEPCO keeps track of the recursion level in the program FLIH recursion word (PSAPCFUN). Multiple levels of routines are necessary to detect all types of DAT errors (two recursive program interrupts in the same address space).	IEAVEPCO	

Extended Description	Module	Label
10 When a recursive interrupt occurs, IEAVEPCO checks for special types of interrupts:		
• If a PER or space switch interrupt occurred, IEAVEPCO turns off the PER enabled mask in the resume PSW and returns to the interrupted program with PER disabled.		
• If a trace interrupt causes the recursion, IEAVEPCO saves the interrupt registers in LCCAPGR4 and calls the trace SLIH (IEAVETIH). After the SLIH runs, returns control to the interrupted program.		
When the recursive level is not one of the above special cases, IEAVEPCO gives control to the appropriate level routine. Step 11 describes how first recursive interrupts are handled; step 12 describes higher level recursion processing.		

Diagram SUP-23. Program Check First Level Interrupt Handler (IEAVEPCO and IEAVEPC) (Part 11 of 14)



11 Process a recursive interrupt that occurred while handling the initial program interrupt.

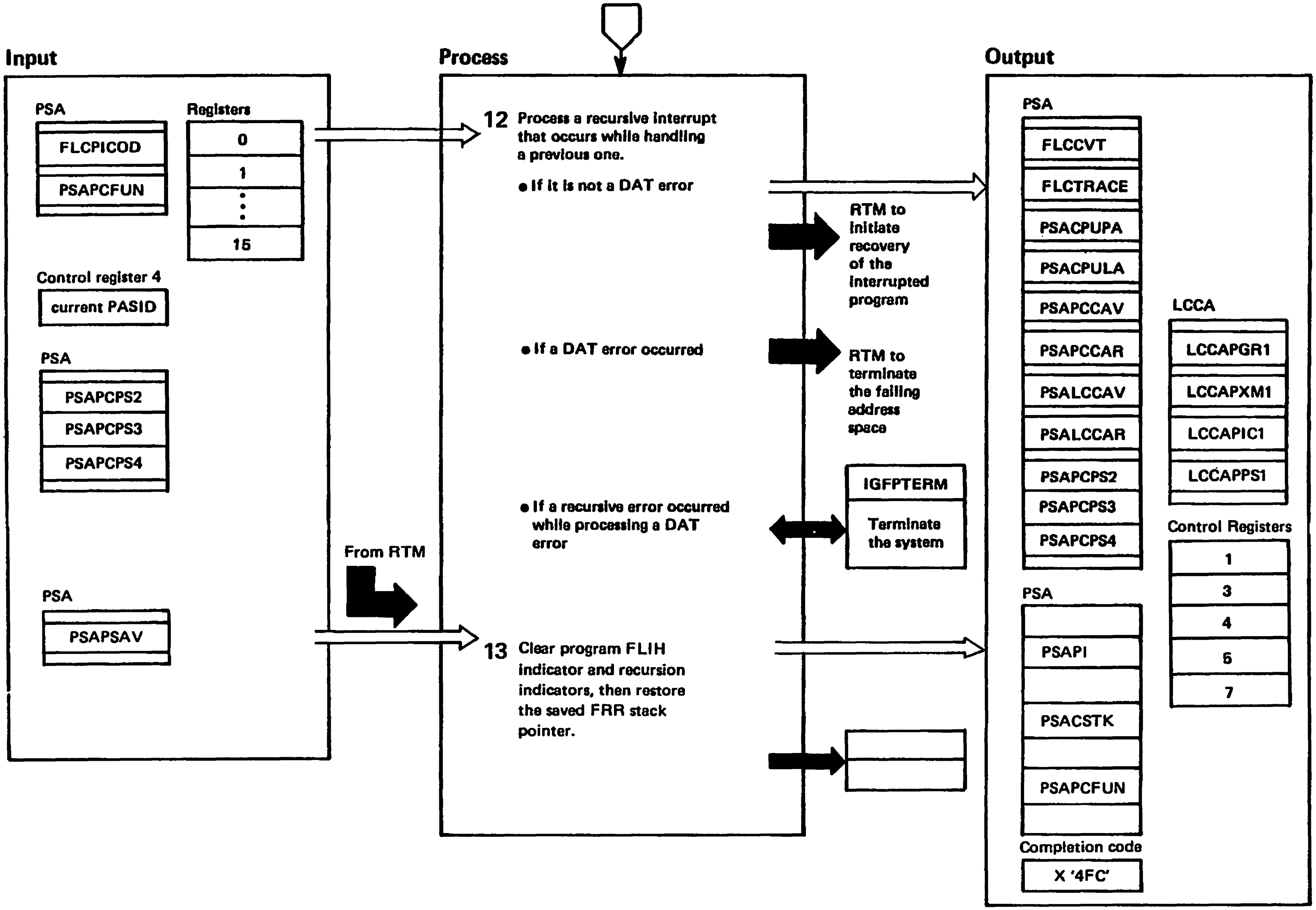
- If a monitor call occurred while handling a page or segment fault

- If the interrupt is any other type

Diagram SUP-23. Program Check First Level Interrupt Handler (IEAVEPCO and IEAVEPC) (Part 12 of 14)

Extended Description	Module	Label
11 The way IEAVEPCO processes a recursive interrupt that occurred while handling an initial program interrupt depends on the types of interrupts involved: • If a monitor call occurred while processing an initial page or segment fault, IEAVEPCO saves the interrupted program's registers in the LCCAPGR3 field and cross memory control registers 3 and 4 in the LCCAPXM3 field. IEAVEPC calls GTF to record the event, then returns control to the program that issued the monitor call. • For all other recursive interrupts IEAVEPCO saves the program interrupt status in the LCCAPRG1 field. The interrupt code and PSW are saved in LCCAPIC1 and LCCAPPS1. If then passes control to IEAVEPC, which traces the recursive interrupt (described in step 4) and calls RTM to initiate recovery for the interrupted program.	IEAVEPCO	

Diagram SUP-23. Program Check First Level Interrupt Handler (IEAVEPCO and IEAVEPC) (Part 13 of 14)



12 Process a recursive interrupt that occurs while handling a previous one.

- If it is not a DAT error
- If a DAT error occurred
- If a recursive error occurred while processing a DAT error

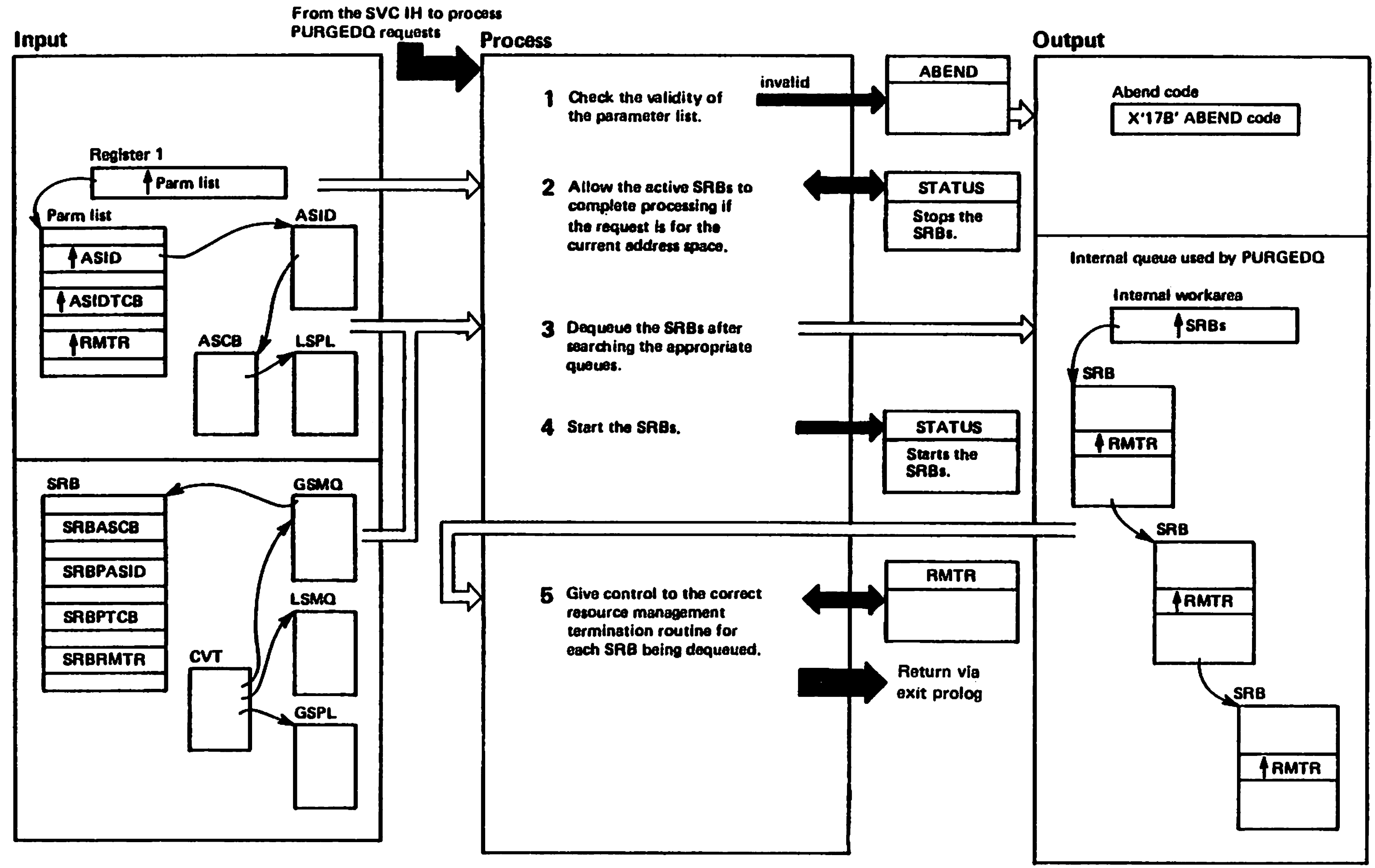
13 Clear program FLIH indicator and recursion indicators, then restore the saved FRR stack pointer.

Diagram SUP-23. Program Check First Level Interrupt Handler (IEAVEPCO and IEAVEPC) (Part 14 of 14)

Extended Description	Module	Label
12 Program interrupts that occur while processing previous recursive program interrupts are considered errors and, except for DAT errors, are handled in much the same way. Unless a DAT error has occurred, the higher level routine (PROGCK2, PROGCK3, or PROGCK4):	PROGCK2, PROGCK3, or PROGCK4	
• Saves the interrupted program's primary ASID in the PSA • Refreshes the PSA fields necessary for continued system operation • Saves the interrupted program's registers in the LCCAPRG1 field and the cross memory status in the LCCAPXM1 field. Saves the recursive interrupt code and PSW in LCCAPIC1 and LCCAPPS1, respectively. • Enables DAT • Traces the interrupt via system trace • Calls RTM to initiate recovery for the interrupted program	DATERR	
Before doing the preceding, PROGCK3 and PROGCK4 compare the primary ASID (PASID) at the time of the interrupt with the previously saved PASID. A match indicates that a DAT error has occurred in that address space. When this happens, IEAVEPCO:		
• Sets the cross memory control registers to address the master scheduler's address space • Refreshes the PSA fields necessary for continued system operation • Saves the interrupted program's registers and cross memory status in the LCCAAPGR1 and LCCAPXM1 fields. Saves the recursive interrupt code and PSW in LCCAPIC1 and LCCAPPS1, respectively. • Enables DAT • When system trace is active, traces the interrupt • Calls RTM to terminate the failing address space		

Extended Description	Module	Label
When a recursive interrupt occurs during DAT error processing, IEAVEPCO terminates the system. It first changes the program check and restart new PSWs to disabled wait PSWs so that any future recursive interrupts will result in the hardware loading a disabled wait PSW. IEAVEPCO then calls IGFPWAIT to terminate the system. IGFPWAIT loads the disabled wait PSW with wait code X'014' and issues termination message IEA999W.	IGFPWAIT	LOADWAIT
13 The program FLIH has a retry routine to which the supervisor FRR (IEAVESPR) will retry if the program FRR stack is current. IEAVEPCR clears the program FLIH indicator (PSAPI) and the FLIH's recursion indicators (PSAPCFUN). The program which received the program interrupt is abended with an X '4FC' completion code.	IEAVEPCR	IEAVEPCR

Diagram SUP-24. PURGEDQ Processing (IEAVEPD0) (Part 1 of 2)



From the SVC IH to process PURGEDQ requests

1 Check the validity of the parameter list.

2 Allow the active SRBs to complete processing if the request is for the current address space.

3 Dequeue the SRBs after searching the appropriate queues.

4 Start the SRBs.

5 Give control to the correct resource management termination routine for each SRB being dequeued.

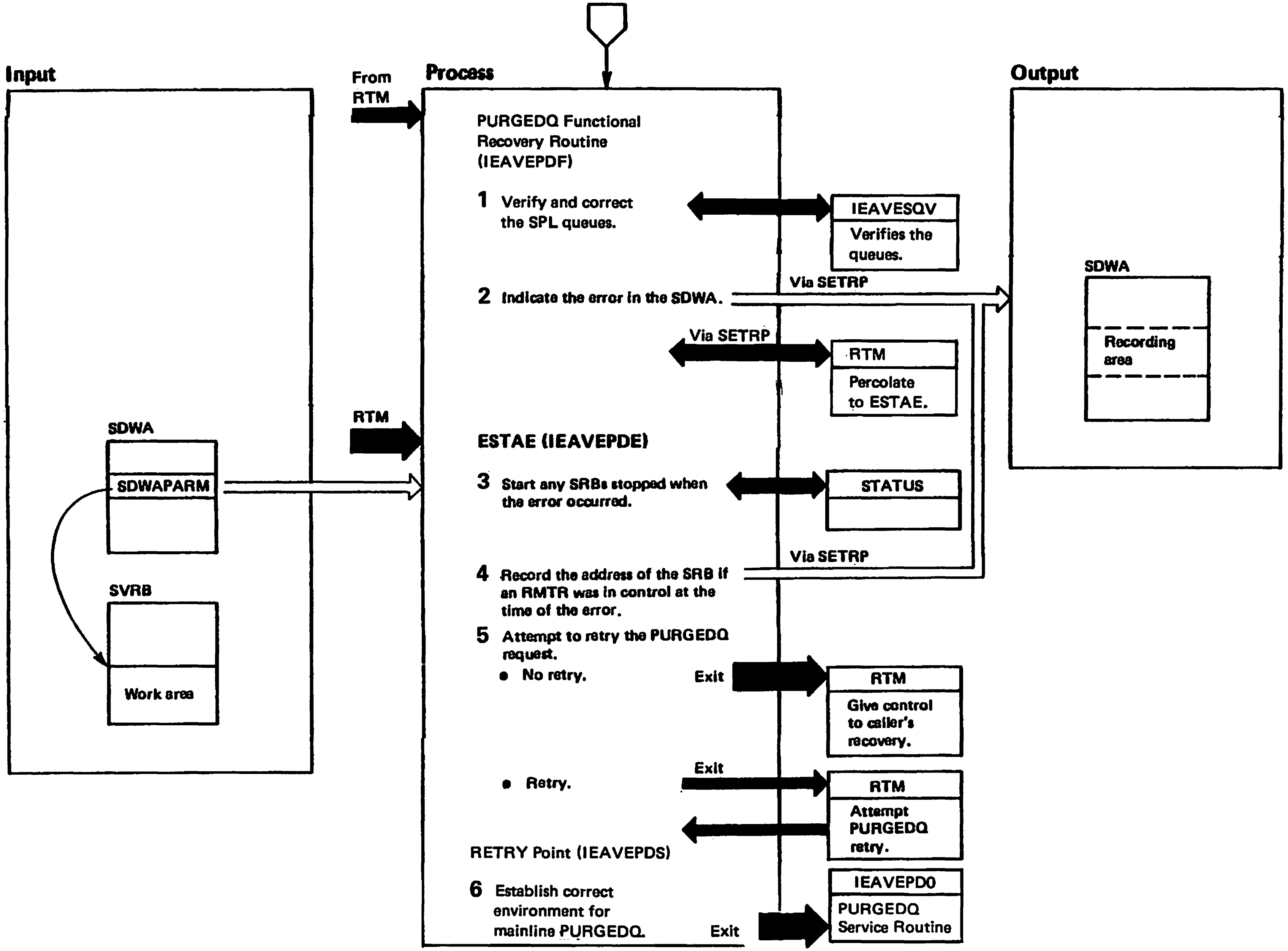
Return via exit prolog

Diagram SUP-24. PURGEDQ Processing (IEAVEPD0) (Part 2 of 2)

Extended Description	Module	Label
Supervisor services use PURGEDQ to cancel SRBs that, for various reasons, should not be executed. The schedule function places SRBs on a queue; and these SRBs execute asynchronously to the schedule request. PURGEDQ cancels SRBs, when necessary.		
1 PURGEDQ terminates callers that have invalid parameter lists.	IEAVEPD0	IGC123
2 PURGEDQ will wait for SRB completion by using the STATUS STOP SRB function. STATUS STOP ensures that SRBs dispatched to the address space have completed.		
PURGEDQ bypasses the waiting operation if the address space specified by the ASID= parameter on the PURGEDQ macro is not the current address space.		
3 PURGEDQ dequeues the SRB by:		
a. Locating the dispatcher queue to be searched. PURGEDQ will search the following queues:		
● Global service management queue (IEAVGSMQ)		
● Global service priority list (IEAGSPL)		
● Local service management queue (IEALSMQ)		
● The local SMQ for the address space specified in the ASID parameter (ASCBLSMQ).		
● The local SPL for the address space specified in the ASID parameter (ASCBLSPL).		
b. Scanning the queues searching for a match on the specified inputs.		
c. Dequeueing those SRBs that match the inputs.		

Extended Description	Module	Label
4 PURGEDQ starts SRBs via STATUS, if they had previously been stopped (in step 2).		
5 PURGEDQ routes control sequentially to the RMTR for each dequeued SRB. When all RMTR routines have been called, PURGEDQ returns via the address passed in register 14 by the caller. This address is established by SVC FLIH to be entry to the exit prolog (IEAVEEXP).		

Diagram SUP-25. PURGEDQ Recovery (IEAVEPDR) (Part 1 of 2)



PURGEDQ Functional Recovery Routine (IEAVEPDF)

1 Verify and correct the SPL queues.

2 Indicate the error in the SDWA.

ESTAE (IEAVEPDE)

3 Start any SRBs stopped when the error occurred.

4 Record the address of the SRB if an RMTR was in control at the time of the error.

5 Attempt to retry the PURGEDQ request.

- No retry.
- Retry.

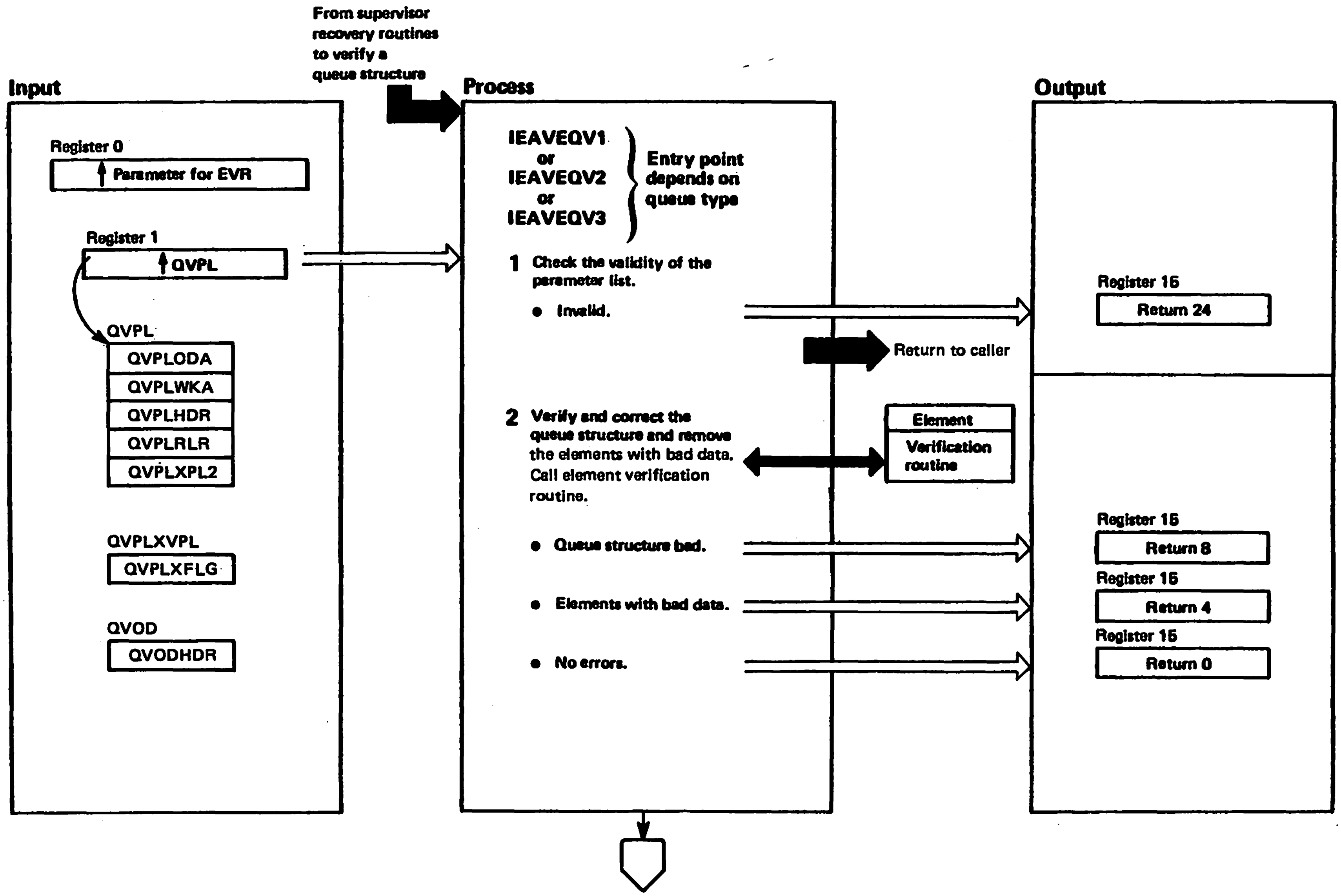
RETRY Point (IEAVEPDS)

6 Establish correct environment for mainline PURGEDQ.

Diagram SUP-25. PURGEDQ Recovery (IEAVEPDR) (Part 2 of 2)

Extended Description	Module	Label
1 PURGEDQ FRR is entered if an error occurred during the queue scanning or updating of the PURGEDQ mainline. The FRR attempts to verify and correct the SPL queues (since bad data on those queues may be causing the errors) by invoking a secondary entry point to the SCHEDULE recovery, IEAVESQV, which performs verification and correction of those queues.	IEAVEPDR	IEAVEPDF
2 Upon receiving control back from that routine, the FRR issues the SETRP macro to set fields in the SDWA for recording information and to indicate that the error should be processed by the PURGEDQ ESTAE routine. It then returns to RTM, which percolates the error to the ESTAE.		
3 The PURGEDQ ESTAE routine receives control if an error occurred anywhere in the PURGEDQ mainline function. It performs cleanup to ensure correct system status. It starts SRBs, via STATUS, if they had been stopped when the error occurred.	IEAVEPDR	IEAVEPDE
4 If an error occurred in an RMTR routine, ESTAE records (in the SDWA) the address of the SRB that the RMTR was cleaning up.		
5 The PURGEDQ ESTAE routine determines if the PURGEDQ function should be retried. It sets up for the retry to the beginning of the PURGEDQ mainline if either this error occurred for the first time during this invocation of PURGEDQ or if the error occurred during the processing of an RMTR routine. If neither of these conditions is true, then the error will be processed by the caller of PURGEDQ.		
6 This retry entry point sets up for the mainline entry point of IEAVEPDO.	IEAVGPDR	IEAVEPDS

Diagram SUP-26. Queue Verification (IEAVEQV0) (Part 1 of 4)



1 Check the validity of the parameter list.

- Invalid.

2 Verify and correct the queue structure and remove the elements with bad data. Call element verification routine.

- Queue structure bad.
- Elements with bad data.
- No errors.

Diagram SUP-26. Queue Verification (IEAVEQV0) (Part 2 of 4)

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

1 IEAVEQV0 verifies and corrects queue structures. It performs validity checking of input parameters to minimize the possibility of the caller incorrectly coding the interface. Queue verification (IEAVEQV0) returns control to the caller immediately with a return code of 24 in register 15 if it detects invalid input parameters. — IEAVEQV0

Setting bit QVPLEXT and initializing the appropriate pointer field in the parameter list, permits the user to specify an Extended Queue Verifer Parameter list (QVPLXVPL). The pointer fields for each queue type are QVPLXPL1, QVPLXPL2, and QVPLXPL3. It contains the address of a 24-byte area used for processing special options.

2 Queue verification corrects queues as follows:

- Single-threaded queues with header only: Since this type of queue contains no duplicate information, queue reconstruction is not possible. Therefore, if any errors in the chaining are found, the queue is truncated at the point of error.
- Single-threaded queues with header and trailer: For this type of queue, the end of the queue found by scanning the forward chain might not coincide with the value in the trailer. In general, if the trailer contains the address of a queue element, that element is considered the "real" last element.

  If the header has been destroyed, queue verification tries to salvage the element pointed to by the trailer.

  If the trailer has been destroyed, it is restored from the forward chain.

  If a forward chain pointer has been destroyed, all the previous elements on the chain will be connected to the element pointed to by the trailer.
- Double-threaded queues: If the header and trailer contain addresses of elements, those elements are considered the real first and last elements, respectively.

  As long as the forward chain is valid, it has precedence over the backward chain. (When scanning the forward chain, the next element should always point back to the current. If it does not, the backward pointer will be corrected.)

Extened Description **Module** **Label**

  If the header is bad, it is restored from the backward chain.

  If the trailer is bad, it is restored from the forward chain.

  If either the forward or backward chain is bad, one is reconstructed from the other. If both are bad, they are connected at their last valid points.
- All types of queues: The queue verification detects circular queues. The last element found before the queue repeats is considered the last good element on the chain.

  All elements that contain bad data, as defined by a return code of 4 from the element verification routine, will be removed from the queue.
- Circularity checking: Queue verification detects circular queues. The last element found before the queue repeats is considered the last good element on the chain.
- Single-threaded queues: If the caller of queue verification uses the extended queue verification parameter list, the caller can set bit QVPLXM and specify a maximum number of expected elements in QVPLXMAX. IEAVEQVO suppresses the circularity check until the count of elements exceeds this maximum. This option significantly reduces the path length to verify a correct queue.
- Double-threaded queues: It is logically impossible for two elements of a double-threaded queue to have a circular relationship if the forward and backward pointers agree. Therefore, when scanning the forward chain, if the next element points back to the current, the circularity check is supressed. Similarly, when scanning the backward chain, if the next element points forward to the current, the circularity check is supressed.
- Removal of elements with bad data: The queue is scanned one time to verify the structure of the queue and if this scan is not perfect, the queue is scanned a second time to verify the data within the elements.

A return code of 8 from the event verification routine (EVR) indicates that the pointer to the current element is invalid and the structure of the queue is damaged. A return code of 4 from the EVR indicates that the current element contains bad data. All elements that contain bad data will be removed from the queue. If the first scan of the queue is perfect, the second scan is supressed.

Diagram SUP-26. Queue Verification (IEAVEQV0) (Part 3 of 4)

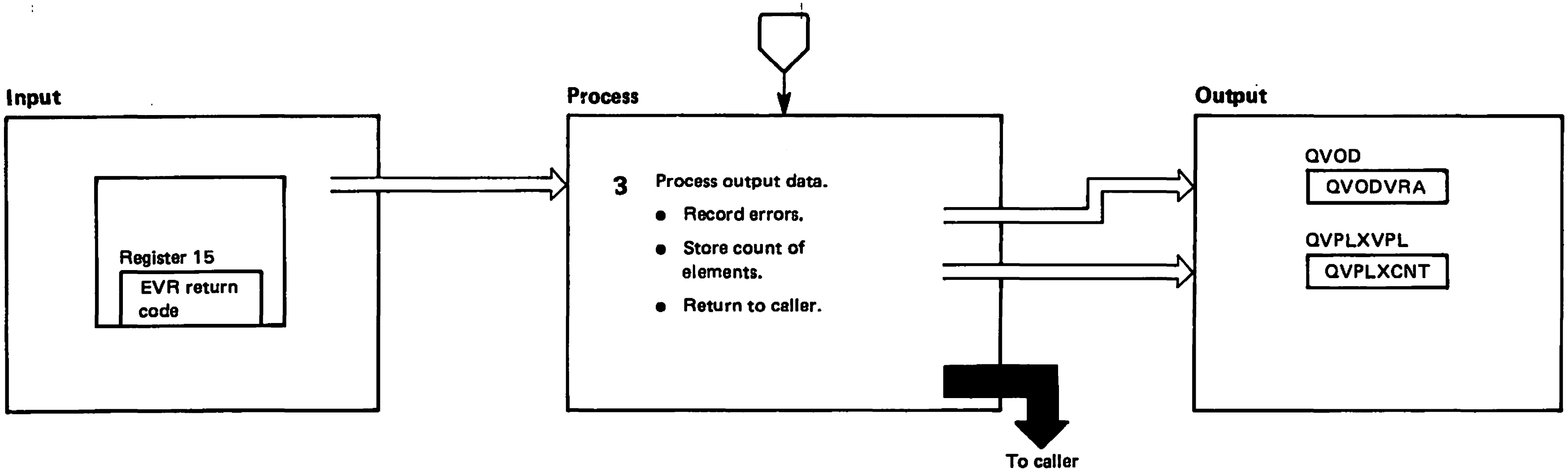


Diagram SUP-26. Queue Verification (IEAVEQV0) (Part 4 of 4)

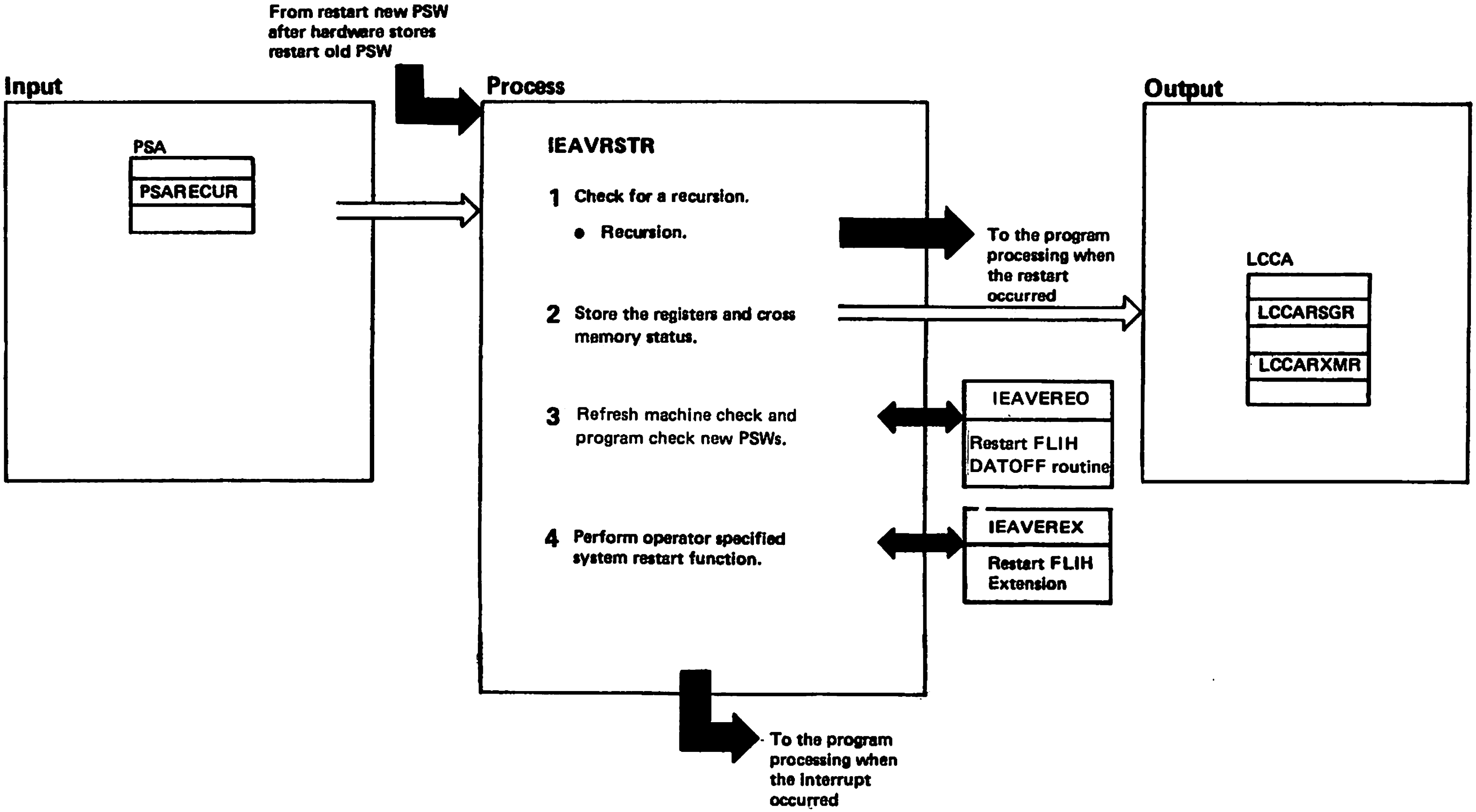
Extended Description	Module	Label

3 All errors encountered are recorded in the queue verification output data area (QVOD). The QVOD maps into the recording area of the SDWA. Generally, the following information will be supplied.

- Error code, describing the specific error.
- If an element had bad chain information, then the address of the element, the old (bad) chain information, and the new (corrected) information are recorded.
- If an element was removed because it contained bad data, then the address of the element, the address of the previous element on the queue, and the address of the next element on the queue are recorded.

If the caller of queue verification is using the extended queue verification parameter list, a count of good elements may be requested by setting the QVPLXC bit. Before returning to its caller, IEAVEQV0 stores the count of good elements into QVPLXCNT.

Diagram SUP-27. Restart Interruption Handler (IEAVERES) (Part 1 of 2)



From restart new PSW after hardware stores restart old PSW

Input

PSA

PSARECUR

Process

IEAVRSTR

1 Check for a recursion.

- Recursion.

To the program processing when the restart occurred

2 Store the registers and cross memory status.

3 Refresh machine check and program check new PSWs.

IEAVEREO

Restart FLIH DATOFF routine

4 Perform operator specified system restart function.

IEAVEREX

Restart FLIH Extension

To the program processing when the interrupt occurred

Output

LCCA

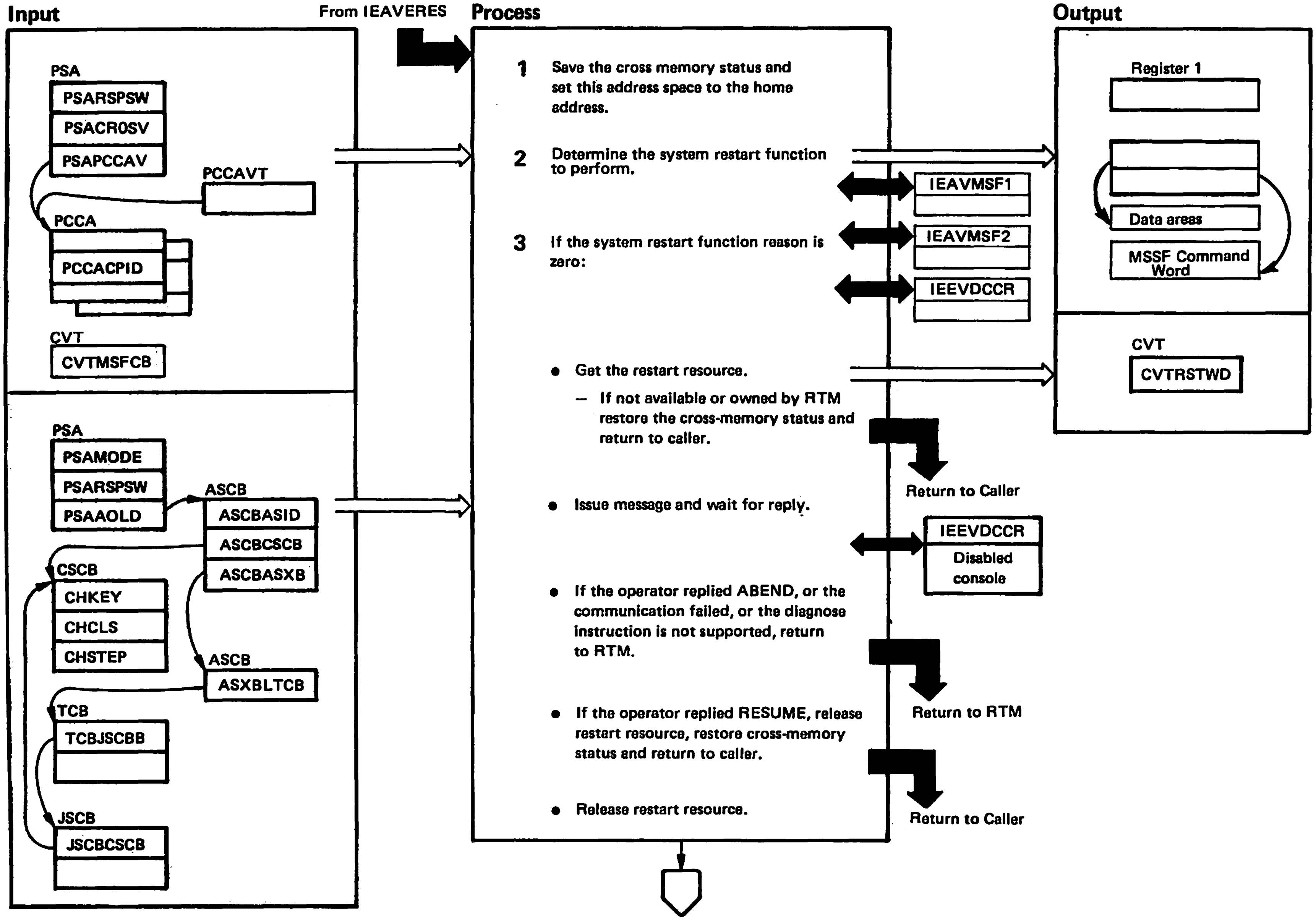
LCCARSGR

LCCARXMR

Diagram SUP-27. Restart Interruption Handler (IEAVERES) (Part 2 of 2)

Extended Description	Module	Label
The restart interruption handler (IH) routes control to recovery termination after the operator uses the system restart function or a routine issues a restart SIGP instruction.		
1 The restart IH ignores recursive entries by giving control back to the interrupted program. Otherwise, normal processing continues.	IEAVERES	IEAVRSTR
2 The restart IH: ● Saves the general registers in the LCCARSGR field ● Saves cross memory control registers 3 and 4 in the LCCARXMR field.		
3 The restart FLIH DATOFF routine refreshes the machine-check new PSW and the program new PSW.	IEAVEREO	IEAVEREO
4 The restart interruption handler extension (IEAVEREX) will use the specified system restart function reason (on the system control operator's frame) to determine further processing, interfacing with CALLRTM TYPE=RESTART, if necessary. The extension routine will acquire and release the restart resource as needed.	IEAVEREX	IEAVEREX
Recovery Processing		
The restart FRR (functional recovery routine) clears the restart interruption indicator in PSARECUR, sets the restart resource in the CVTRSTWD field of the CVT to zero, and points the FRR pointer (PSACSTK) to the normal FRR stack. It then terminates the program executing when the interruption occurred with a X'5FC' completion code.	IEAVERER	IEAVERER

Diagram SUP-28. Restart Interruption Handler Extension (IEAVEREX) (Part 1 of 4)



1 Save the cross memory status and set this address space to the home address.

2 Determine the system restart function to perform.

3 If the system restart function reason is zero:

- Get the restart resource.
  - If not available or owned by RTM restore the cross-memory status and return to caller.
- Issue message and wait for reply.
- If the operator replied ABEND, or the communication failed, or the diagnose instruction is not supported, return to RTM.
- If the operator replied RESUME, release restart resource, restore cross-memory status and return to caller.
- Release restart resource.

Diagram SUP-28. Restart Interruption Handler Extension (IEAVEREX) (Part 2 of 4)

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

The restart interruption handler (IH) extension (IEAVEREX) processes the restart interrupt called by IEAVERES. Depending on the system restart function reason, IEAVEREX either terminates the interrupted program and invokes the necessary recovery routines (0) or performs high-level system diagnosis not related to the work currently being processed (1).

1 IEAVEREX saves the cross memory status of the interrupted program and sets this address space to the home address space (PSAAOLD).

2 IEAVEREX checks the CVTMSFCB field to see if the MSSF is available. IEAVEREX checks the PCCACPID field of all processors to determine that no processor is running under VM. IEAVEREX obtains the restart reason by an MSSFCALL DIAGNOSE instruction issued by MSSFCALL SVC routine through branch entry IEAVMSF1 or IEAVMSF2. IEAVEREX uses IEAVMSF4 if the interrupted routine is enabled for EMS, MFA and MSSF external interrupts and use as IEAVMSF2 in all other cases. If the MSSFCALL fails, IEAVEREX issues message IEA502A via module IEEVDCCR to prompt the operator for a restart reason value. IEAVEREX stores the operator-entered value of the system restart function in the system restart processor-related work area.

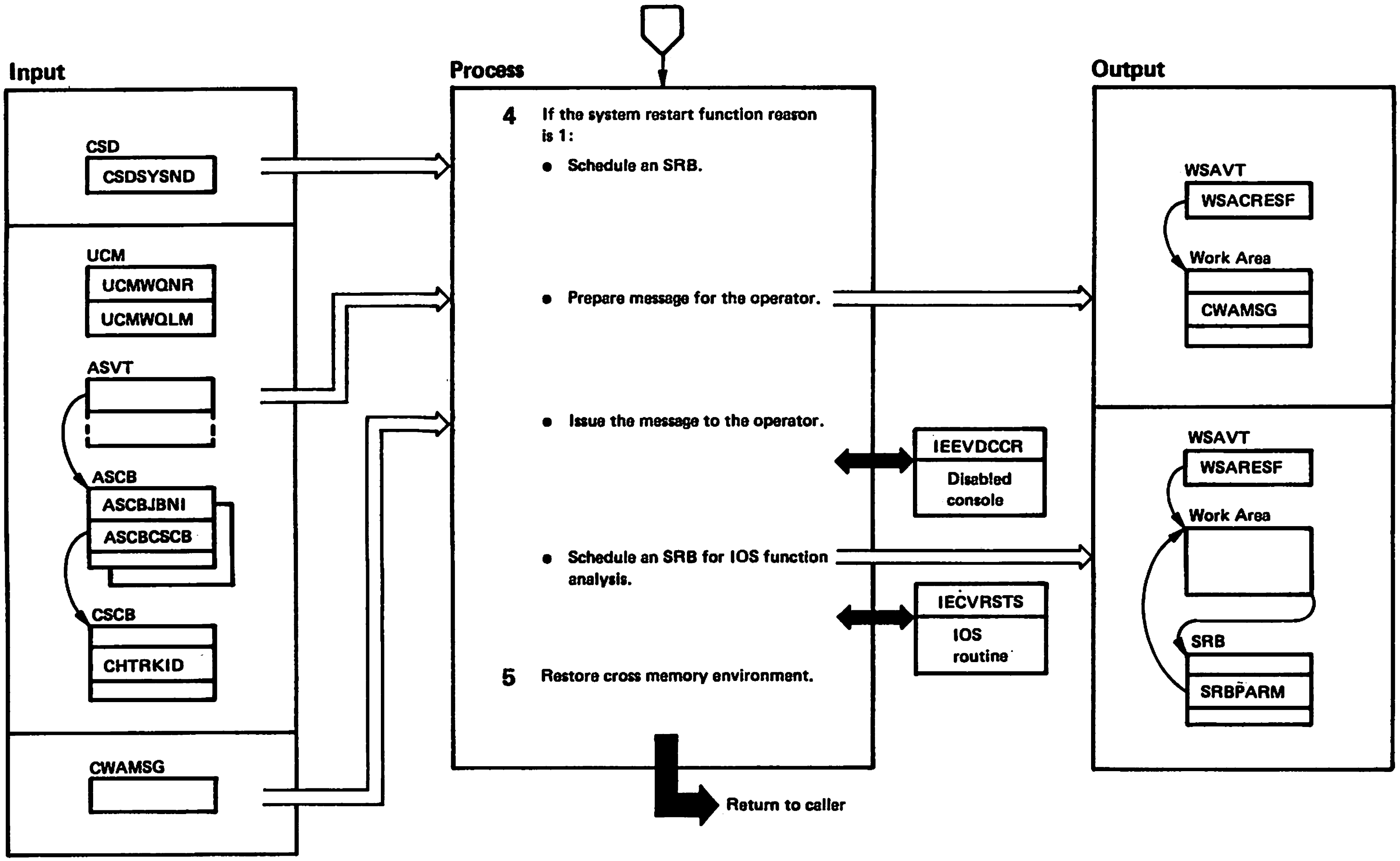
3 If the system restart function reason is zero, IEAVEREX performs as follows:

- The restart resource must be obtained before entry to RTM. If the restart lock (CVRSTWD) equals zero, or is owned by RTM on this CPU (the CPU ID and function code 'RF') or is being passed to IEAVEREX by RTM (the CPU ID and function code 'RP') IEAVEREX acquires the restart resource (set to the CPU ID and function code 'RF'). If the resource cannot be acquired, IEAVEREX resumes the interrupted program by returning to IEAVERES.
- IEAVEREX issues message IEA500A to the operator via module IEEVDCCR and waits for a reply of "ABEND" or "RESUME". IEEVDCCR

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

- If the operator replied "ABEND" or if the operator communications failed, IEAVEREX restores the FRR stack and goes to RTM (CALLRTM TYPE=RESTART).
- If the operator replied "RESUME", IEAVEREX ends further restart processing and restores the cross memory status. IEAVEREX then returns to IEAVERES.
- IEAVEREX releases the restart resource if acquired.

Diagram SUP-28. Restart Interruption Handler Extension (IEAVEREX) (Part 3 of 4)



4 If the system restart function reason is 1:

- Schedule an SRB.
- Prepare message for the operator.
- Issue the message to the operator.
- Schedule an SRB for IOS function analysis.

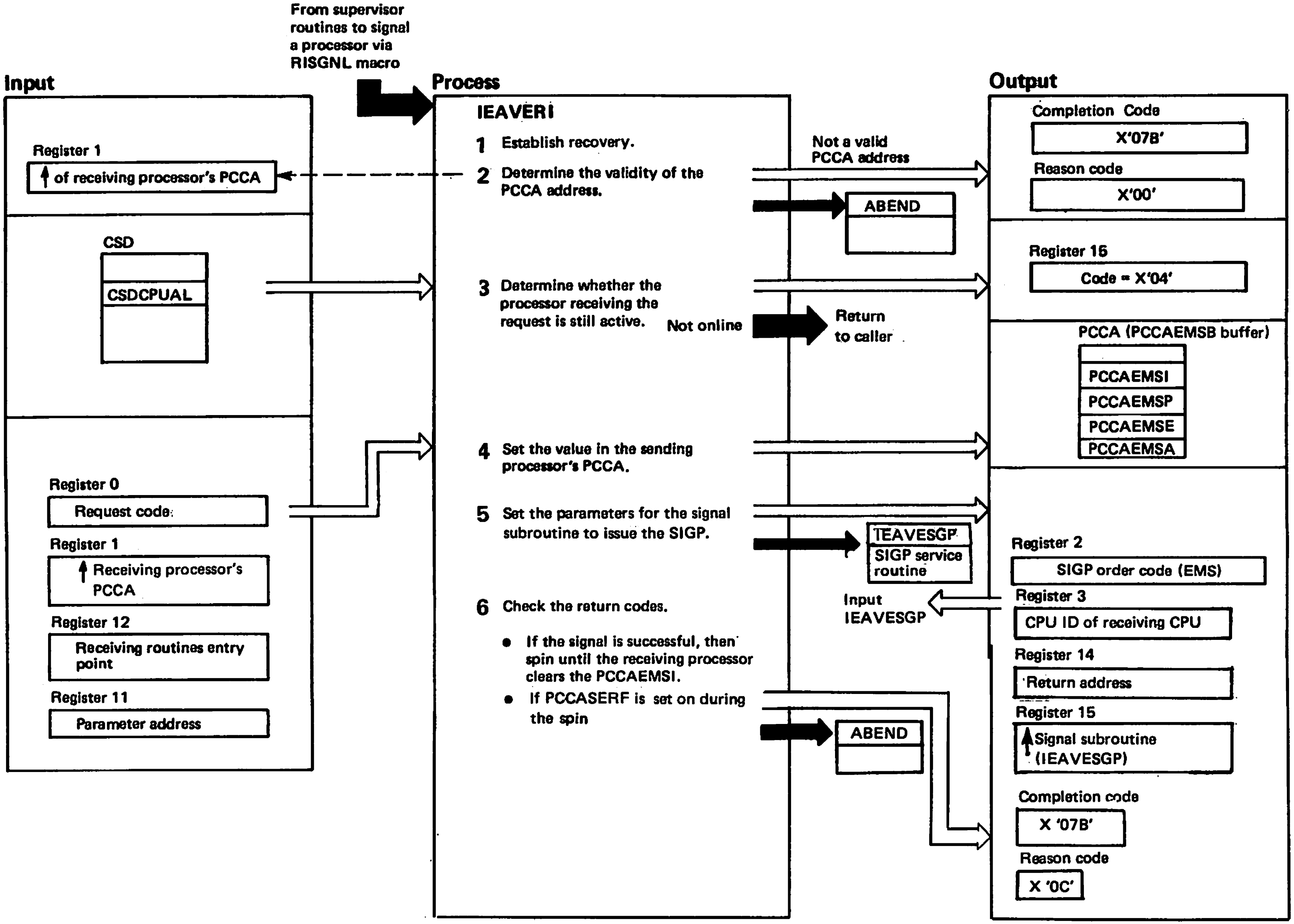
5 Restore cross memory environment.

Return to caller

Diagram SUP-28. Restart Interruption Handler Extension (IEAVEREX) (Part 4 of 4)

Extended Description	Module	Label
4 If the system restart function reason is 1, IEAVEREX performs the necessary system diagnostics and repair actions.		
• If the system is nondispatchable (CSDSYSND=1), IEAVEREX schedules an SRB (global, nonquiesceable) in the master address space and resets the system status (NDISPSRB). IEAVEREX issues a Test and Set (TS) instruction to serialize the SRB. This allows the SRB routine to receive control with the local lock held. IEAVEREX formats the first line of message IEA501I.		
• IEAVEREX adds text to message IEA501I as follows.		
– If the limitations of the WTO buffer have been exceeded (that is, the contents of UCMWQNR is greater than the contents of UCMWQLM).		
– If neither a batch job or a timesharing user is found (for each active ASCB (located via the ASVT), IEAVEREX checks the CSCB for a batch job (ASCBJBNI=0) or a timesharing user (CHTRKID=X'01').		
• IEAVEREX notifies the operator of anything it repairs or diagnoses by issuing message IEA501I (via IEEVDCCR).	IEEVDCCR	
• IEAVEREX serializes the global work area (including the SRB). The first byte of the work area is the serializing byte. IEAVEREX invokes the IECVRSTS routine for IOS functional analysis.		
5 IEAVEREX restores the cross memory environment using information stored in the processor-related work area. IEAVEREX returns to IEAVERES to resume the interrupted program.		

Diagram SUP-29. Interprocessor Communications (IPC) Remote Immediate Signal Routine (IEAVERI) (Part 1 of 4)



1 Establish recovery.

2 Determine the validity of the PCCA address.

3 Determine whether the processor receiving the request is still active.

4 Set the value in the sending processor's PCCA.

5 Set the parameters for the signal subroutine to issue the SIGP.

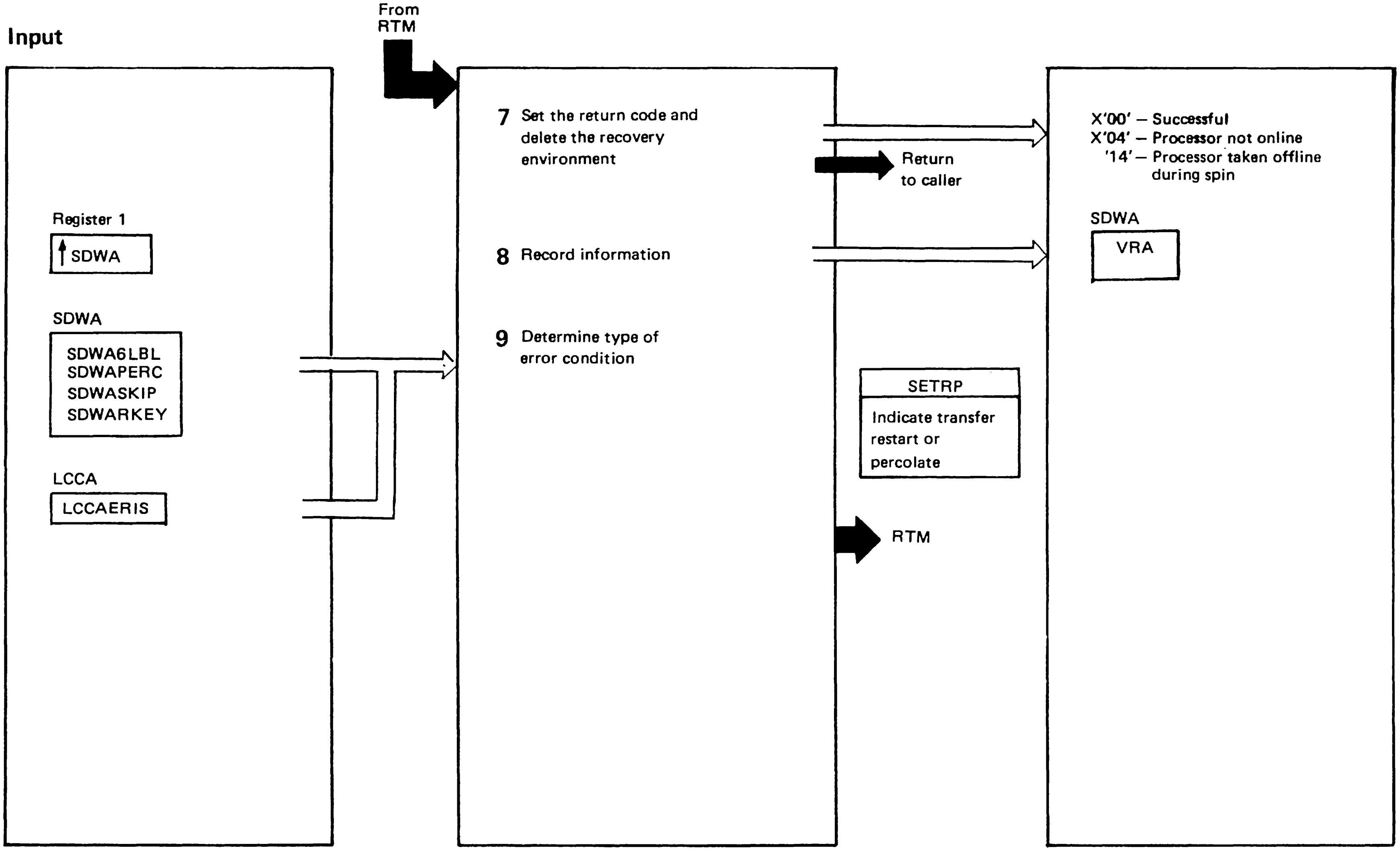
6 Check the return codes.

- If the signal is successful, then spin until the receiving processor clears the PCCAEMSI.
- If PCCASERF is set on during the spin

Diagram SUP-29. Interprocessor Communications (IPC) Remote Immediate Signal Routine (IEAVERI) (Part 2 of 4)

Extended Description	Module	Label
1 The remote immediate signal routine on the sending processor establishes the interface to cause a specific program to get control on a specific receiving processor.		
IEAVERI causes an emergency signal to a specified processor. The emergency signal SLIH (IEAVEES) on the receiving processor routes control to the program.		
2 IEAVERI determines the validity of the PCCA (physical configuration communication area) address. If the PCCA address is invalid, IEAVERI ABENDS the caller with a completion code of X'07B' and a reason code of X'00'. Otherwise, normal processing follows.	IEAVERI	IEAVERI
3 IEAVERI returns control to the caller if the receiving processor is not online, with a return code of 4 in register 15.		
4 IEAVERI sets the request type entry point address, PCCA address of the receiving routine, and parameter address in the PCCAEMSB field of the sending processor's PCCA.		
5 IEAVERI sets the input values for the signal routine (IEAVESGP), which actually issues the SIGP instruction.		
6 IEAVESGP passes back a return code based on the condition code of the SIGP instruction. (See IEAVESGP.) IEAVERI uses this return code to determine the success of the signal.		
For serial or parallel requests, if the signal was successful, IEAVERI spins until the receiving processor clears the PCCAEMSI field.		
If the signal is not accepted by the receiving CPU after spinning for a reasonable length of time (i.e., the PCCAEMSI field is not cleared), IEAVERI resets the input values for the signal routine and invokes IEAVESGP to reissue the signal. Otherwise, IEAVERI continues spinning. If the PCCAEMSI field is still not cleared after the second spin, IEAVERI calls the excessive spin notification routine (IEEVEXSN), which issues message IEE331A to notify the operator of the spin. If the receiving processor remains online, IEAVERI continues to spin. If the receiving routine for a serial request fails, the receiving processor does not clear PCCAEMSI and indicates the failure (sets PCCASERF to one). IEAVERI abends the caller with an X'073' and a reason code of X'0C'.		

Diagram SUP-29. Interprocessor Communications (IPC) Remote Immediate Signal Routine (IEAVERI) (Part 3 of 4)



Input

From RTM

Register 1

SDWA

SDWA

SDWA6LBL
SDWAPERC
SDWASKIP
SDWARKEY

LCCA

LCCAERIS

7 Set the return code and delete the recovery environment

8 Record information

9 Determine type of error condition

Return to caller

SETRP

Indicate transfer restart or percolate

RTM

X'00' – Successful
X'04' – Processor not online
'14' – Processor taken offline during spin

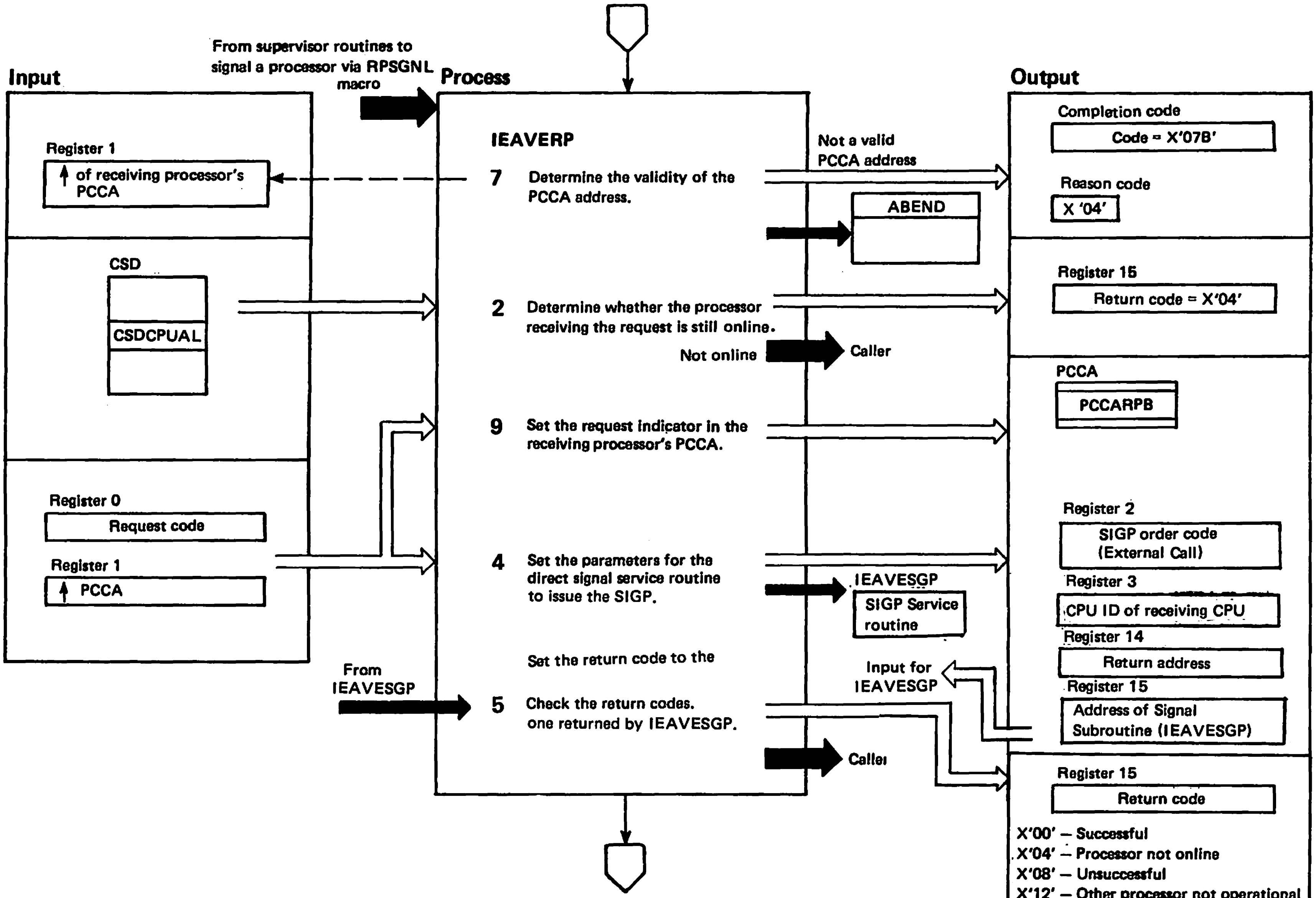
SDWA

VRA

Diagram SUP-29. Interprocessor Communications (IPC) Remote Immediate Signal Routine (IEAVERI) (Part 4 of 4)

Extended Description	Module	Label
7 The caller receives a return code, indicating the status of the request, from the remote immediate routine. If the requested function is successful, the return code is X'00'. For a return code of X'04' the specified processor is not online and no signal was issued. If the processor is taken offline during the spin, the return code is X'14'.		
8 Indicates error information in the SQWA and VRA areas.		
9 Depending on the reason entered, uses the SETRP macro to indicate to RTM to either transfer a restart or to percolate.		

Diagram SUP-30. Interprocessor Communications (IPC) Remote Pendable Signal Routine (IEAVERP) (Part 1 of 2)



7 Determine the validity of the PCCA address.

2 Determine whether the processor receiving the request is still online.

9 Set the request indicator in the receiving processor's PCCA.

4 Set the parameters for the direct signal service routine to issue the SIGP.

Set the return code to the

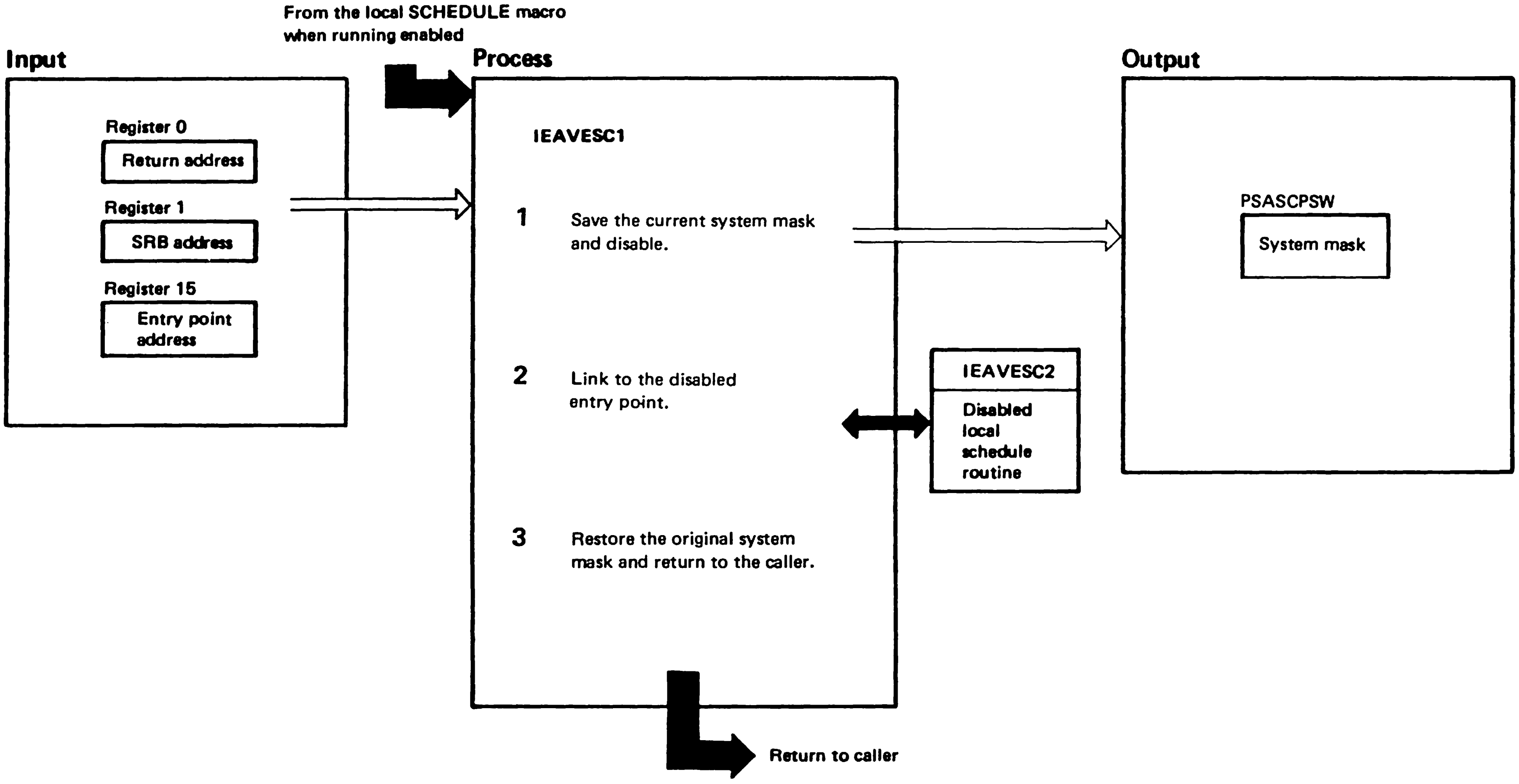
5 Check the return codes.
one returned by IEAVESGP.

X'00' – Successful
X'04' – Processor not online
X'08' – Unsuccessful
X'12' – Other processor not operational
X'16' – Uniprocessor system

Diagram SUP-30. Interprocessor Communications (IPC) Remote Pendable Signal Routine (IEAVERP) (Part 2 of 2)

Extended Description	Module	Label
1 The remote pendable signal routine on the sending processor establishes the interface to cause a specific system routine to get control on a specific receiving processor. IEAVERP causes an external call signal to a specified processor. The external call SLIH (IEAVEXS) on the receiving processor routes control to the program. IEAVERP determines the validity of the PCCA address. The remote pendable routine abends the caller if it finds the PCCA address invalid. The caller receives a X'07B' ABEND with an X'04' reason code. Otherwise, normal processing follows.	IEAVERP	
2 The remote pendable routine determines whether the processor receiving the request is still online. Control goes back to the caller, with a return code of 4 in register 15, if the receiving processor is not online. Otherwise, normal processing continues.		
3 Next, the remote pendable routine sets the request code in the PCCARPB field of the receiving processor's PCCA.		
4 The remote pendable signal routine sets the input values for the signal routine, which actually issues the SIGP instruction with an external call order code.		
5 The remote signal routine returns to the caller with the code it received from IEAVESGP in register 15 indicating the status of the request. On return from the signal routine, either the signal has been sent to the specified receiving processor (set the return code to X'00') or the receiving processor is no longer online (set the return code to X '04').		

Diagram SUP-31. SCHEDULE Processing (IEAVESC0) (Part 1 of 22)



1 Save the current system mask and disable.

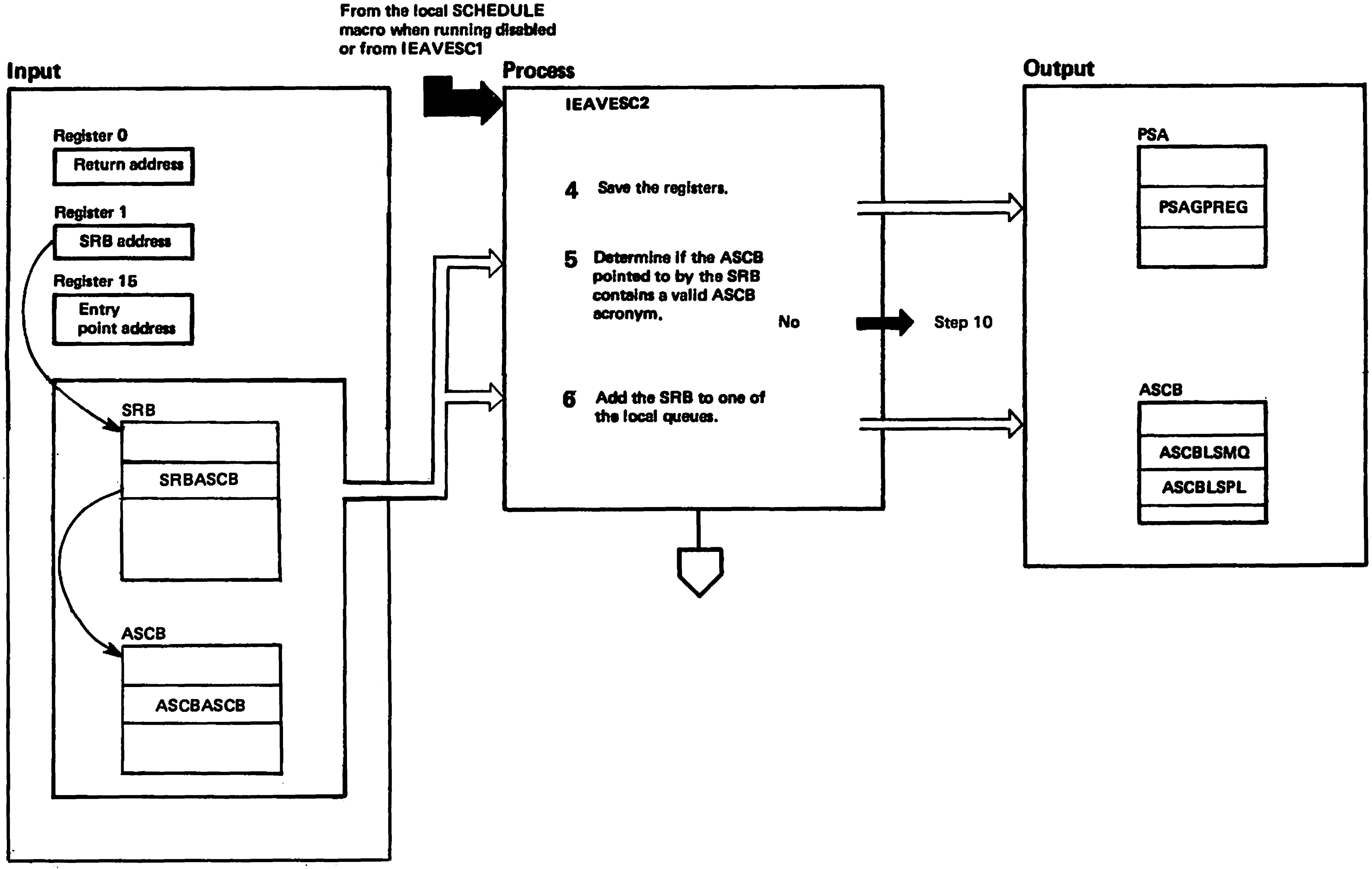
2 Link to the disabled entry point.

3 Restore the original system mask and return to the caller.

Diagram SUP-31. SCHEDULE Processing (IEAVESC0) (Part 2 of 22)

Extended Description	Module	Label
This routine is entered (using a BALR) from the SCHEDULE macro when an enabled issuer requests that an SRB be placed on a local SRB queue.		
The ASCB contains a local service management queue (LSMQ) and a local service priority list (LSPL). This routine determines which queue the SRB should be added to.		
1-3 This routine disables for I/O and external interrupts and passes control to the disabled local entry point (IEAVESCR) to do the schedule processing. This routine restores the caller's original system mask prior to returning control to the caller.	IEAVESC0	IEAVESC1

Diagram SUP-31. SCHEDULE Processing (IEAVESC0) (Part 3 of 22)



4 Save the registers.

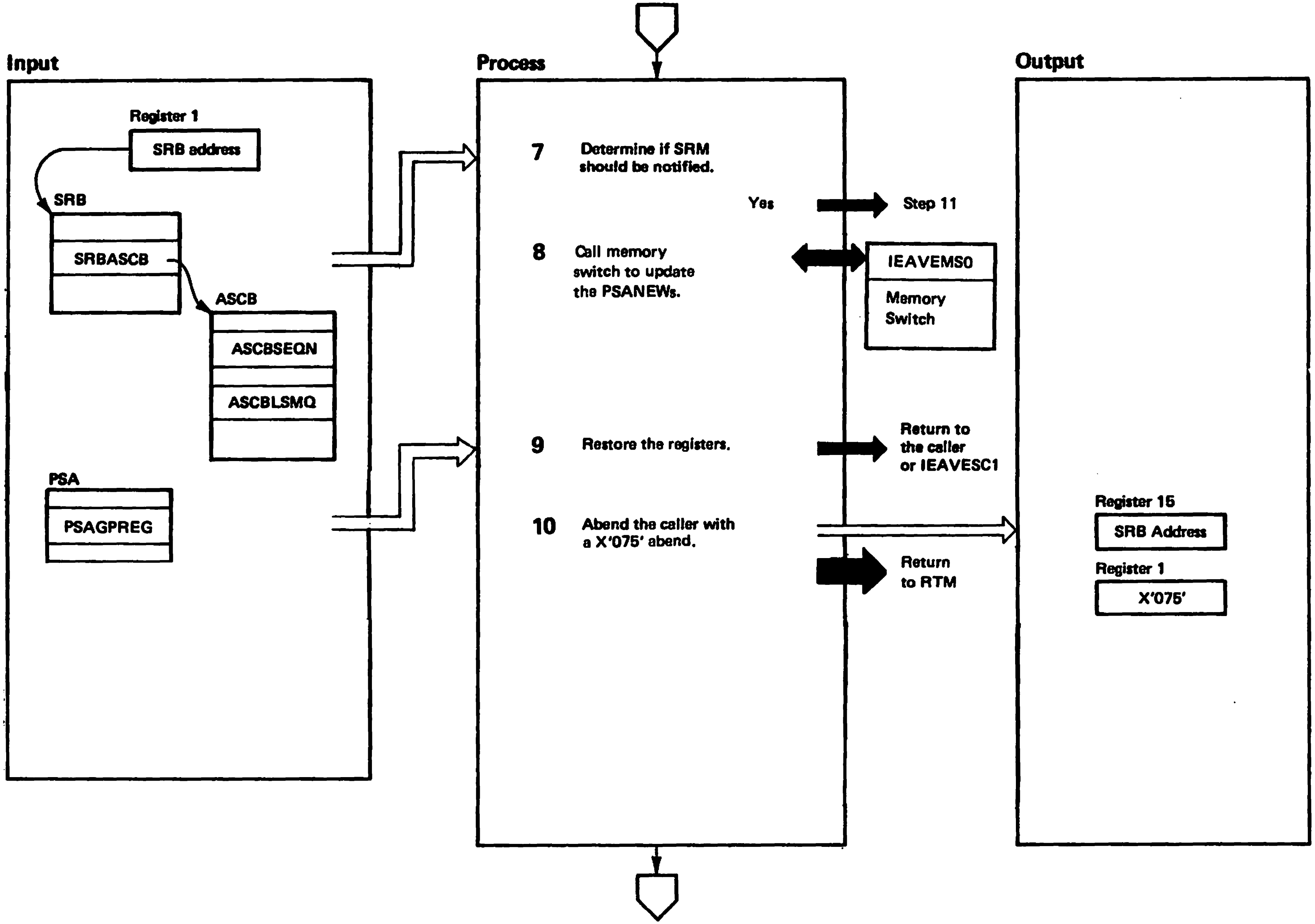
5 Determine if the ASCB pointed to by the SRB contains a valid ASCB acronym.

6 Add the SRB to one of the local queues.

Diagram SUP-31. SCHEDULE Processing (IEAVESC0) (Part 4 of 22)

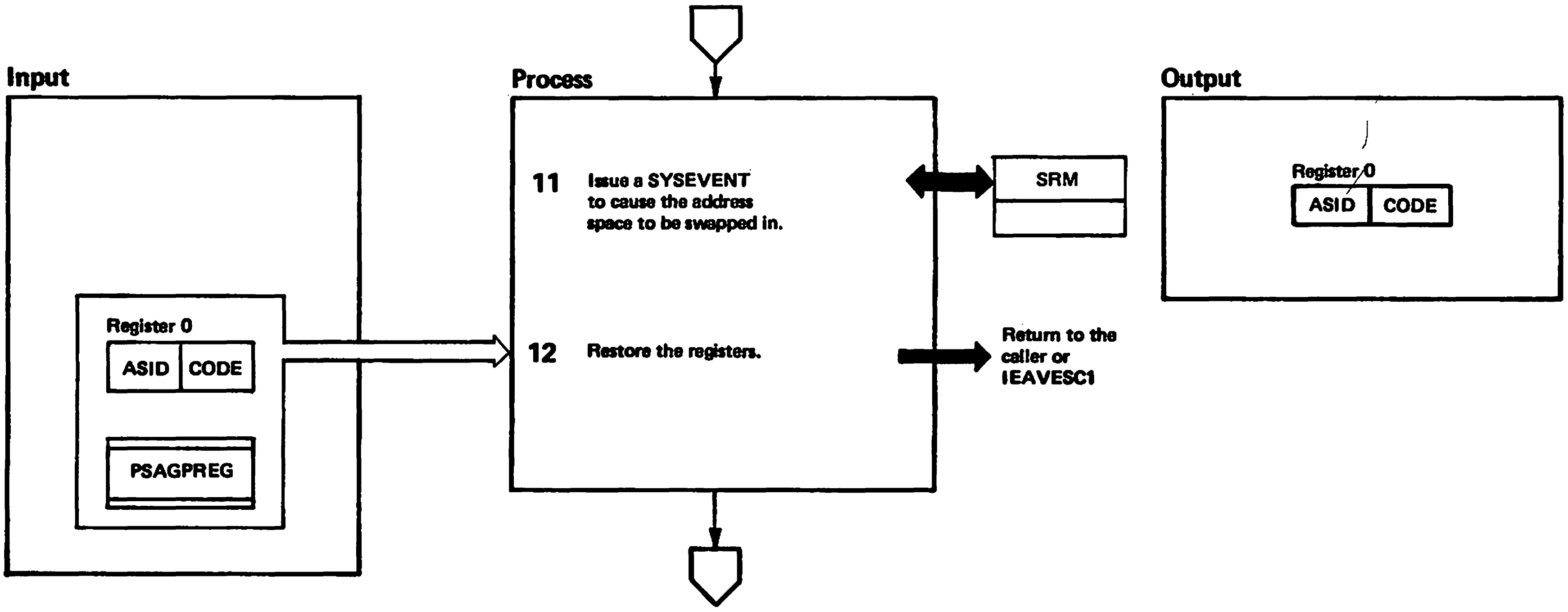
Extended Description	Module	Label
This routine is entered from the SCHEDULE macro (via BALR) or from IEAVESC1 (via BASR) when a disabled issuer requests that an SRB be placed on a local SRB queue or from the enabled local entry point (IEAVESC1).		
4 Save registers 12, 13 and 14 in the PSAGPREG.	IEAVESC0	IEAVESC2
5 If the ASCB pointed to by the SRB does not contain a valid ASCB acronym, control will be passed to step 10 and the caller will be abended.		
6 The SRB is added to either the ASCBLSMQ or the ASCBLSPL. If both queues are empty, the SRB is placed on the ASCBLSPL; otherwise the SRB is placed on the ASCBLSMQ.		

Diagram SUP-31. SCHEDULE Processing (IEAVESC0) (Part 5 of 22)



Extended Description	Module	Label
7 If ASCBURR (the high-order bit of ASCBLSMQ) is set to 1, then notify the system resource manager (SRM), using SYSEVENT (step 11), to swap-in the address space in which the SRB will run.		
8 IEAVESC0 invokes the memory switch service (IEAVEMS0) with the ASCB address that has ready work as input. Memory switch looks for a processor eligible to run the ready work. If an eligble processor is found, PSANEW for that processor is updated. (Memory switch is described in the DISP section of the *System Logic Library.)*		
9 Restore the registers and return control to IEAVESC1 or the disabled caller.		
10 Abend the caller.		

Diagram SUP-31. SCHEDULE Processing (IEAVESC0) (Part 7 of 22)



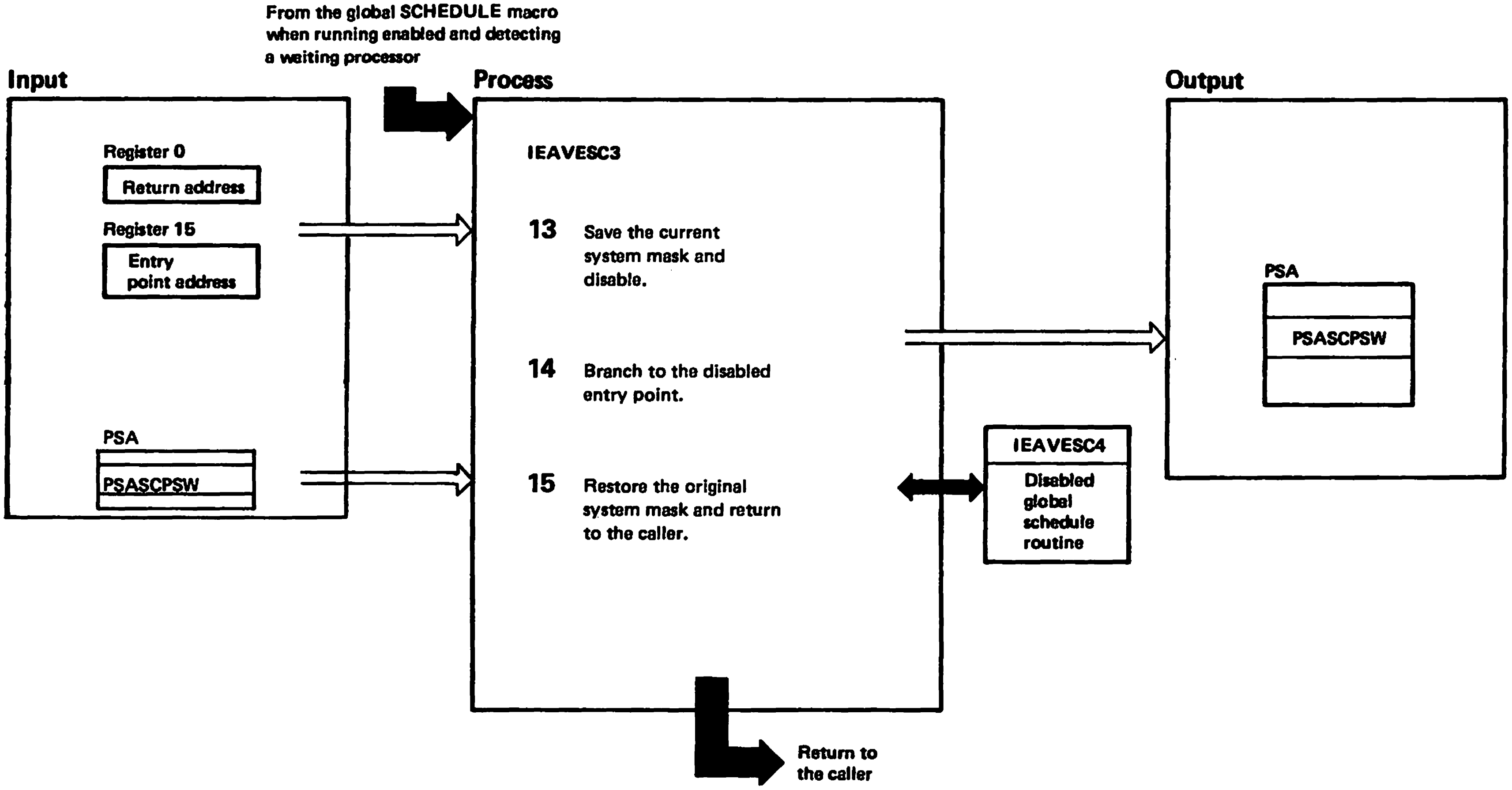
11 Issue a SYSEVENT to cause the address space to be swapped in.

12 Restore the registers.

Diagram SUP-31. SCHEDULE Processing (IEAVESC0) (Part 8 of 22)

Extended Description	Module	Label
11 SRM (system resource manager) is notified, using SYSEVENT (USERRDY option), that there is ready work for a swapped-out address space. This will cause a swap-in of that address space.		
12 The saved registers 12, 13 and 14 are restored from field PSAGPREG and control is returned to IEAVESC1 or the disabled caller.		

Diagram SUP-31. SCHEDULE Processing (IEAVESC0) (Part 9 of 22)



13 Save the current system mask and disable.

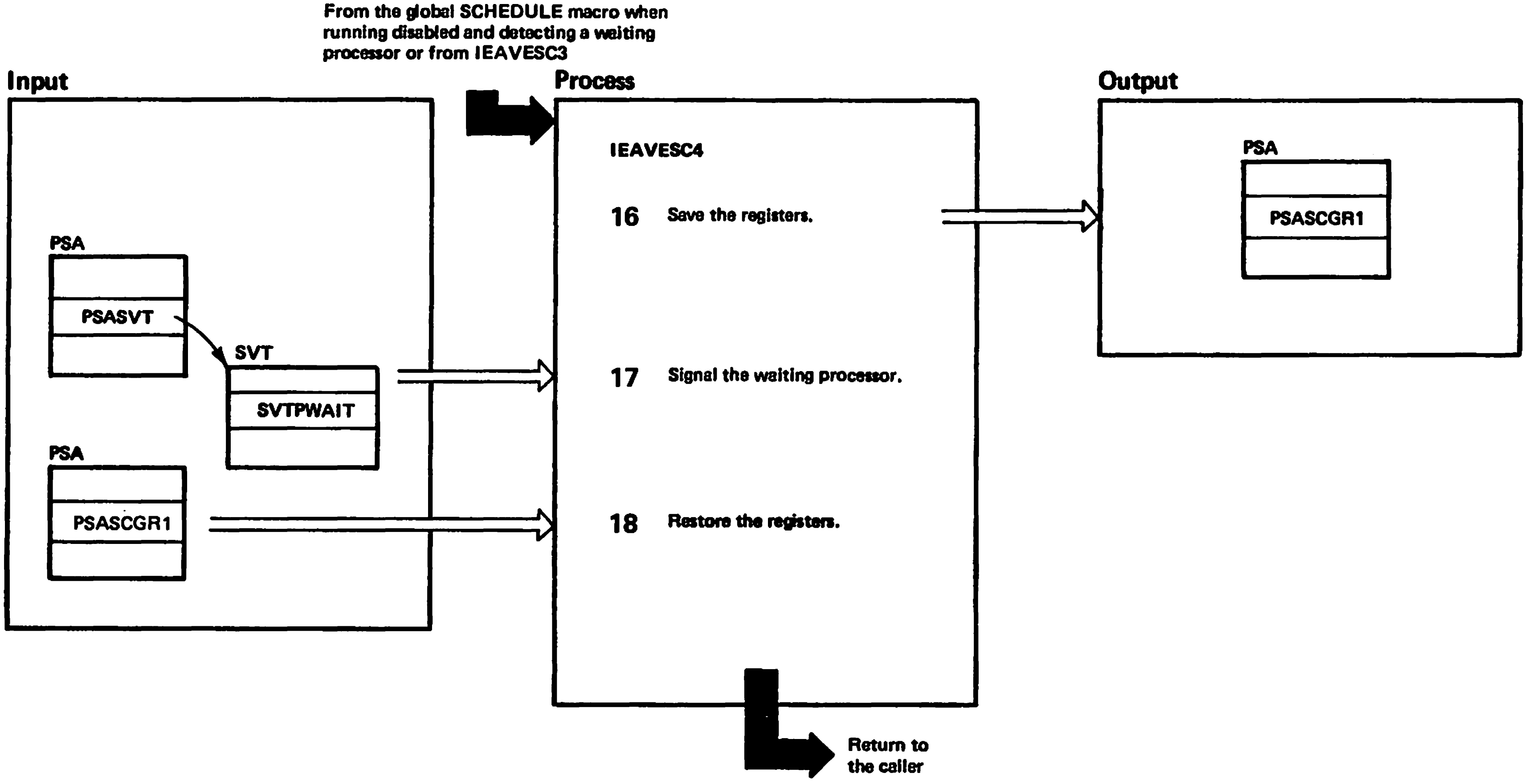
14 Branch to the disabled entry point.

15 Restore the original system mask and return to the caller.

Diagram SUP-31. SCHEDULE Processing (IEAVESC0) (Part 10 of 22)

Extended Description	Module	Label
This routine is entered (via a BALR) from the SCHEDULE macro when an enabled issuer has requested that the SRB be placed on a global SRB queue, and a waiting processor is detected. The SRB at this time has already been placed on the appropriate queue by the SCHEDULE macro.		
This routine disables for interrupts and passes control to the disabled global entry point (IEAVESC4) to signal any waiting processors.	IEAVESC0	IEAVESC3
13 This routine initially saves the caller's system mask in the first byte of PSASCPSW. This routine disables the PSW for I/O and external interrupts.		
14 This routine passes control to the SCHEDULE entry point, IEAVESC4.		
15 This routine restores the caller's registers and saved system mask and returns to the caller.		

Diagram SUP-31. SCHEDULE Processing (IEAVESC0) (Part 11 of 22)


From the global SCHEDULE macro when running disabled and detecting a waiting processor or from IEAVESC3
Input
PSA
PSASVT
SVT
SVTPWAIT
PSA
PSASCGR1
Process
IEAVESC4
Output
PSA
PSASCGR1
Return to the caller

16 Save the registers.

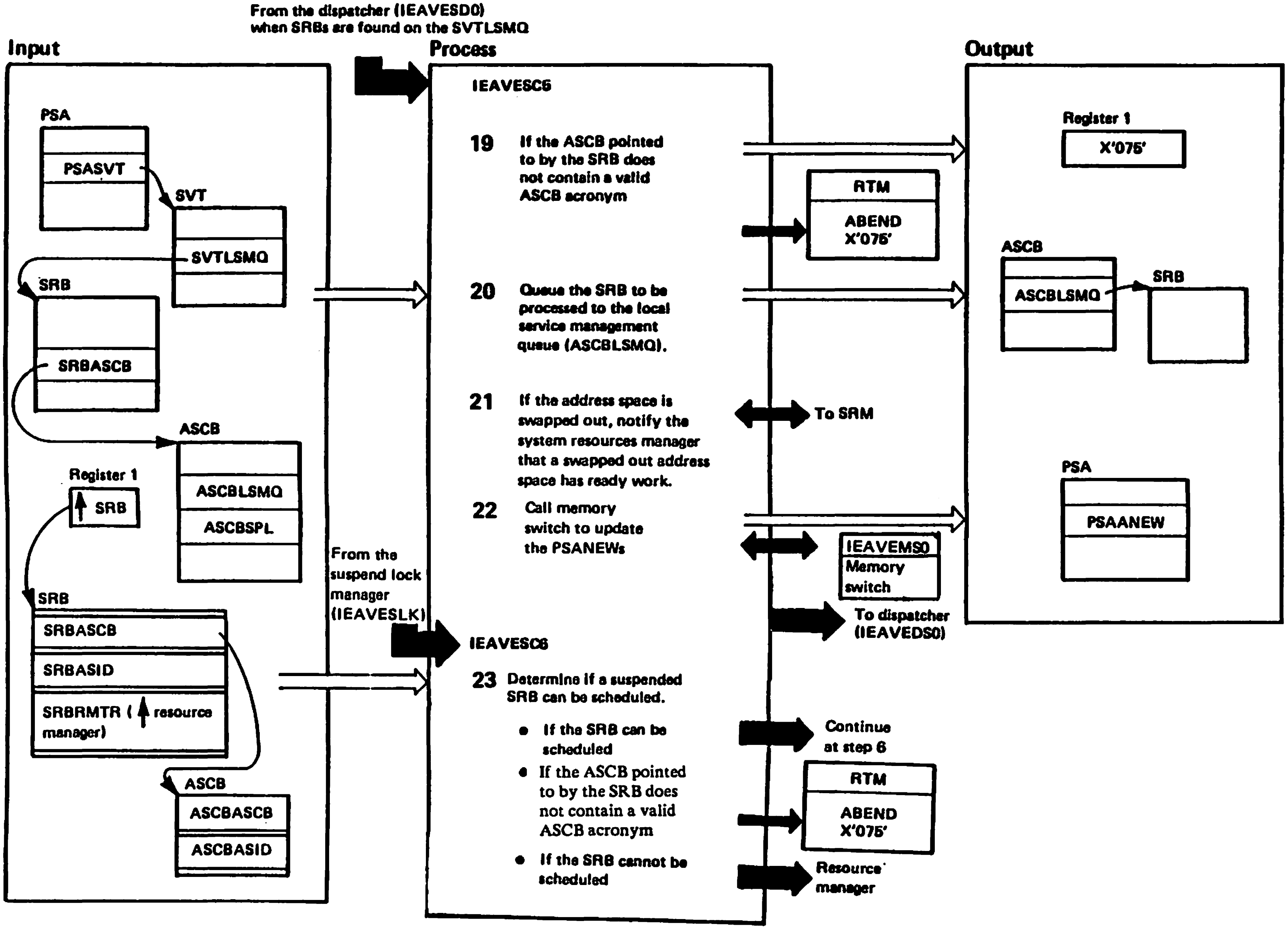
17 Signal the waiting processor.

18 Restore the registers.

Diagram SUP-31. SCHEDULE Processing (IEAVESC0) (Part 12 of 22)

Extended Description	Module	Label
This routine is entered (via a BALR) from the SCHEDULE macro when a disabled issuer has requested that the SRB be placed on a global SRB queue and a waiting processor is detected. It can also be entered from the enabled global entry point (IEAVESC3) via BASR. In this case the SRB has already been added to the appropriate queue by the SCHEDULE macro.		
16 IEAVESC0 saves registers 13 and 14 in PSASCGR1.	IEAVESC0	IEAVESC4
17 IEAVESC0 issues a SIGP instruction with an EXTERNAL CALL order code to signal the waiting processor.		
18 IEAVESC0 returns control to IEAVESC3 or the disabled caller.		

Diagram SUP-31. SCHEDULE Processing , (IEAVESC0) (Part 13 of 22)



IEAVESC5

19 If the ASCB pointed to by the SRB does not contain a valid ASCB acronym

20 Queue the SRB to be processed to the local service management queue (ASCBLSMQ).

21 If the address space is swapped out, notify the system resources manager that a swapped out address space has ready work.

22 Call memory switch to update the PSANEWs

IEAVESC6

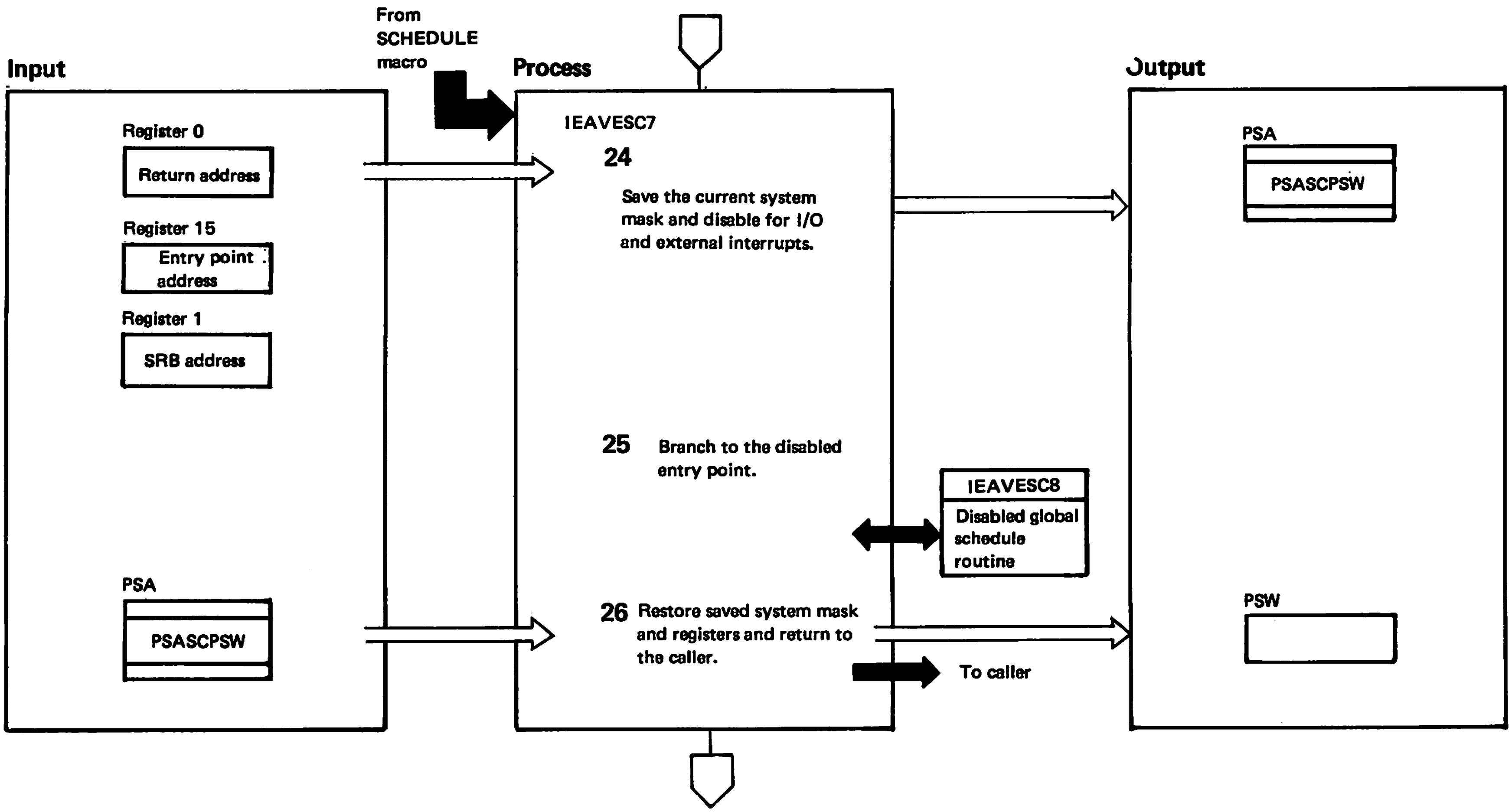
23 Determine if a suspended SRB can be scheduled.

- If the SRB can be scheduled
- If the ASCB pointed to by the SRB does not contain a valid ASCB acronym
- If the SRB cannot be scheduled

Diagram SUP-31. SCHEDULE Processing (IEAVESC0) (Part 14 of 22)

Extended Description	Module	Label
For each SRB on the queue the following processing is done:		
19 If the ASCB pointed to by the SRB does not contain a valid ASCB acronym, IEAVESC0 issues an X'076' ABEND.		IEAVESC5
20 IEAVESC0 queues the SRB to the ASCBLSMQ in the address space identified by SRBASCB.		
21 If the address space is swapped out, IEAVESC0 notifies SRM (system resources manager) of work ready to be dispatched to an address space already swapped-out. This causes an eventual swap-in of that address space.		
22 IEAVESC0 invokes the memory switch service (IEAVEMS0) with the ASCB address that has ready work. Memory switch looks for a processor eligible to run the ready work. If an eligible processor is found, PSAANEW for that processor is updated. (Memory Switch is described in the DISP section of the *System Logic Library.)*		
23 IEAVESC0 receives control at entry point IEAVESC6 when the suspend lock manager (IEAVESLK) finds a CML requestor on the suspend queue of a local lock it is releasing. IEAVESC0 ensures that the SRB is scheduled to the appropriate ASCB and that the target ASCB is valid. If the ASCB pointed to by the SRB contains a valid acronym, and its ASCBASID value matches the ASID saved at the time the SRB was suspended, IEAVESC0 continues processing at step 6. There it places the SRB on the appropriate local SRB queue. If the ASCB does not contain a valid acronym, IEAVESC0 issues an X'075' abend. Otherwise, it gives control to the resource manager whose address is in the SRBRMTR field. The resource manager frees the SRB storage and returns to IEAVESLK.		

Diagram SUP-31 SCHEDULE Processing (IEAVESC0) (Part 15 of 22)



24 Save the current system mask and disable for I/O and external interrupts.

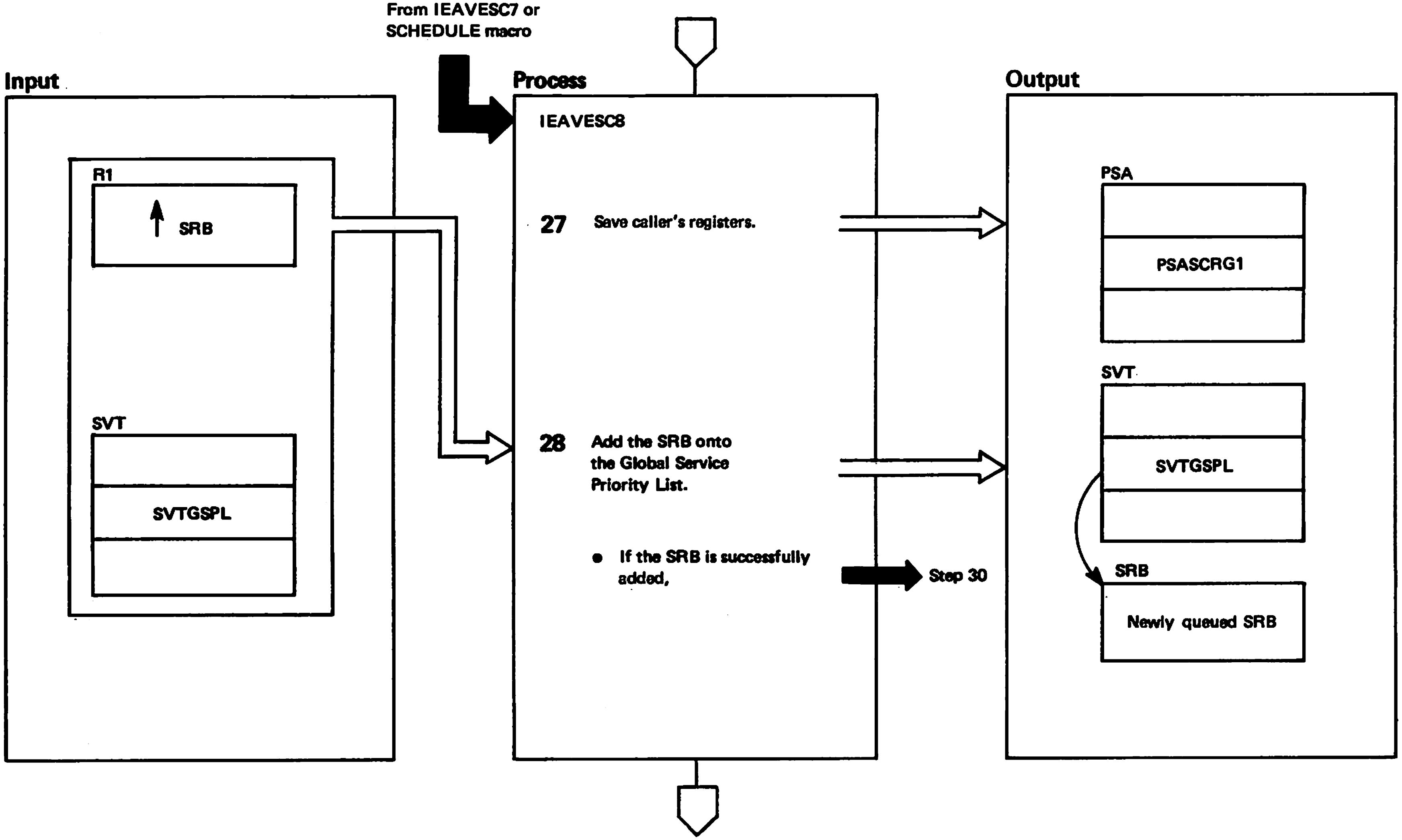
25 Branch to the disabled entry point.

26 Restore saved system mask and registers and return to the caller.

Diagram SUP-31. SCHEDULE Processing (IEAVESC0) (Part 16 of 22)

Extended Description	Module	Label
Entry points IEAVESC7 and IEAVESC8 are entered when a global schedule macro is invoked and SRB queueing is not done inline as part of the SCHEDULE macro expansion.		
IEAVESC7 is entered from the SCHEDULE macro via BALR when an enabled user has requested that an SRB be placed on a global SRB queue.		
IEAVESC8 is entered from the SCHEDULE macro when a disabled user has requested that an SRB be placed on a global SRB queue. IEAVESC8 can also be entered from the enabled global entry point IEAVESC7 via a BASR.		
24 IEAVESC0 initially saves the caller's system mask in the first byte of PSASCPSW and disables the PSW for I/O and external interrupts.		
25 IEAVESC7 passes control to the SCHEDULE entry point IEAVESC8.		
26 IEAVESC7 restores the caller's registers and saved system mask and returns to the caller.		

Diagram SUP-31. SCHEDULE Processing (IEAVESC0) (Part 17 of 22)



IEAVESC8

27 Save caller's registers.

28 Add the SRB onto the Global Service Priority List.

- If the SRB is successfully added,

Diagram SUP-31. SCHEDULE Processing (IEAVESCO) (Part 18 of 22)

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

IEAVESC8 is entered from the SCHEDULE macro via a BALR when the schedule macro with the disabled keyword was specified and inline SRB queuing is not done as part of the macro expansion. IEAVESC8 is also entered from SCHEDULE entry point IEAVESC7 via a BALR.

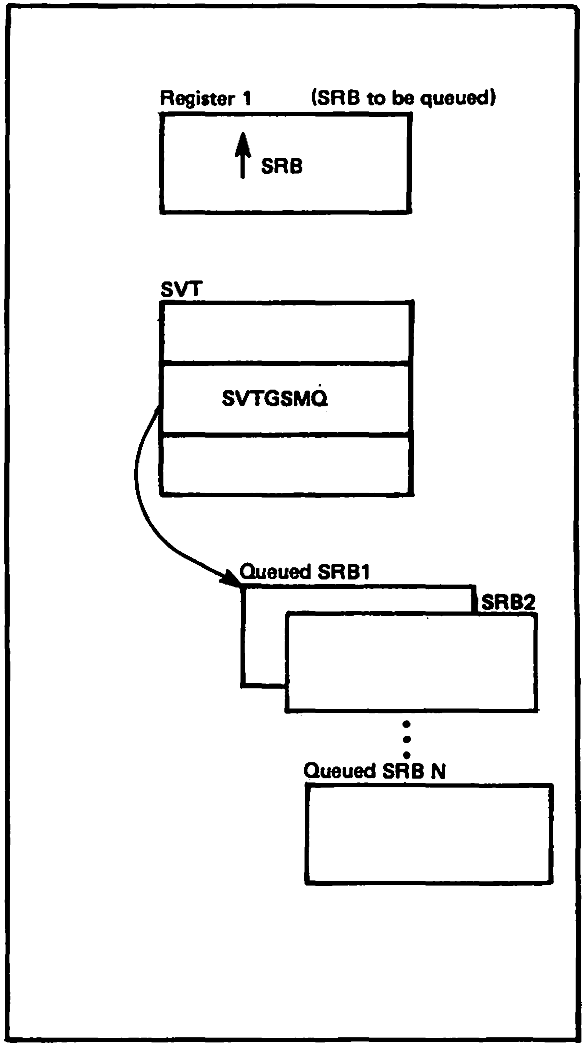
27 IEAVESC8 saves the caller's register 13 and 14.

28 IEAVESC8 adds the SRB to the global service priority list (GSPL). If there were no elements queued on the GSPL, the add will be successful. If the add is successful,

- IEAVESC8 transfers SRB queued on the global service management queue (GSMQ) if any, to the GSPL.
- IEAVESC8 continues at step 35.

Diagram SUP-31. SCHEDULE Processing (IEAVESC0) (Part 19 of 22)

Input



Process

29 Add the SRB onto the Global Service management queue.

Output

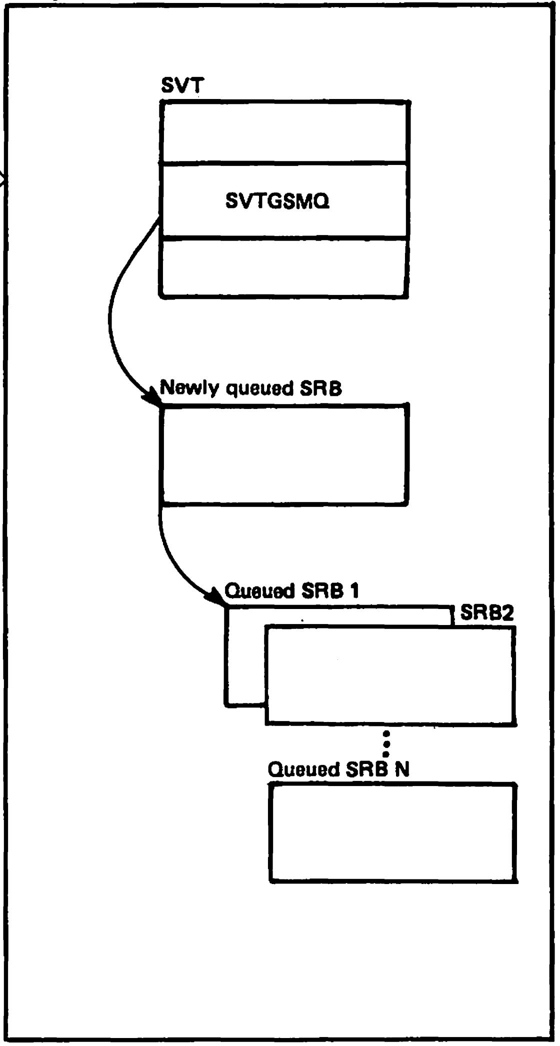
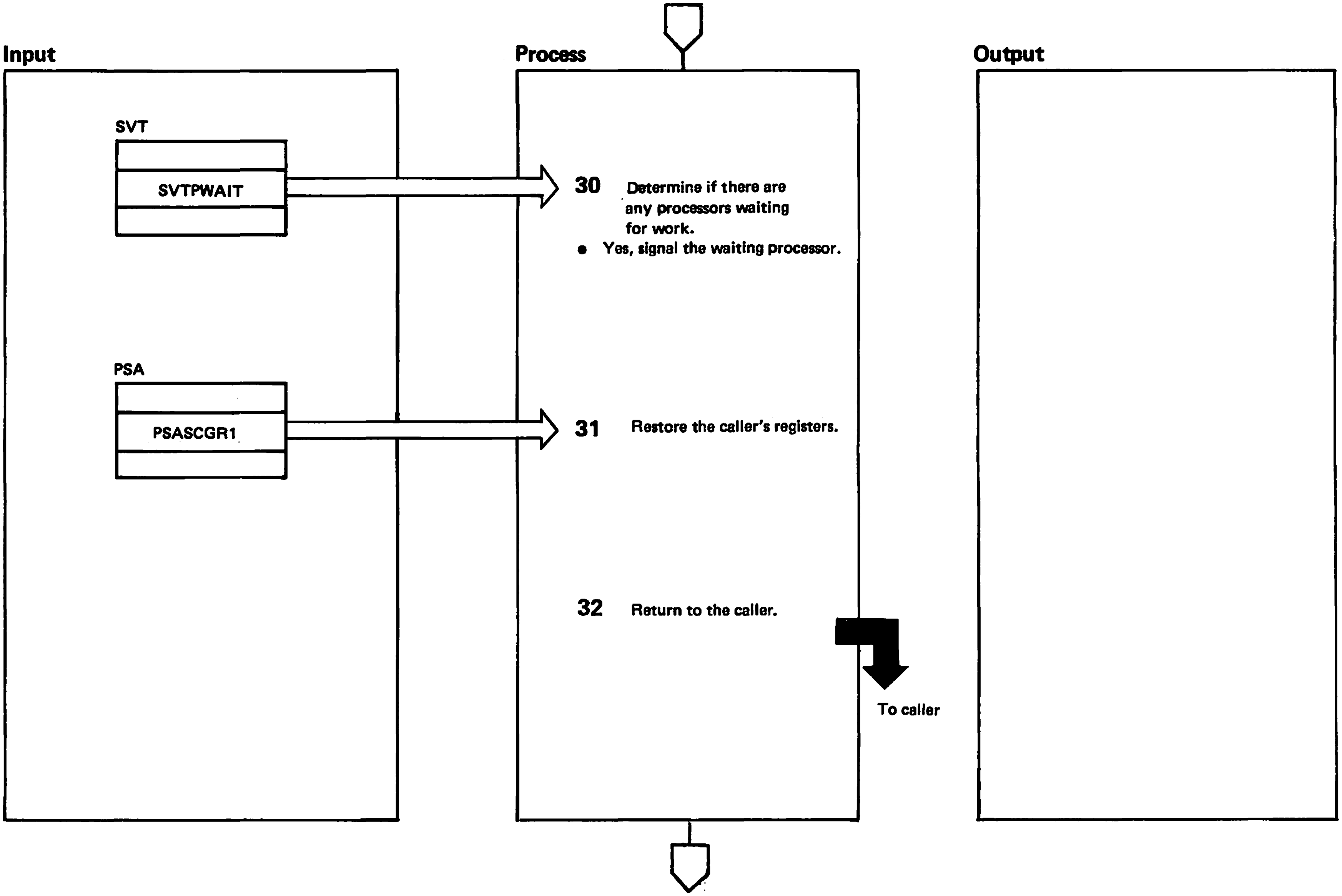


Diagram SUP-31. SCHEDULE Processing (IEAVESC0) (Part 20 of 22)

Extended Description	Module	Label
29 The SRB is added to the GSMQ. The add function is retried until it is successful.		

Diagram SUP-31. SCHEDULE Processing (IEAVESC0) (Part 21 of 22)



30 Determine if there are any processors waiting for work.

- Yes, signal the waiting processor.

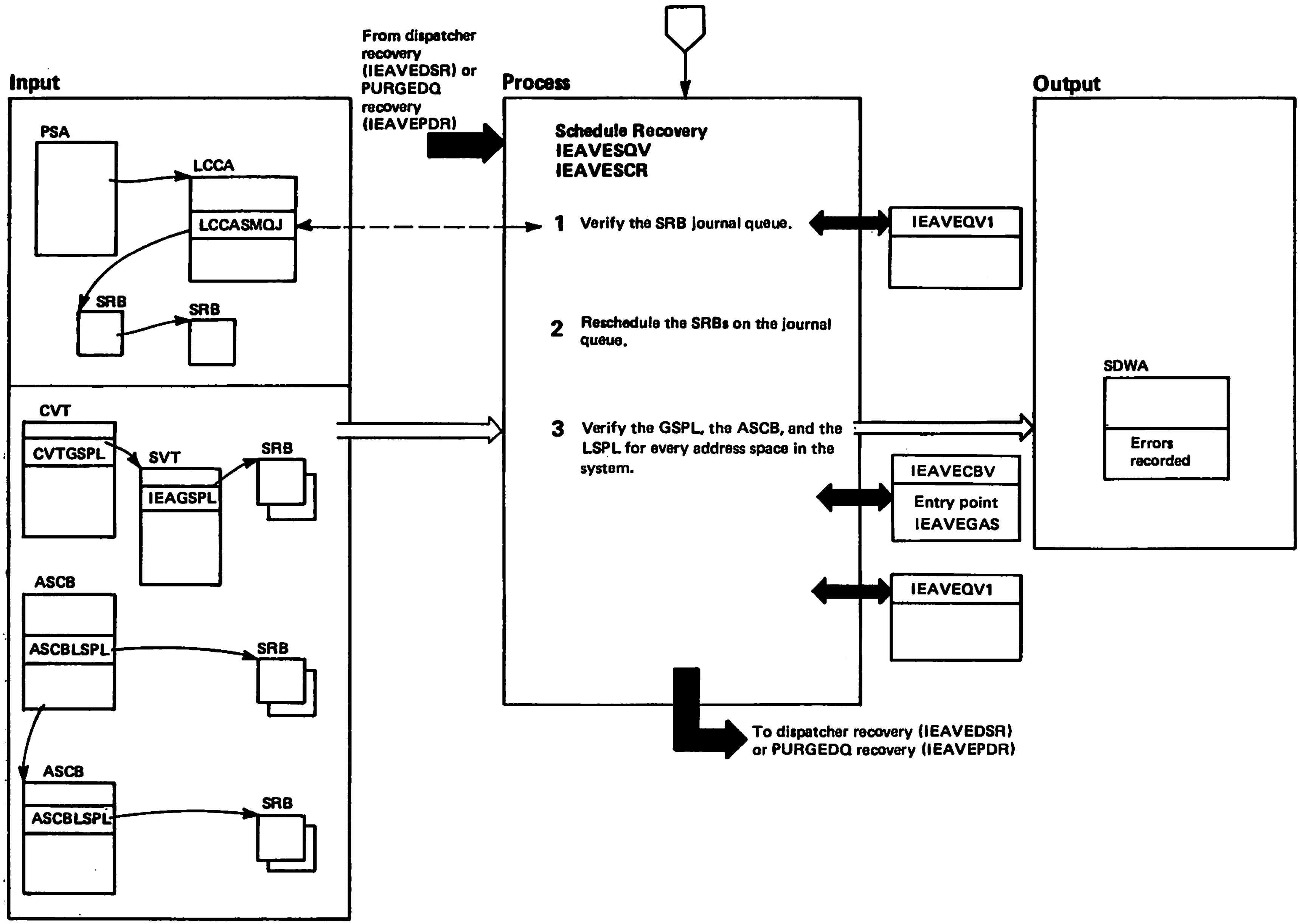
31 Restore the caller's registers.

32 Return to the caller.

Diagram SUP-31. SCHEDULE Processing (IEAVESC0) (Part 22 of 22)

Extended Description	Module	Label
30 Examine the processor waiting vector (SVTPWAIT) for waiting processors. If one is found, issue a SIGP external call to activate the processor.		
31 Restore the caller's registers.		
32 Return to the caller.		

Diagram SUP-32. SCHEDULE Recovery Processing (IEAVESCR) (Part 1 of 2)



1 Verify the SRB journal queue.

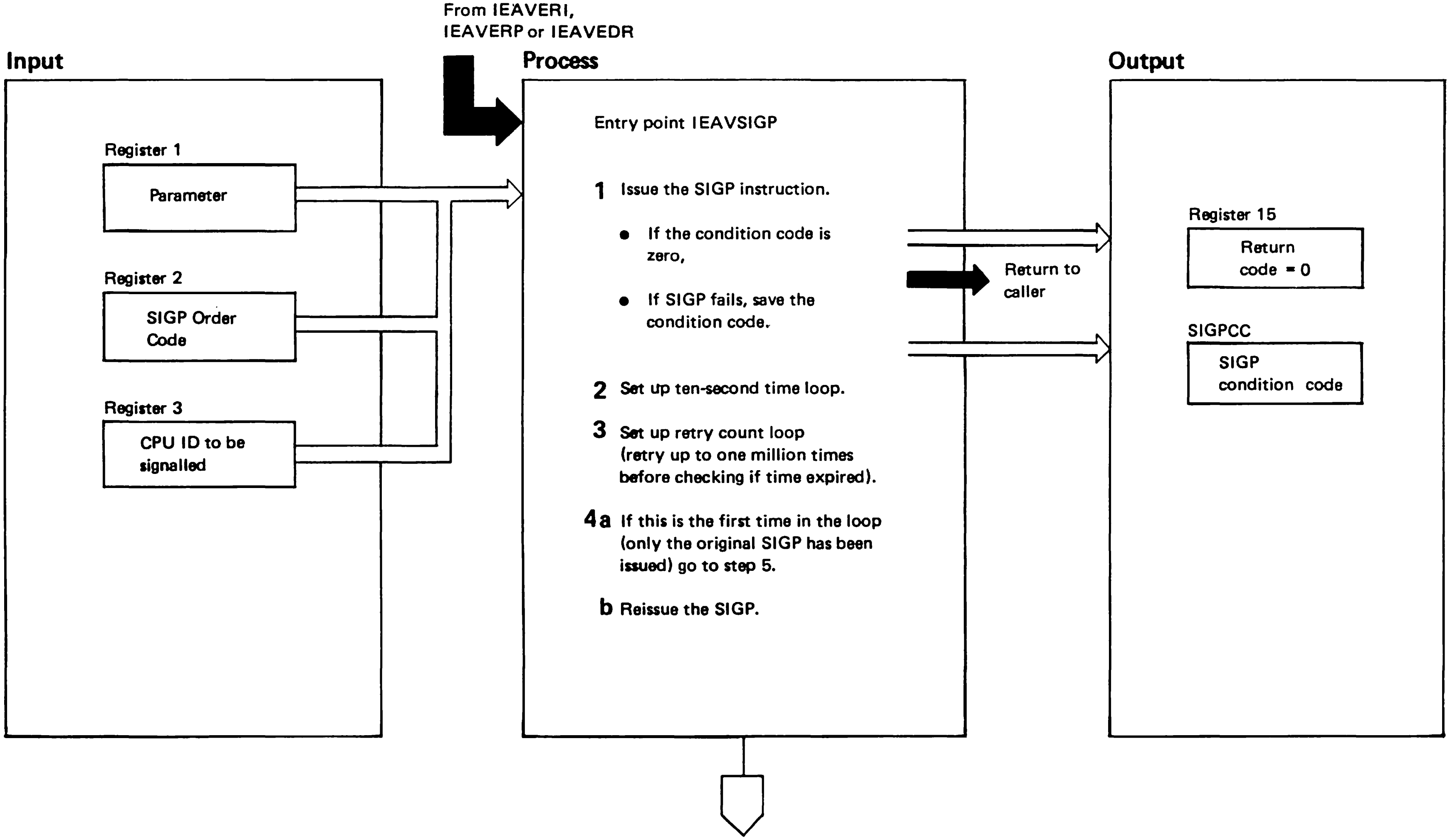
2 Reschedule the SRBs on the journal queue.

3 Verify the GSPL, the ASCB, and the LSPL for every address space in the system.

Diagram SUP-32. SCHEDULE Recovery Processing (IEAVESCR) (Part 2 of 2)

Extended Description	Module	Label
1 The schedule FRR verifies the SRB journal queue anchored out of the LCCASMQJ field and removes SRBs with bad information. The journal queue is used by SCHEDULE to prevent losing SRBs that are being processed.	IEAVESCR	SRBQV
2 The schedule FRR then reschedules any SRBs remaining on the journal queue.		
3 The schedule FRR uses the queue verifier to verify SRB queues — the GSPL, then the ASCB and the LSPL for every address space in the system. Errors detected are recorded in the SDWA; elements removed are also noted in the SDWA.		

Diagram SUP-33. Interprocessor Communication SIGP Routine (IEAVESGP) (Part 1 of 8)



Entry point IEAVSIGP

1 Issue the SIGP instruction.

- If the condition code is zero,
- If SIGP fails, save the condition code.

2 Set up ten-second time loop.

3 Set up retry count loop (retry up to one million times before checking if time expired).

4a If this is the first time in the loop (only the original SIGP has been issued) go to step 5.

b Reissue the SIGP.

Diagram SUP-33. Interprocessor Communication SIGP Routine (IEAVESGP) (Part 2 of 8)

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

The interprocessor communication SIGP routine provides the necessary environment and interfaces to issue a SIGP request. It transmits an eight-bit order code to a specified processor and handles all error conditions resulting from the execution of the SIGP instruction.

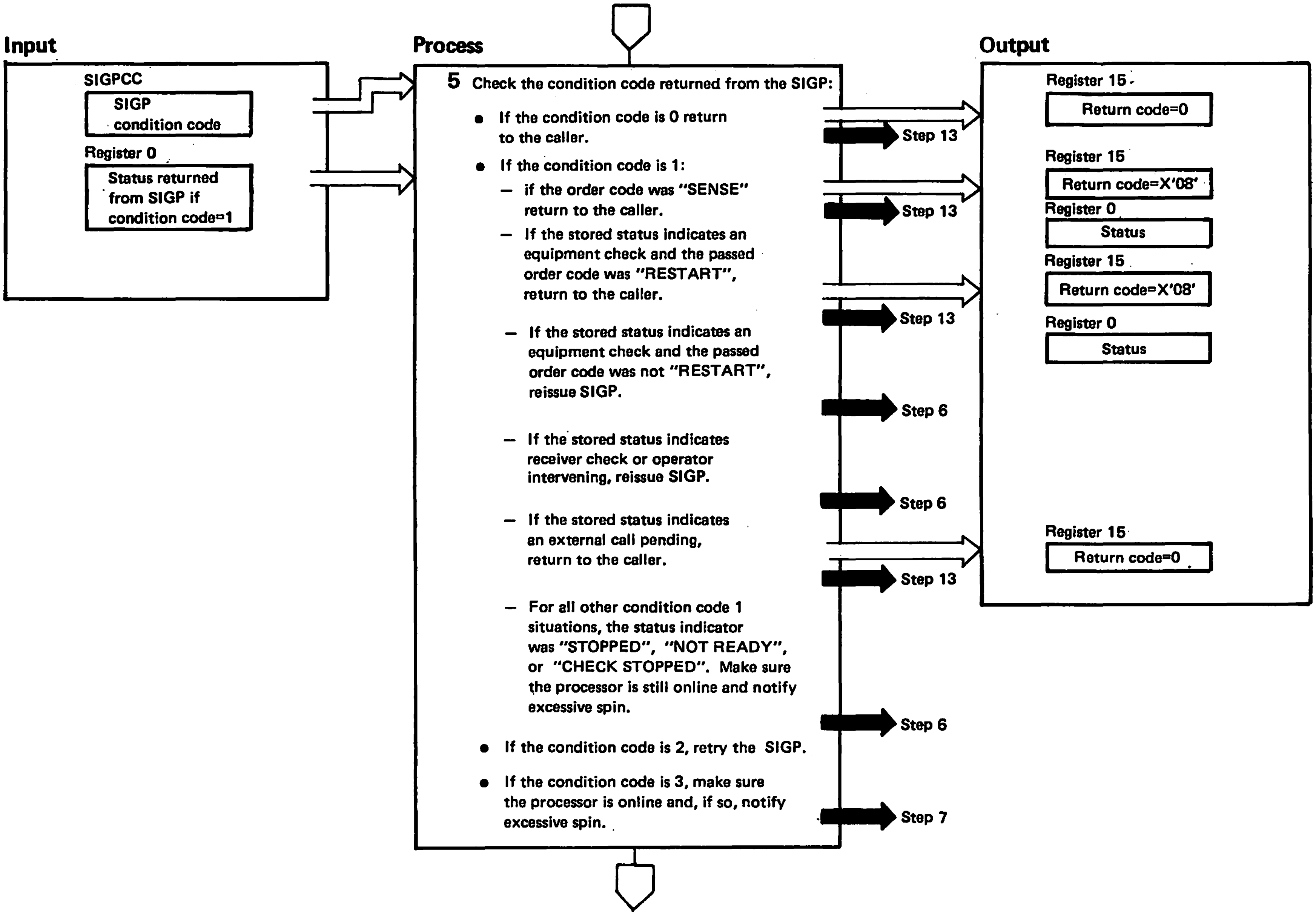
1 Issues the SIGP instruction.

- If a zero condition code results, returns to the caller.
- If the SIGP fails, saves the condition code.

2-3 Sets up the loop controls. If necessary, reissues the SIGP up to one million times. If after one million retries the SIGP is still unsuccessful, IEAVESGP assumes a problem exists and determines if the processor to be signalled is still online. If not, it exits to its caller with a return code of X'0C' if the caller was the direct signal service routine; else it exits with a return code of X'04'. If the processor still appears to be online, a window for MFA interrupts is opened. If after the window has been opened the processor appears offline, IEAVESGP returns to its caller with a return code of X'0C' if the caller was the direct signal service routine; else it exits with a return code of X'04'. If not, and if the ten-second time loop has not expired, the SIGP is retried up to one million more times. If the ten-second loop has expired, IEAVESGP calls excessive spin notification to inform the operator of the problem if the caller was not the direct signal service routine.

4 If this is not the first time the SIGP instriction is issued, reissues the SIGP. If this is the first time, checks condition codes in the next step.

Diagram SUP-33. Interprocessor Communication SIGP Routine (IEAVESGP) (Part 3 of 8)



Input

SIGPCC

SIGP condition code

Register 0

Status returned from SIGP if condition code=1

Process

5 Check the condition code returned from the SIGP:

- If the condition code is 0 return to the caller. → Step 13
- If the condition code is 1:
  - – if the order code was "SENSE" return to the caller. → Step 13
  - – If the stored status indicates an equipment check and the passed order code was "RESTART", return to the caller. → Step 13
  - – If the stored status indicates an equipment check and the passed order code was not "RESTART", reissue SIGP. → Step 6
  - – If the stored status indicates receiver check or operator intervening, reissue SIGP. → Step 6
  - – If the stored status indicates an external call pending, return to the caller. → Step 13
  - – For all other condition code 1 situations, the status indicator was "STOPPED", "NOT READY", or "CHECK STOPPED". Make sure the processor is still online and notify excessive spin. → Step 6
- If the condition code is 2, retry the SIGP.
- If the condition code is 3, make sure the processor is online and, if so, notify excessive spin. → Step 7

Output

Register 15

Return code=0

Register 15

Return code=X'08'

Register 0

Status

Register 15

Return code=X'08'

Register 0

Status

Register 15

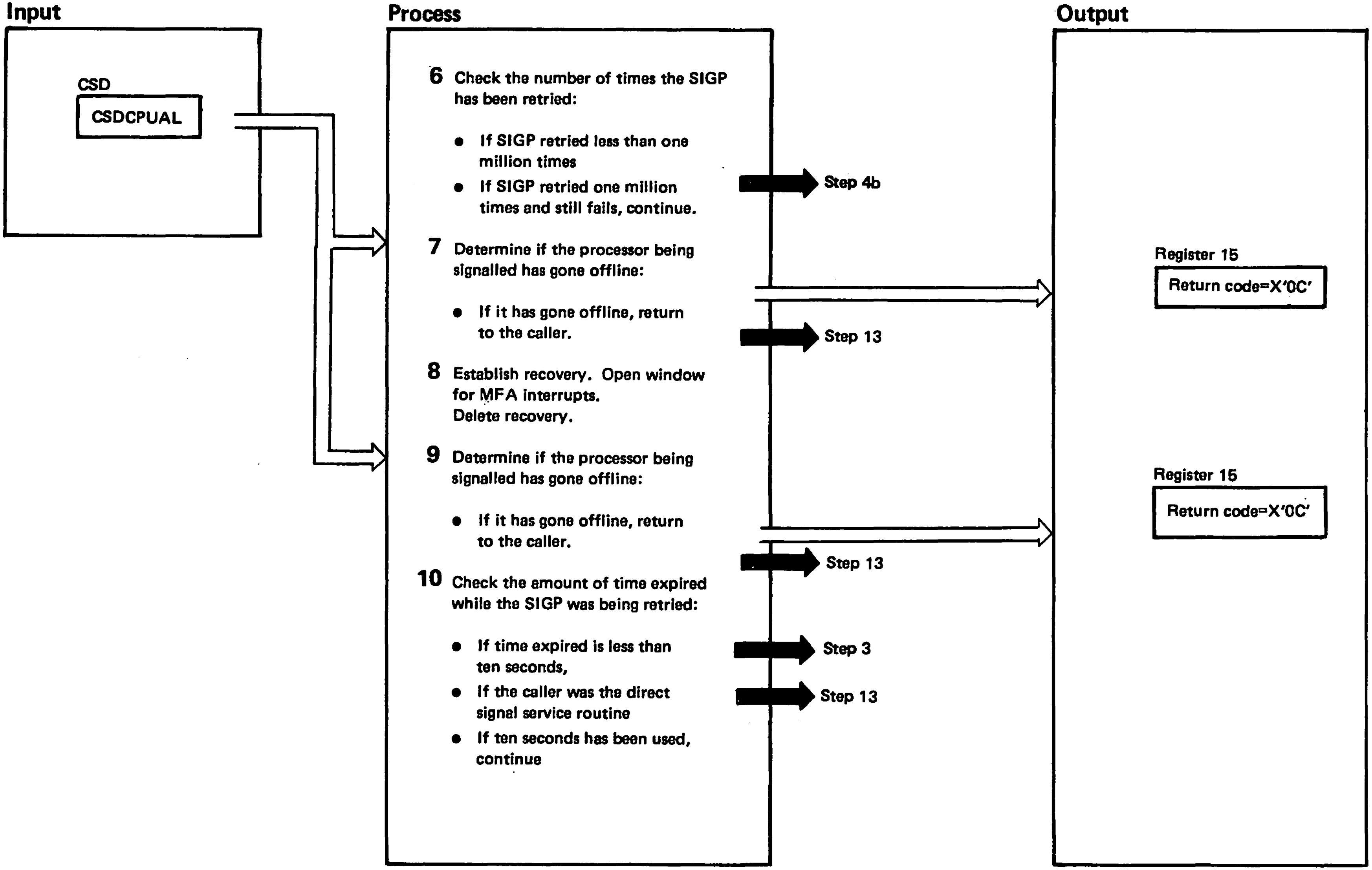
Return code=0

Extended Description	Module	Label

5 Analyzes the condition code from the first SIGP or any reissued SIGP to determine the appropriate action. The following actions can be taken:

- Return to the caller with a return code of:
  - 0 if the condition code=0.
  - 8 and status in register 0 if the condition code=1 and the order code was "SENSE".
  - 8 and status in register 0 if the condition code=1, stored status indicates an equipment check and the order code was "RESTART".
  - 0 if the condition code=1 and stored status indicates that an external call is pending.
- Reissue the SIGP if the condition code is:
  - 1, stored status indicates an equipment check and the order code was not "RESTART".
  - 1 and stored status indicates a "receiver check".
  - 1 and stored status indicates "operator intervening".
  - 2 (addressed processor is busy).
- Determine if processor is still online and if the caller was IEAVERI or IEAVERP. If so, call excessive spin if the condition code is:
  - 1 and the status indicator was "STOPPED", "NOT READY", or "CHECK STOP".
  - 3 (addressed processor is not operational).

Diagram SUP-33. Interprocessor Communication SIGP Routine (IEAVESGP) (Part 5 of 8)



6 Check the number of times the SIGP has been retried:

- If SIGP retried less than one million times
- If SIGP retried one million times and still fails, continue.

7 Determine if the processor being signalled has gone offline:

- If it has gone offline, return to the caller.

8 Establish recovery. Open window for MFA interrupts. Delete recovery.

9 Determine if the processor being signalled has gone offline:

- If it has gone offline, return to the caller.

10 Check the amount of time expired while the SIGP was being retried:

- If time expired is less than ten seconds,
- If the caller was the direct signal service routine
- If ten seconds has been used, continue

Diagram SUP-33. Interprocessor Communication SIGP Routine (IEAVESGP) (Part 6 of 8)

Extended Description	Module	Label

6 Checks the number of times SIGP has been issued. If the number of retries is less than one million, reissues the SIGP. If the number of retries is one million, continues processing.

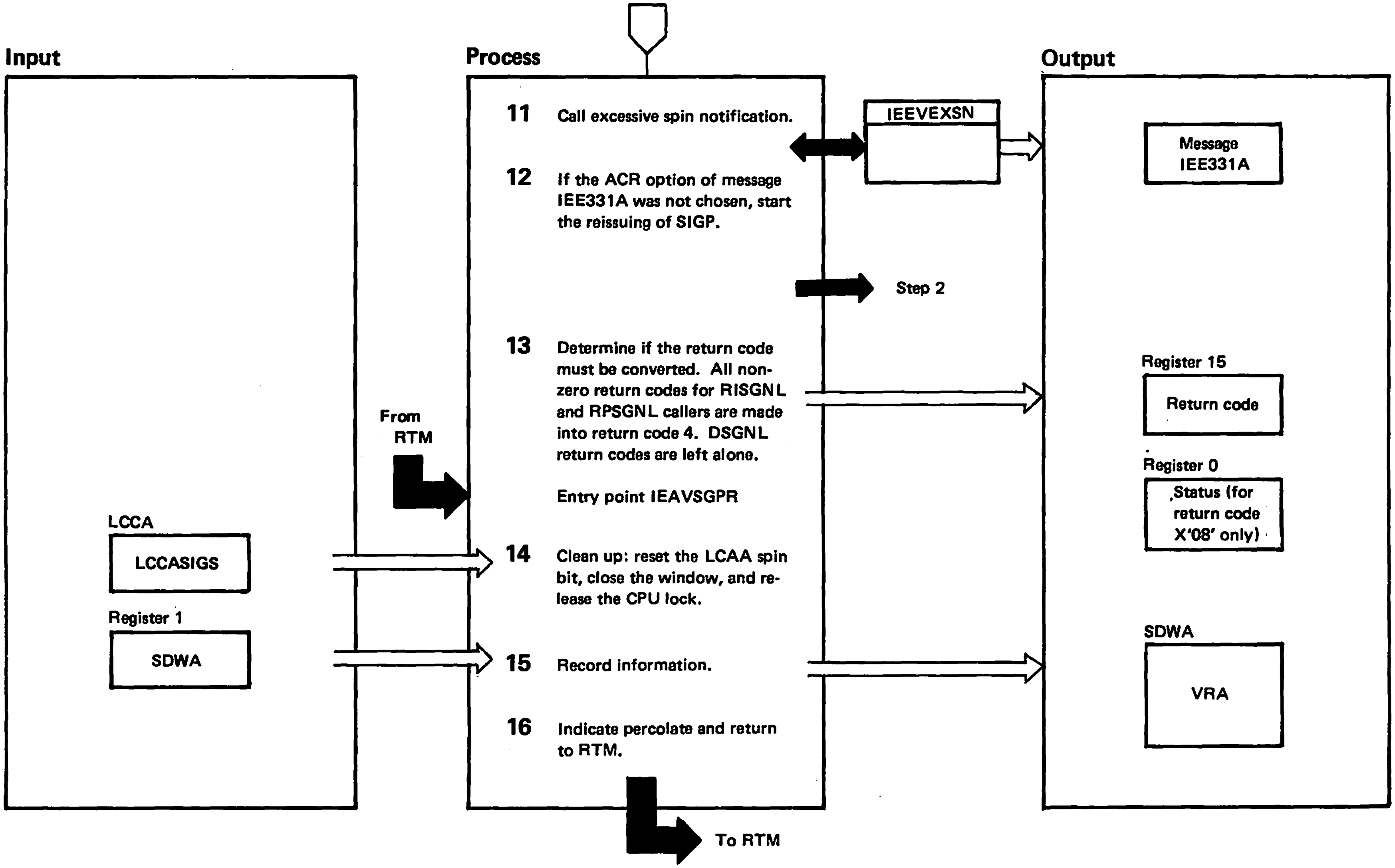
7 Determines if the processor being signalled is still online. If it is not, sets a return code of X'0C' and returns to the caller (see step 13).

8 If the processor is still online, opens a window for MFA interrupts.

9 When the window is closed, checks again to determine if the processor is still online. If it is not, sets a return code of X'0C' and returns to the caller (see step 13).

10 If ten seconds has not expired after the first one million retries, starts with another million retries of the SIGP. If ten seconds has expired and the SIGP is still not successful, a problem exists with the processor to be signalled and excessive spin notification is called. (Note: Excessive spin notification is not called if the caller of IEAVESGP was the direct signal routine (IEAVEDR). In that case, the last condition code returned from the SIGP instruction is converted to a return code and control returns to the caller.

Diagram SUP-33. Interprocessor Communication SIGP Routine (IEAVESGP) (Part 7 of 8)



11 Call excessive spin notification.

12 If the ACR option of message IEE331A was not chosen, start the reissuing of SIGP.

13 Determine if the return code must be converted. All nonzero return codes for RISGNL and RPSGNL callers are made into return code 4. DSGNL return codes are left alone.

Entry point IEAVSGPR

14 Clean up: reset the LCAA spin bit, close the window, and release the CPU lock.

15 Record information.

16 Indicate percolate and return to RTM.

Diagram SUP-33. Interprocessor Communication SIGP Routine (IEAVESGP) (Part 8 of 8)

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

11-12 If the processor is still online, sends the appropriate message code to the excessive spin notification routine (IEEVEXSN) to issue message IEE331A. If the ACR option of the message is chosen, the process is over. If the option to continue the spin is chosen, resets the loop counters and begins reissuing the SIGP again (step 2).

13 Determines if the return code must be converted. Converts all non-zero return codes for RISGNL and RPSGNL callers to return code 4. DSGNL return codes are not converted. The following are the return codes:

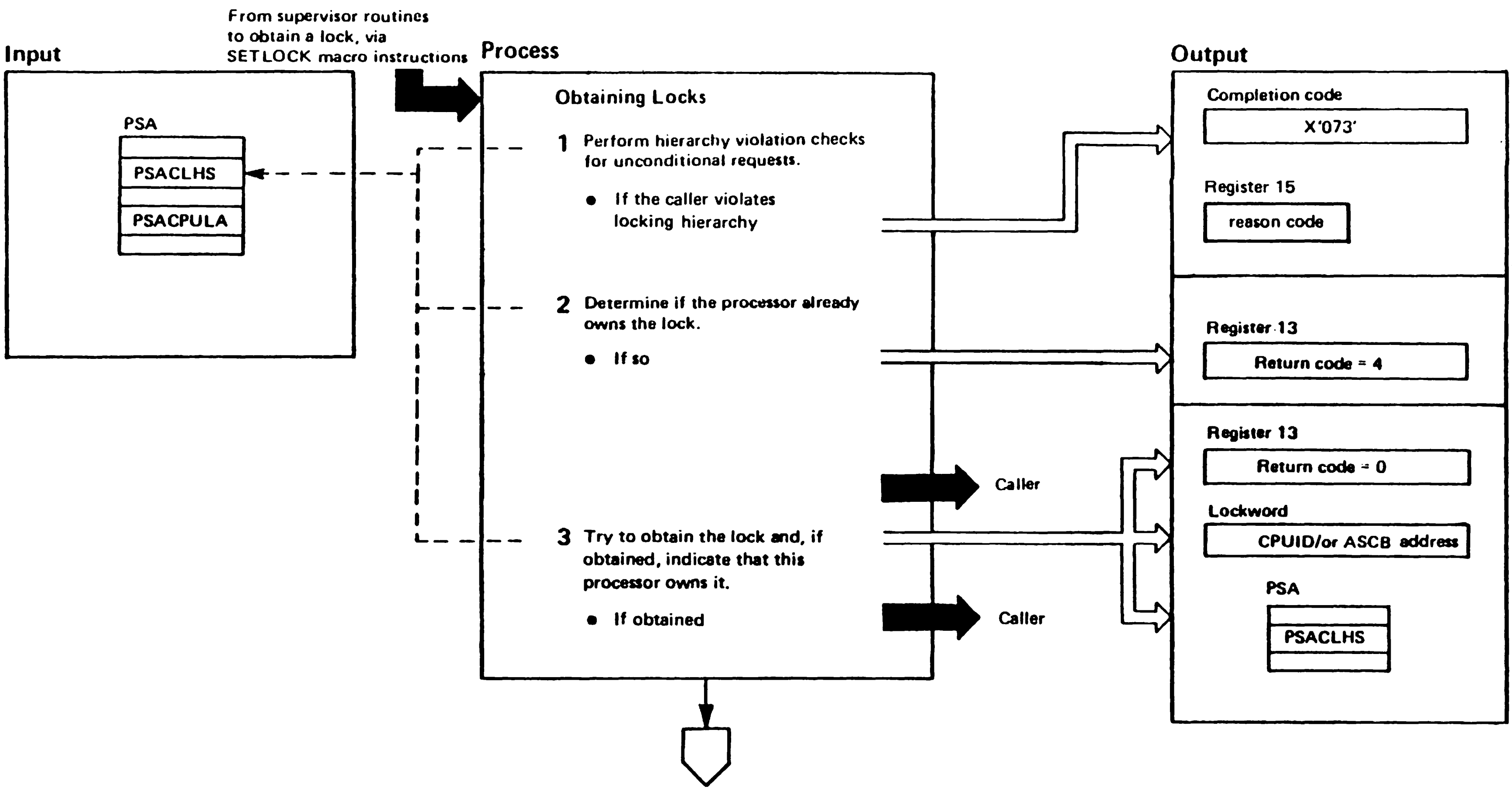
X '00' = successful
X '04' = RI/RP SGNL function not initiated
X '08' = DSGNL function unsuccessful or "SENSE" request is made.
X '0C' = DSGNL processor not operational.
X '14' = MSSF currently inoperative

14 Clean up: resets the LCCA spin bit, closes the window, and releases the CPU lock.

15 Records information in the SDWA.

16 Indicates percolate and returns to RTM.

Diagram SUP-34. Suspend Lock Manager Processing (IEAVESLK) (Part 1 of 10)



Input

PSA

PSACLHS

PSACPULA

From supervisor routines to obtain a lock, via SETLOCK macro instructions

Process

Obtaining Locks

1 Perform hierarchy violation checks for unconditional requests.

- If the caller violates locking hierarchy

2 Determine if the processor already owns the lock.

- If so

Caller

3 Try to obtain the lock and, if obtained, indicate that this processor owns it.

- If obtained

Caller

Output

Completion code

X'073'

Register 15

reason code

Register 13

Return code = 4

Register 13

Return code = 0

Lockword

CPUID/or ASCB address

PSA

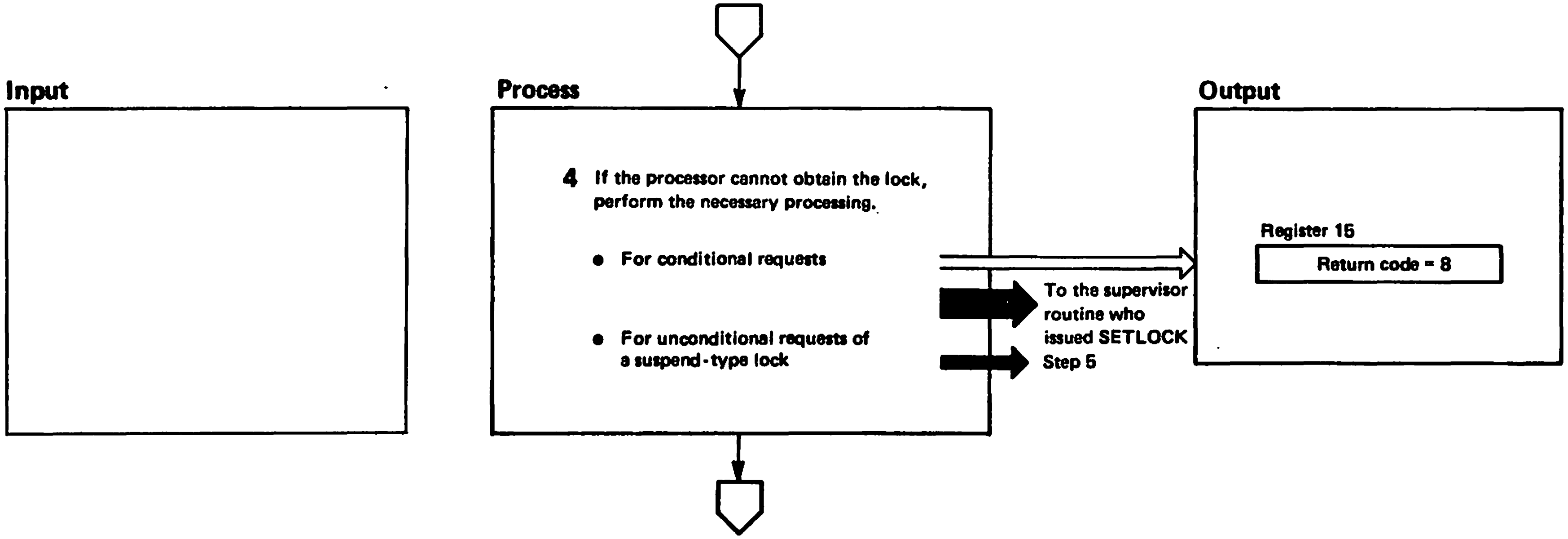
PSACLHS

Diagram SUP-34. Suspend Lock Manager Processing (IEAVESLK) (Part 2 of 10)

Extended Description	Module	Label
The lock manager provides the means for a user to obtain locks that serialize the use of a resource. The suspend lock manager provides the following locks: ● Cross memory services CMS (the general cross memory services lock) CMSEQDQ (the ENQ/DEQ cross memory services lock) CMSSMF (the SMF cross memory services lock) ● LOCAL (local address space lock) ● CML (cross memory local address space lock) The lock manager both obtains and releases locks. There are two distinct methods of obtaining locks; conditionally and unconditionally. If the lock cannot be obtained for a conditional request, the lock manager immediately returns control to the caller with the appropriate return code in register 13. If an unconditional request for a suspend type lock cannot be satisified, the lock manager places the requestor on a lock manager suspend queue until the lock becomes available.	IEAVESLK	

Extended Description	Module	Label
1 The lock manager determines whether the caller has violated the locking hierarchy by: ● Unconditionally requesting a lock lower in the hierarchy than a lock it already holds. ● Requesting a cross memory services lock while not holding the local lock. ● Requesting a suspend lock while disabled. The lock manager abnormally terminates callers who violate the hierarchy with a X'073' completion code and a reason code in register 15.	IEAVESLK	
2 The lock manager determines whether this processor already owns the requested lock. If this processor owns it, the lock manager puts a code of 4 in register 13, and returns control to the caller. Otherwise, processing continues.		
3 The lock manager tries to obtain the lock. If the lock is available (the lockword contains 0), the lock manager indicates ownership by: ● Placing into the lockword the logical processor ID or, for a cross memory services lock, the locally locked ASCB address ● Setting the appropriate bit in the processor-locks-held string (PSACLHS) ● For the CML lock, placing into the PSALOCAL field a pointer to the ASCB of the address space whose CML lock is held The lock manager then returns to the caller with a zero return code. If the lock is not available, processing continues at step 4.		

Diagram SUP-34. Suspend Lock Manager Processing (IEAVESLK) (Part 3 of 10)



Input

Process

Output

4 If the processor cannot obtain the lock, perform the necessary processing.

- For conditional requests

- For unconditional requests of a suspend-type lock

To the supervisor routine who issued SETLOCK

Step 5

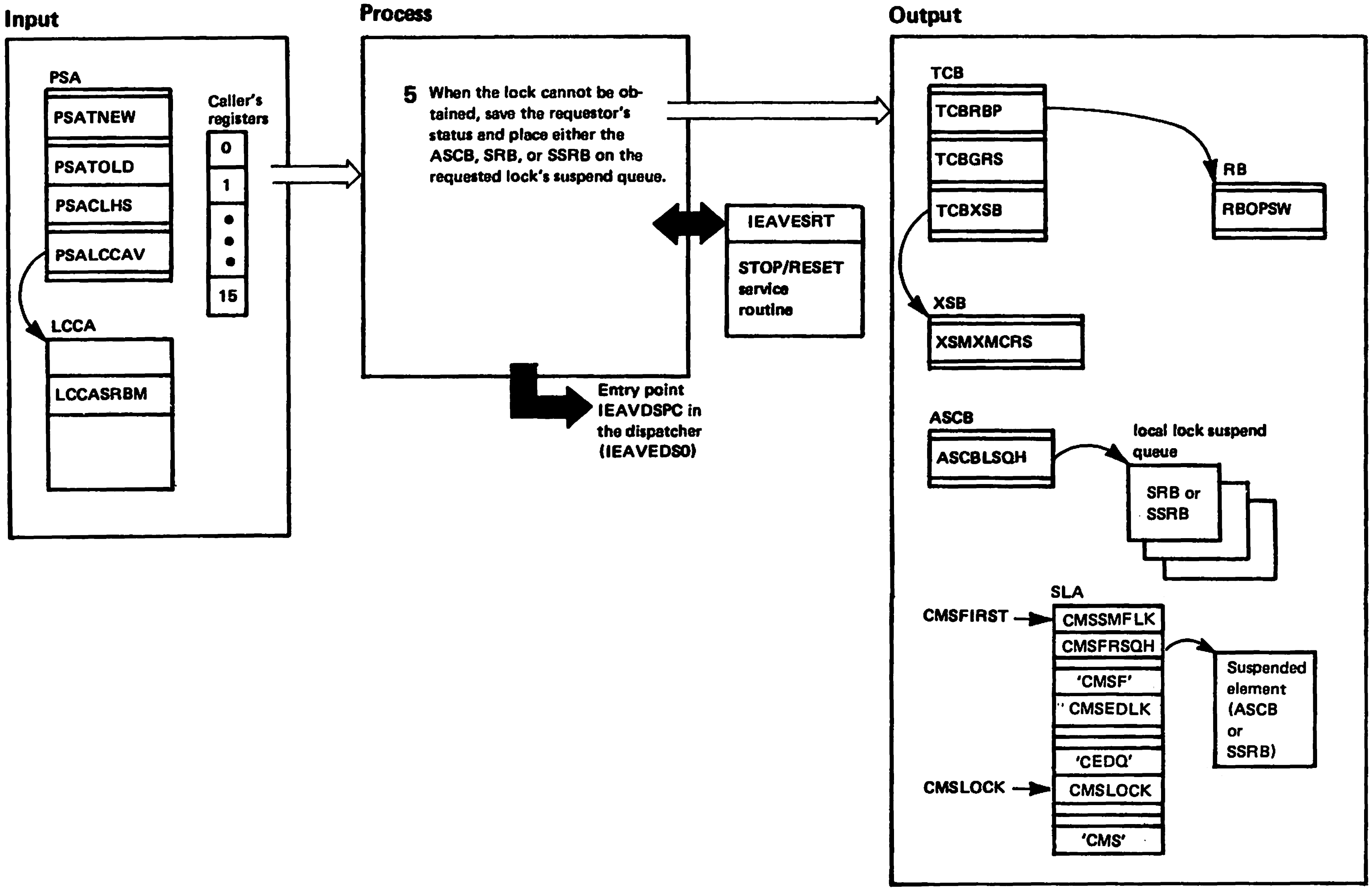
Register 15

Return code = 8

Diagram SUP-34. Suspend Lock Manager Processing (IEAVESLK) (Part 4 of 10)

Extended Description	Module	Label
4		
• For conditional requests, IEAVESLK sets a return code of 8 and returns control to the caller.		
• For unconditional requests for suspend locks (LOCAL, CML, or cross memory services), processing continues at step 5.		

Diagram SUP-34. Suspend Lock Manager Processing (IEAVESLK) (Part 5 of 10)



5 When the lock cannot be obtained, save the requestor's status and place either the ASCB, SRB, or SSRB on the requested lock's suspend queue.

Diagram SUP-34. Suspend Lock Manager Processing (IEAVESLK) (Part 6 of 10)

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

5 The lock manager suspends the requesting task or SRB.

- IEAVESLK issues the SETFRR macro to establish IEAVLKRR as the functional recovery routine (FRR); and turns on the lock manager super bit.
- IEAVESLK saves the caller's status depending on the caller's mode (task or SRB), and which lock the caller requested (LOCAL, CML, or cross-memory services).

Lock requested \ Caller's mode	Task mode	SRB mode
LOCAL	• Saves the caller's registers and cross memory status in the TCB and XSB. • Stores into the RB a PSW that causes the requestor to reenter IEAVESLK when redispatched. There IEAVESLK tries again to obtain the local lock.	Calls the STOP/RESET service routine (IEAVESRT) at entry point IEAVSUSF to save the requestor's status. IEAVESRT obtains storage for an SSRB and XSB and saves the requestor's status in them. The saved SRB causes reentry to IEAVESLK where IEAVESLK tries to obtain the LOCAL lock. It returns the address of the SSRB to IEAVESLK.
CML	• Obtains an SRB to represent the task's request on the lock suspend queue. • Initializes the SRB to enter the lock manager subroutine IEAVRTCB when dispatched. IEAVRTCB issues a RESUME macro to resume the suspended task, which tries again to obtain the CML lock.	Same as above

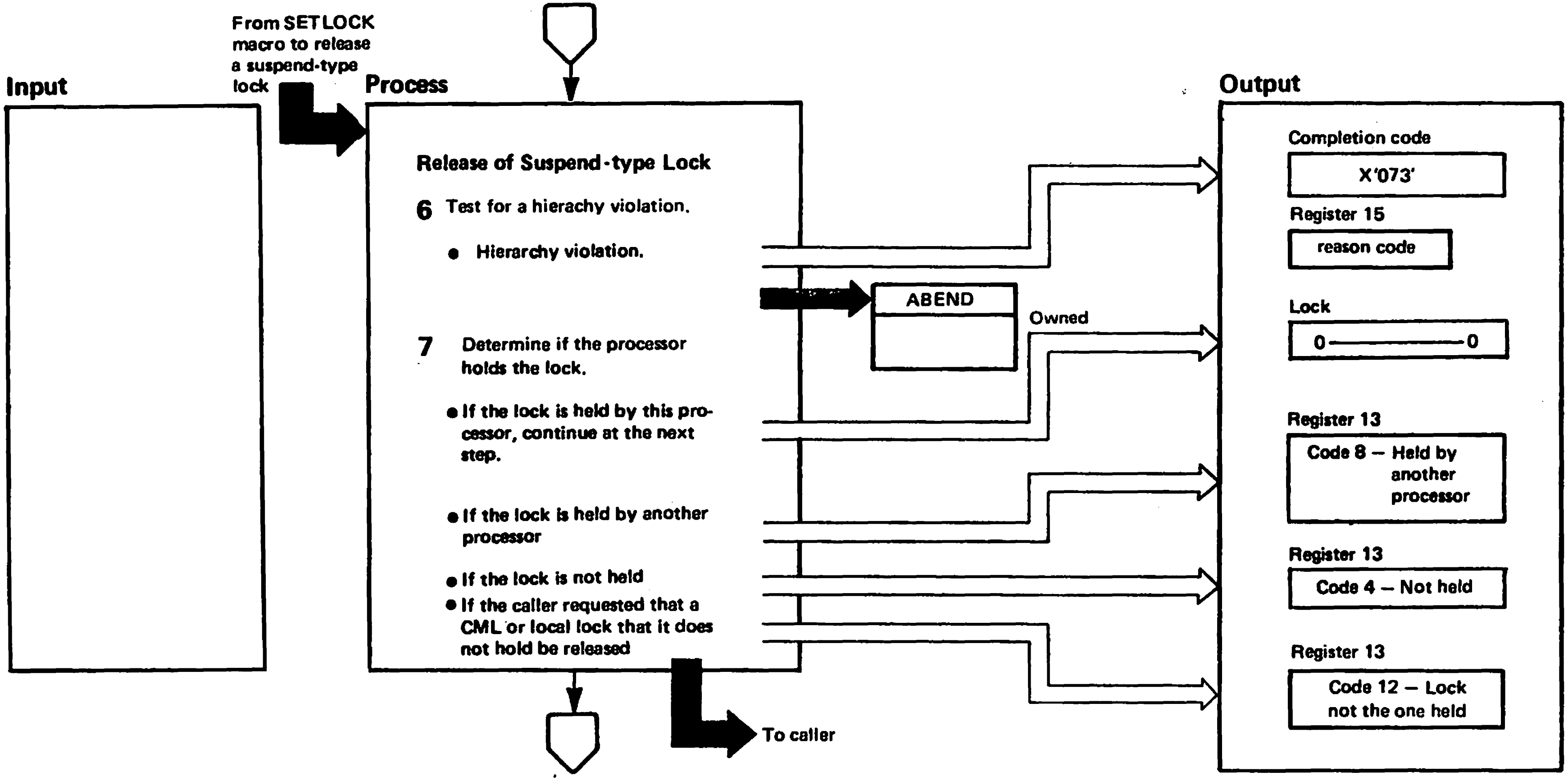
Extended Description **Module** **Label**

5 (continued)

Lock requested \ Caller's mode	Task mode	SRB mode
Cross-memory-services	Calls the STOP/RESET service routine (IEAVESRT) at entry point IEAVSUSF to save the requestor's status. IEAVESRT saves the task's status in the IHSA, TCB, and XSB.	Same as above

Unless IEAVESLK is saving status for a task that requested the LOCAL lock, it suspends the lock requestor on the appropriate lock's suspend queue. To do so, IEAVESLK places either the locally locked ASCB address (for a task requesting a cross-memory-services lock), the SRB (for a task requesting the CML lock), or the SSRB (for an SRB) on the suspend queue. IEAVESLK then gives control to the dispatcher at entry point IEAVDSPC.

Diagram SUP-34. Suspend Lock Manager Processing (IEAVESLK) (Part 7 of 10)



Input

From SETLOCK macro to release a suspend-type lock

Process

Release of Suspend-type Lock

6 Test for a hierachy violation.

- Hierarchy violation.

7 Determine if the processor holds the lock.

- If the lock is held by this processor, continue at the next step.
- If the lock is held by another processor
- If the lock is not held
- If the caller requested that a CML or local lock that it does not hold be released

ABEND

Owned

To caller

Output

Completion code

X'073'

Register 15

reason code

Lock

0————0

Register 13

Code 8 – Held by another processor

Register 13

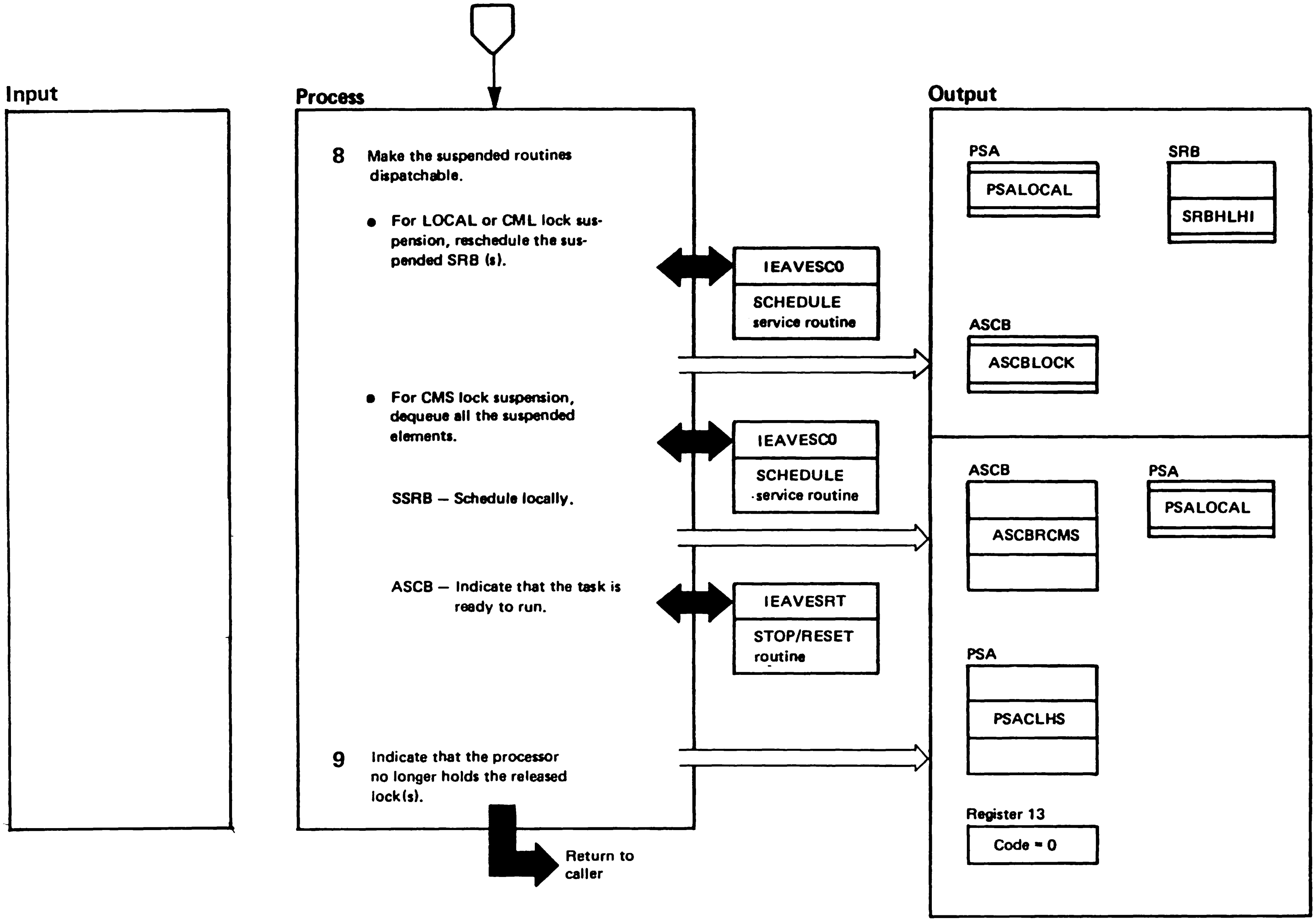
Code 4 – Not held

Register 13

Code 12 – Lock not the one held

Extended Description	Module	Label
6 The lock manager tests for a hierarchy violation. There are two violations that can occur during a release. • The caller tries to release a local lock while holding a cross memory services lock. • The caller tries to release one of the cross memory services locks while holding all of them. If one of these occurs, IEAVESLK abnormally terminates the caller with an X'073' completion code.		
7 If this processor does not hold the lock, IEAVESLK returns to the caller with a return code in register 13. If no processor holds the lock, the return code is 4. If another processor owns the lock, the return code is 8. If the caller requested that a CML or local lock that the caller does not hold be released, the return code is 12. If this processor owns the lock, the lock manager releases it by setting the lockword to zeros.		

Diagram SUP-34. Suspend Lock Manager Processing (IEAVESLK) (Part 9 of 10)



8 Make the suspended routines dispatchable.

- For LOCAL or CML lock suspension, reschedule the suspended SRB (s).
- For CMS lock suspension, dequeue all the suspended elements.

SSRB — Schedule locally.

ASCB — Indicate that the task is ready to run.

9 Indicate that the processor no longer holds the released lock(s).

Return to caller

Diagram SUP-34. Suspend Lock Manager Processing (IEAVESLK) (Part 10 of 10)

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

8 When releasing a local lock on which no CML requestor is suspended, IEAVESLK:

- Dequeues the first suspended SRB on the lock's suspend queue.
- Stores the ready-to-run ID in the lockword (ASCBLOCK).
- Sets the SRBHLHI field to indicate that the unit of work holds the LOCAL lock.
- Sets the local lock field (PSALOCAL) to zero.
- Schedules the SRB.

If a CML requestor is suspended on the local lock, IEAVESLK:

- Sets the lockword (ASCBLOCK) to zero.
- Sets the local-lock-held flag (PSACCLHS) to zero.
- Sets the local lock field (PSALOCAL) to zero.
- Dequeues and reschedules every SRB on the lock's suspend queue.

For cross memory services locks, IEAVESLK dequeues all of the ASCBs and SSRBs on the lock's suspend queue. For each SRB released, IEAVESLK reschedules the SRB locally.

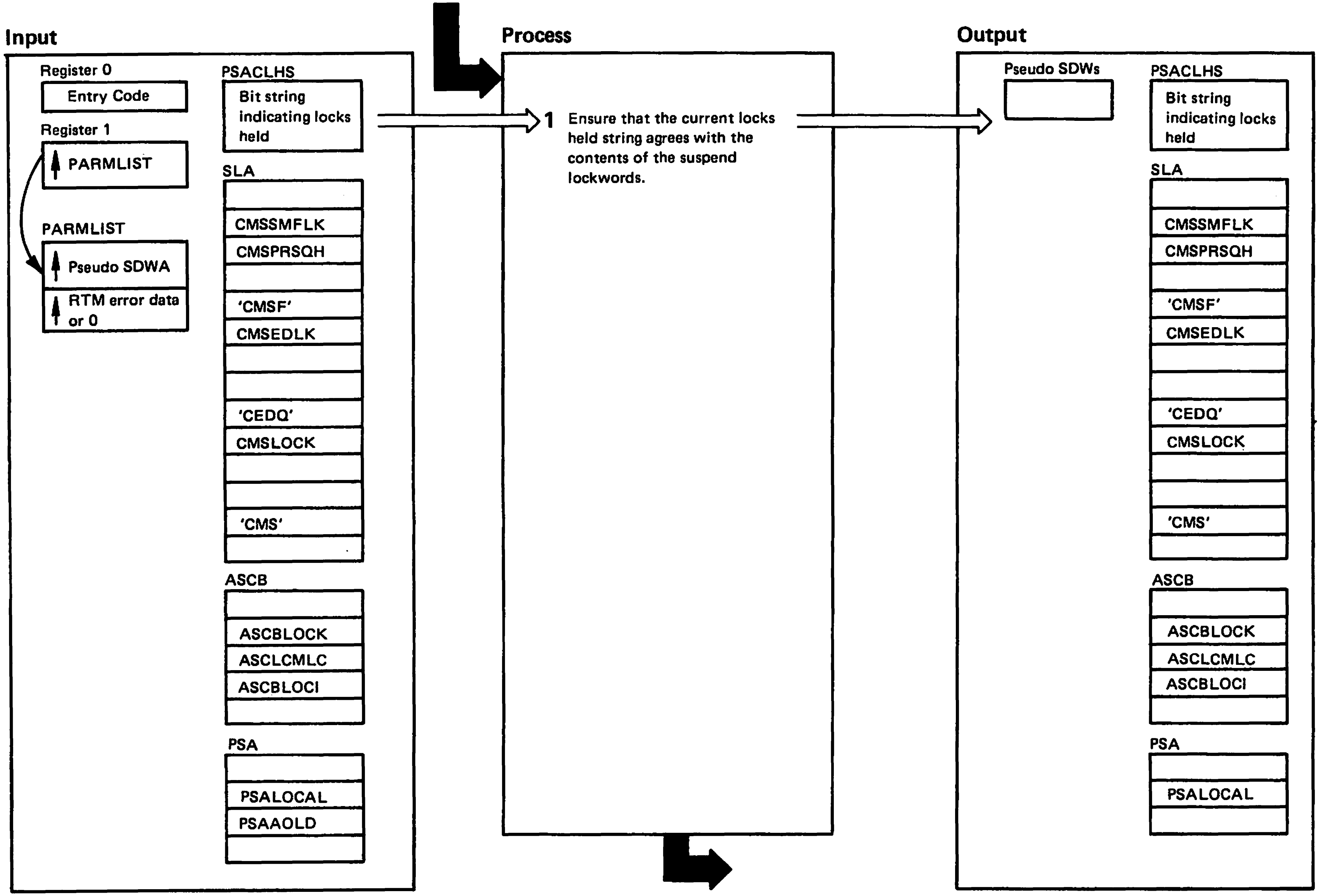
For each ASCB dequeued, IEAVESLK calls the STOP/RESET routine (IEAVESRT) to mark the task dispatchable. (See the IEAVESRT diagram for details).

9 IEAVESLK updates the PSACLHS field to show that the processor no longer holds the released locks, and returns to the caller with a 0 in register 13.

Recovery Processing:

Lock recovery processing is described in the diagram "Suspend Lock Manager Repair Router (IEAVESLR)". The FRR routine for suspend lock manager, IEAVLKRR, is an entry point within IEAVESLR.

Diagram SUP-35. Suspend Lock Repair Routine (IEAVESLR) (Part 1 of 2)

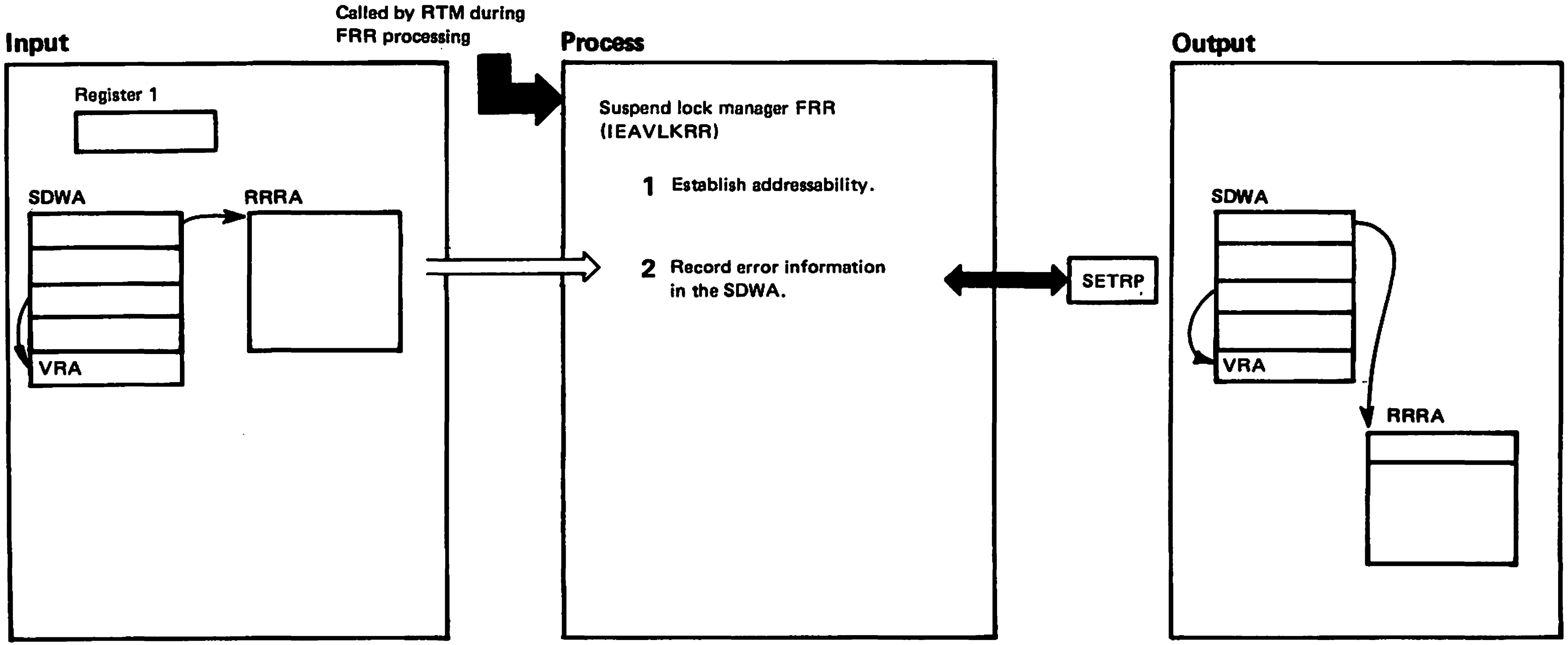


1 Ensure that the current locks held string agrees with the contents of the suspend lockwords.

Diagram SUP-35. Suspend Lock Repair Routine (IEAVESLR) (Part 2 of 2)

Extended Description	Module	Label
The suspend lock repair routine correlates the current locks held string to the suspend lockwords with the assumption that a double error has not occurred (i.e., the lockword and the current locks held string are not both invalid), and assures that a valid current locks held string, when matched against the lockwords, does exist. If an invalid state is detected, the suspend lock repair routine either alters the current locks held string or corrects the lockword, depending on the detected state. In extreme cases, the routine steals a lock or locks in order to allow the system to continue. If either a DAT error occurs while a CMS lock owner is suspended, or a restart interrupt occurs while a CMS lock owner is suspended and the system is in a wait situation (unless the lock owner is suspended on a CMS suspend queue as a result of an unconditional obtain request for all CMS locks), the routine steals all CMS locks owned by the lock owner. In certain cases, a lock will not be corrected even though an invalid value may be in the lock. In this case, rather than stealing a resource away from a possible owner, the lock is made unavailable for use until all CPUs involved are waiting on the lockword. At this time, depressing the restart key is an appropriate action which enables this routine to clear the unavailable lock.		
The suspend lock repair routine logs repair in the VRA portion of the pseudo SDWA. If an invalid state is detected and the lockword is modified, the pseudo SDWA area is provided by the caller.		

Diagram SUP-36. Suspend Lock Manager FRR Routine (IEAVESLR) (Part 1 of 4)



Suspend lock manager FRR (IEAVLKRR)

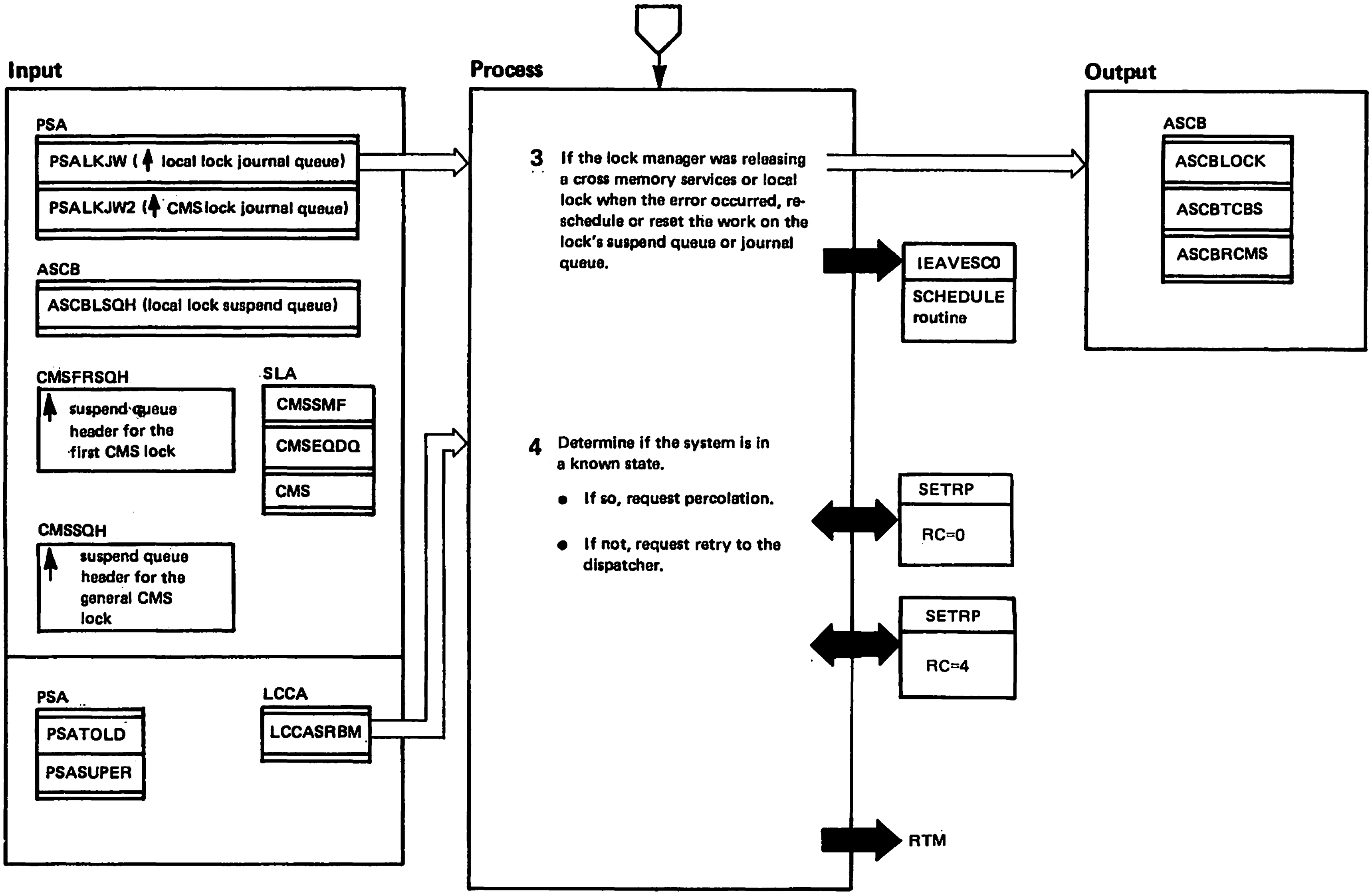
1 Establish addressability.

2 Record error information in the SDWA.

Diagram SUP-36. Suspend Lock Manager FRR Routine (IEAVESLR) (Part 2 of 4)

Extended Description	Module	Label
IEAVESLR contains the suspend lock manager FRR routine and the suspend lock repair routine. The suspend lock manager (IEAVESLK) places the suspend lock manager FRR (IEAVLKRR) on the FRR stack.		
1 IEAVLKRR establishes module addressability and establishes addressability to SDWA and VRA.		
2 IEAVLKRR logs information in the SDWA and VRA. It uses lock recovery routine recording area (RRRA) to communicate to the FRR routine the type of processing being performed. The RRRA is logged in the VRA for debugging information. IEAVESLK indicates (via the SETRP macro) to record the logged information.	IEAVESLR	LOCKFRR

Diagram SUP-36. Suspend Lock Manager FRR Routine (IEAVESLR) (Part 3 of 4)


Input
PSA
PSALKJW (↑ local lock journal queue)
PSALKJW2 (↑ CMS lock journal queue)
ASCB
ASCBLSQH (local lock suspend queue)
CMSFRSQH
↑ suspend queue header for the first CMS lock
SLA
CMSSMF
CMSEQDQ
CMS
CMSSQH
↑ suspend queue header for the general CMS lock
PSA
PSATOLD
PSASUPER
LCCA
LCCASRBM
Process
IEAVESC0
SCHEDULE routine
SETRP
RC=0
SETRP
RC=4
RTM
Output
ASCB
ASCBLOCK
ASCBTCBS
ASCBRCMS

3 If the lock manager was releasing a cross memory services or local lock when the error occurred, reschedule or reset the work on the lock's suspend queue or journal queue.

4 Determine if the system is in a known state.

- If so, request percolation.
- If not, request retry to the dispatcher.

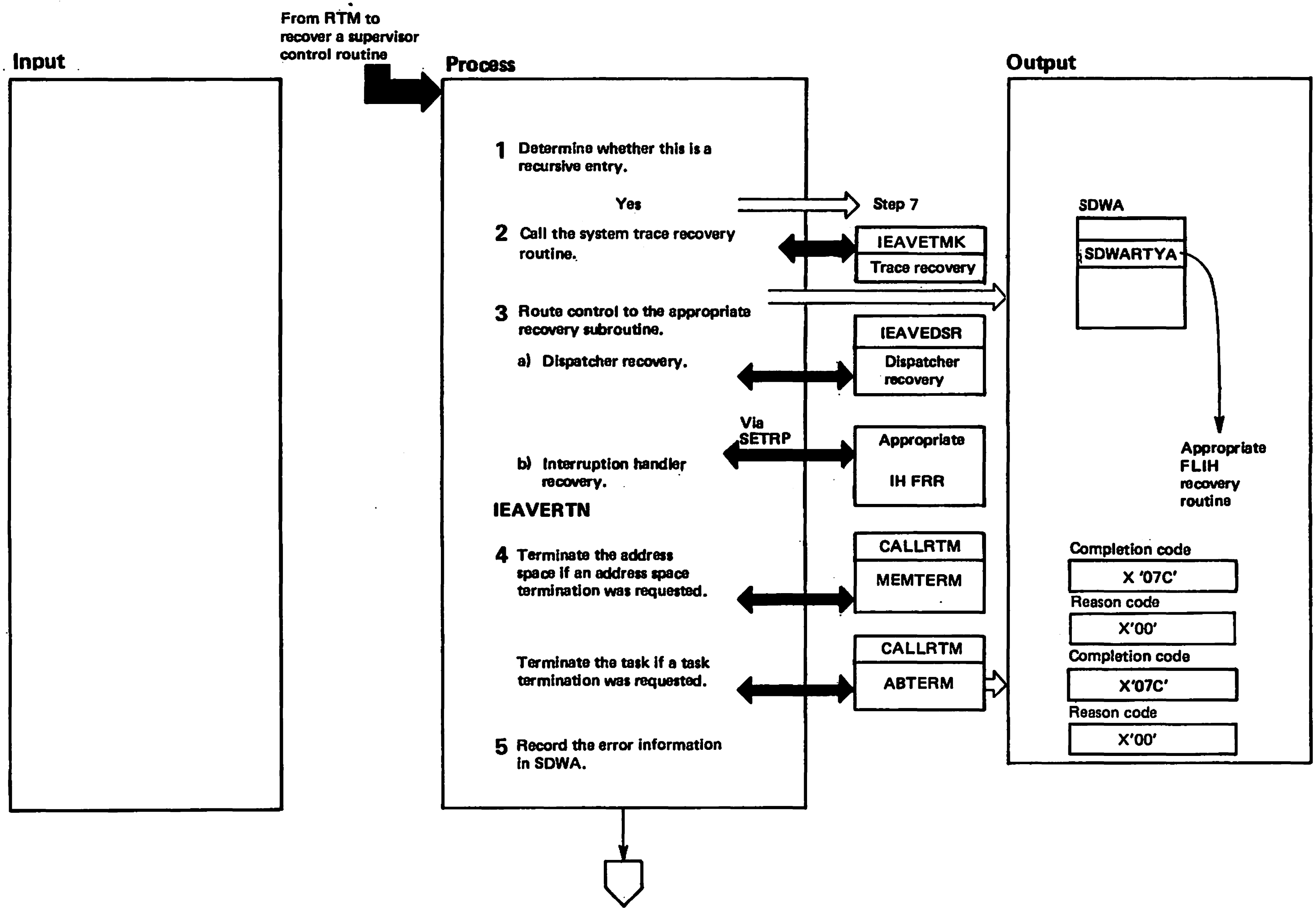
Extended Description	Module	Label

3 IEAVESLK was releasing LOCAL or CML lock at the time of the error, IEAVKLRR reschedules all of the suspended SRBs on the local journal queue or the ASCB lock suspend queue.

If IEAVESLK was releasing a CML lock at the time of the error, IEAVKLRR resets each ASCB and reschedules each SRB suspended on the CMS lock suspend queue or the CMS journal queue.

4 IEAVKLRR determines if the system is in a known state (SRB mode, task mode, or supervisor system mode) by checking the LCCASRBM bit, the PSATOLD field, and the PSASUPER bits. If the system is in a known state, IEAVKLRR issues a SETRP macro to request percolation to the next FRR. If the system is in an unknown state, IEAVKLRR issues a SETRP requesting retry at entry point IEAODS in the dispatcher.

Diagram SUP-37. Super FRR (IEAVESPR) (Part 1 of 4)



1 Determine whether this is a recursive entry.

Yes → Step 7

2 Call the system trace recovery routine.

3 Route control to the appropriate recovery subroutine.

a) Dispatcher recovery.

b) Interruption handler recovery.

IEAVERTN

4 Terminate the address space if an address space termination was requested.

Terminate the task if a task termination was requested.

5 Record the error information in SDWA.

Diagram SUP-37. Super FRR (IEAVESPR) (Part 2 of 4)

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

The super FRR determines the routines processing when an error occurred, routes control to that routine's recovery routine (if one exists) and performs actions based on return information.

1 The super FRR checks for a recursive entry. Control goes to step 7 for recursive entries; otherwise, processing continues. If a DAT error occurred, super FRR requests an address space termination (see step 4). IEAVESPR

2 The system trace recovery routine is called to allow clean up and recovery of system trace fields.

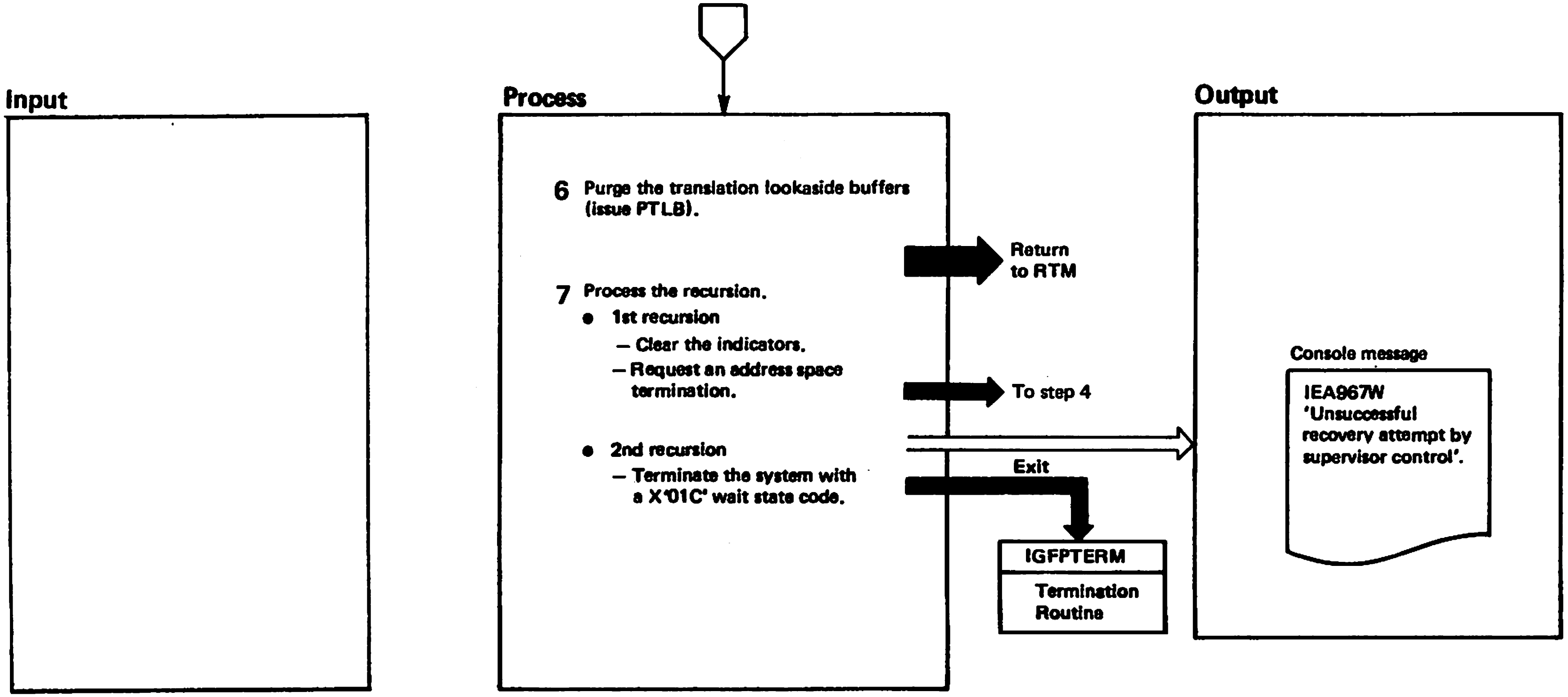
2 The super FRR uses SETRP to indicate a retry address to one of the FLIH recovery routines. After super FRR returns to RTM, RTM routes control to the specified retry address. The recovery routines that protect the dispatcher and the interruption handler are:

- Dispatcher – IEAVEDSR
- RTM – IEAVTRTF
- SVC IH – IEAVESVR
- I/O IH – IEAVEIOR
- External IH – IEAVEE1R, IEAVEE2R, and IEAVEE3R
- Machine check IH – IGFPMSUP
- Program check IH – IEAVEPCR
- Restart IH – IEAVERER

4 The super FRR, after receiving control back from the recovery routine, will terminate the address space or the task, as requested by the dispatcher FRR or as in step 7. The MEMTERM or ABTERM completion code is X '07C' and the reason code is zero.

5 The super FRR records error information in the SDWA (system diagnostic work area).

Diagram SUP-37. Super FRR (IEAVESPR) (Part 3 of 4)



6 Purge the translation lookaside buffers (issue PTLB).

7 Process the recursion.

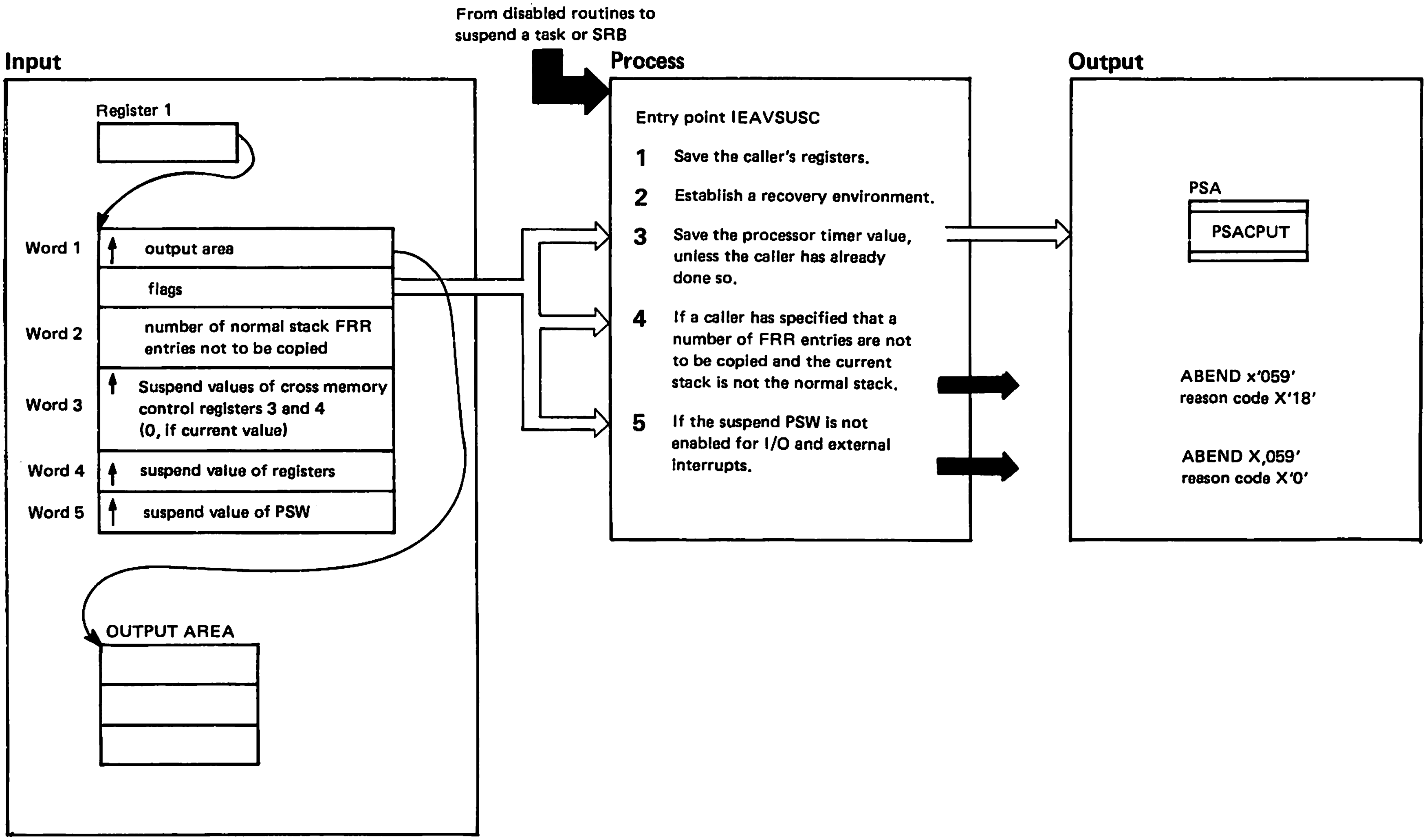
- 1st recursion
  - Clear the indicators.
  - Request an address space termination.
- 2nd recursion
  - Terminate the system with a X'01C' wait state code.

IEA967W 'Unsuccessful recovery attempt by supervisor control'.

Diagram SUP-37. Super FRR (IEAVESPR) (Part 4 of 4)

Extended Description	Module	Label
6 The super FRR purges the translation lookaside buffers via a PTLB (purge translation lookaside buffers) instruction. Control returns to RTM when the PTLB operation completes. RTM will retry to the appropriate FLIH or dispatcher retry routine whose address was put into the SDWA retry field, SDWARTYA.		
7 For one recursion, the super FRR terminates the home address space in which the error occurred. If a second recursion occurs during super FRR processing, the system will be terminated. System termination prints an IEA967W message at the console: 'Unsuccessful recovery attempt by supervisor control'. The super FRR issues a system wait state code of X'01C'.	IGFPTERM	

Diagram SUP-38. STOP/RESET Service Routine (IEAVESRT) (Part 1 of 22)



From disabled routines to suspend a task or SRB
Input
Register 1
Word 1
output area
flags
Word 2
number of normal stack FRR entries not to be copied
Word 3
Suspend values of cross memory control registers 3 and 4 (0, if current value)
Word 4
suspend value of registers
Word 5
suspend value of PSW
OUTPUT AREA
Process
Output
PSA
PSACPUT
ABEND x'059' reason code X'18'
ABEND X,059' reason code X'0'

Entry point IEAVSUSC

1 Save the caller's registers.

2 Establish a recovery environment.

3 Save the processor timer value, unless the caller has already done so.

4 If a caller has specified that a number of FRR entries are not to be copied and the current stack is not the normal stack.

5 If the suspend PSW is not enabled for I/O and external interrupts.

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

IEAVESRT performs one of two functions. It either saves status and suspends a task or SRB, or restores status and makes a suspended task or SRB redispatchable. Suspend processing is described in steps 1-25. Reset processing is described in steps 26-28.

IEAVESRT has two entry points:

IEAVSUSC – Used to suspend a task or SRB

IEAVRSTC – Used to reset a task or SRB

Input to IEAVESRT is shown in the input section of the diagram opposite each entry point.

IEAVESRT contains two recovery routines:

STOPFRR – The FRR entry point for recovery for Suspend processing

RESETFRR – The FRR entry point for recovery for Reset processing

1 IEAVESRT saves the caller's registers in the save area pointed to by register 13. IEAVESRT IEAVSUSC

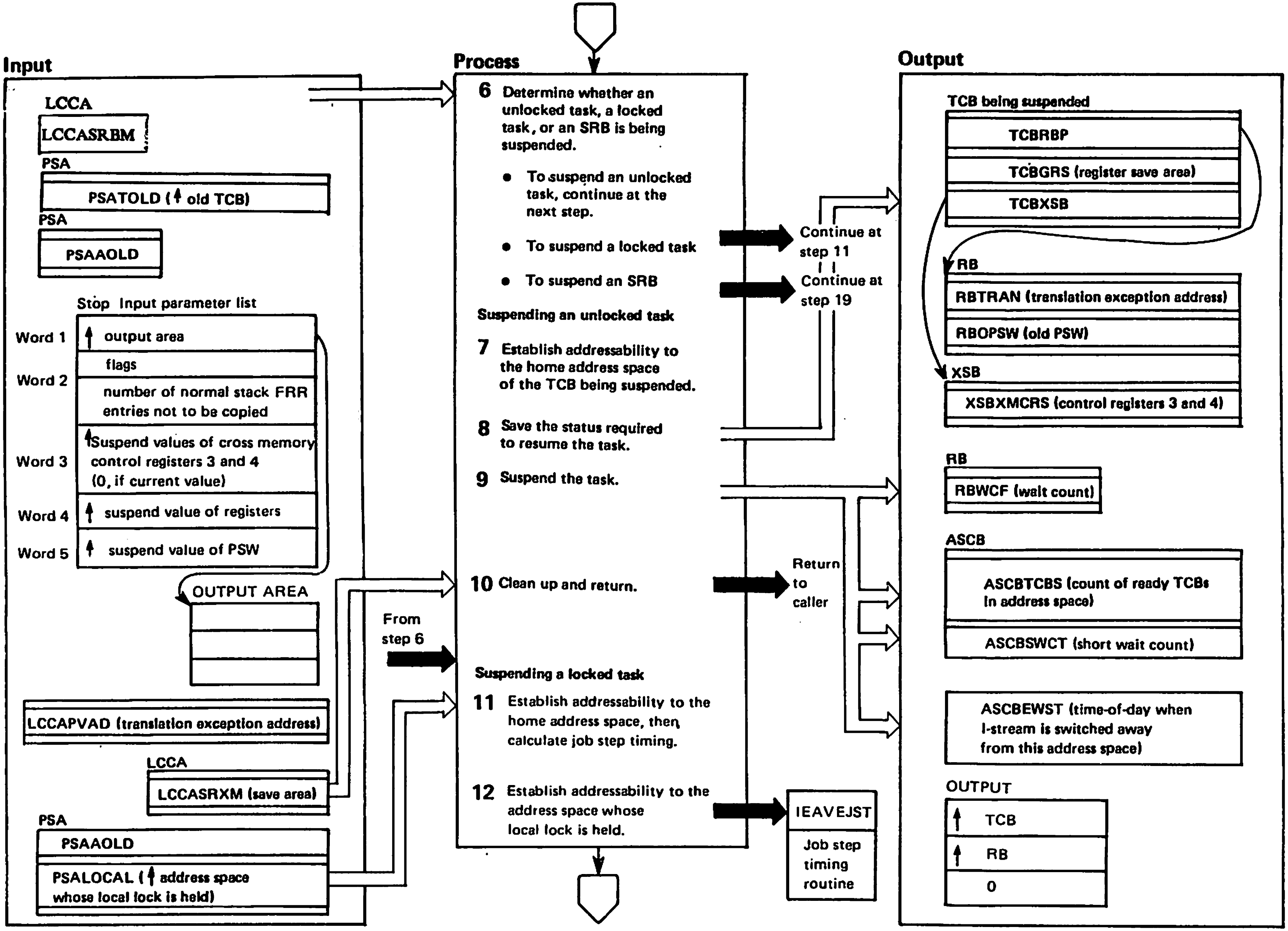
2 IEAVESRT establishes STOPFRR as the recovery routine for suspend processing. For more information, see "Recovery for Suspend Processing" at the end of this extended description.

3 Unless the processor timer is damaged, IEAVESRT records the processor time in the PSA, unless the caller has already done so.

4 If the caller has specified a number of FRR entries not to be copied to the normal stack, and the current stack is not the normal stack, then the caller is abnormally terminated with a completion code of X'059' and a reason code of X'18'.

5 If the PSW at the time of the suspend request is not enabled for external and I/O interrupts, IEAVESRT issues ABEND X'059' with reason code 0.

Diagram SUP-38. STOP/RESET Service Routine (IEAVESRT) (Part 3 of 22)



Input

LCCA

LCCASRBM

PSA

PSATOLD (↑ old TCB)

PSA

PSAAOLD

Stop Input parameter list

Word 1	↑ output area
Word 2	flags number of normal stack FRR entries not to be copied
Word 3	↑Suspend values of cross memory control registers 3 and 4 (0, if current value)
Word 4	↑ suspend value of registers
Word 5	↑ suspend value of PSW

OUTPUT AREA

LCCAPVAD (translation exception address)

LCCA

LCCASRXM (save area)

PSA

PSAAOLD

PSALOCAL (↑ address space whose local lock is held)

Process

6 Determine whether an unlocked task, a locked task, or an SRB is being suspended.

- To suspend an unlocked task, continue at the next step.
- To suspend a locked task — Continue at step 11
- To suspend an SRB — Continue at step 19

Suspending an unlocked task

7 Establish addressability to the home address space of the TCB being suspended.

8 Save the status required to resume the task.

9 Suspend the task.

10 Clean up and return. — Return to caller

From step 6

Suspending a locked task

11 Establish addressability to the home address space, then calculate job step timing.

12 Establish addressability to the address space whose local lock is held. — IEAVEJST Job step timing routine

Output

TCB being suspended

TCBRBP

TCBGRS (register save area)

TCBXSB

RB

RBTRAN (translation exception address)

RBOPSW (old PSW)

XSB

XSBXMCRS (control registers 3 and 4)

RB

RBWCF (wait count)

ASCB

ASCBTCBS (count of ready TCBs in address space)

ASCBSWCT (short wait count)

ASCBEWST (time-of-day when I-stream is switched away from this address space)

OUTPUT

↑ TCB

↑ RB

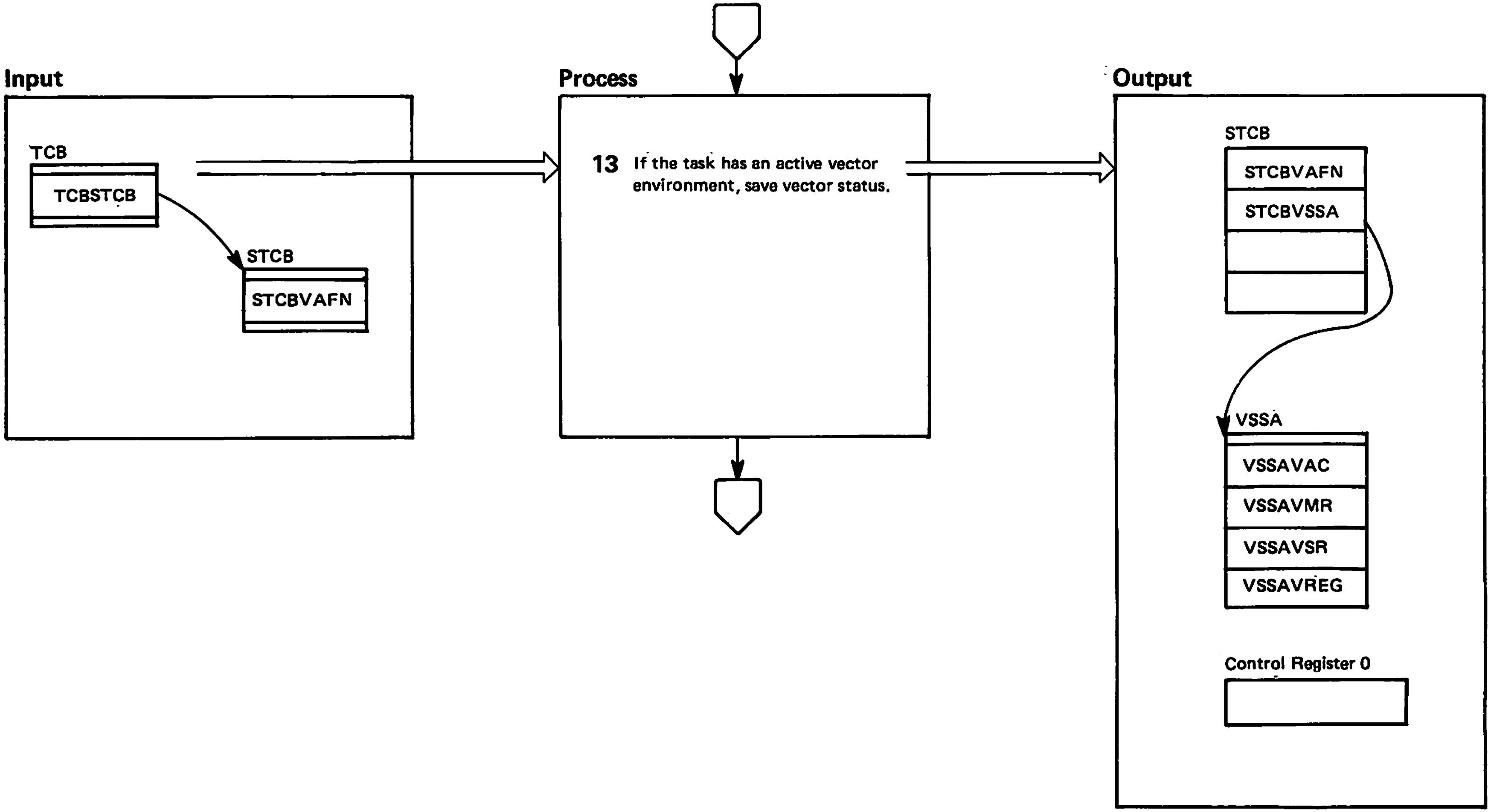
0

Diagram SUP-38. STOP/RESET Service Routine (IEAVESRT) (Part 4 of 22)

Extended Description	Module	Label
6 If the processor is in task mode and the task being suspended holds the LOCAL or CML lock, processing continues at step 9. If the processor is in task mode but neither of these locks is held, processing continues at the next step. If the processor is in SRB mode, processing continues at step 18.		
Suspending an Unlocked Task		
7 IEAVESRT issues a CMSET SET macro to make the home address space of the TCB being suspended the primary address space.	IEAVESRT	UNLKTCB
If the current RB has a non-zero wait count (RBWCF), IEAVESRT abends the caller with a completion code of X'059' reason code X'0C'. This indicates a request to stop an unlocked task that is already stopped.		
8 IEAVESRT saves the current status in the TCB being suspended and in its associated XSB and RB. The status saved includes: ● Cross memory control registers. 3 and 4. ● General purpose registers. ● The PSW at the time of the suspend request. ● The translation exception address (TEA). When the suspend request is not the result of a page fault, the TEA (the LCCAPVAD field) is zero.		
To suspend the task, IEAVESRT set the wait count in the current RB (RBWCF) to one and subtracts one from the count of ready TCBs in the home address space (ASCBTCBS).		
If the count of TCBs requiring the local lock (ASCBTCBL) is zero. IEAVESRT takes a time stamp in ASCBEWST, and increases the current short wait count (ASCBSWCT) by one.		

Extended Description	Module	Label
10 IEAVESRT restores the caller's cross memory state, deletes the FRR, loads the 3 words of the output area with values shown on the output section of the diagram (the value of the TCB, the value of RB and zero, respectively), and return to caller.		
Suspending a Locked Task		
11 IEAVESRT issues a CMSET SET macro to make the home address space the primary address space.		
IEAVESRT calls the job step timing module (IEAVEJST) to calculate elapsed job step time.		
12 IEAVESRT issues a CMSET SET macro to make the address space whose local lock is held the primary address space.	IEAVESRT	LOCKTCB

Diagram SUP-38. STOP/RESET Service Routine (IEAVESRT) (Part 5 of 22)



13 If the task has an active vector environment, save vector status.

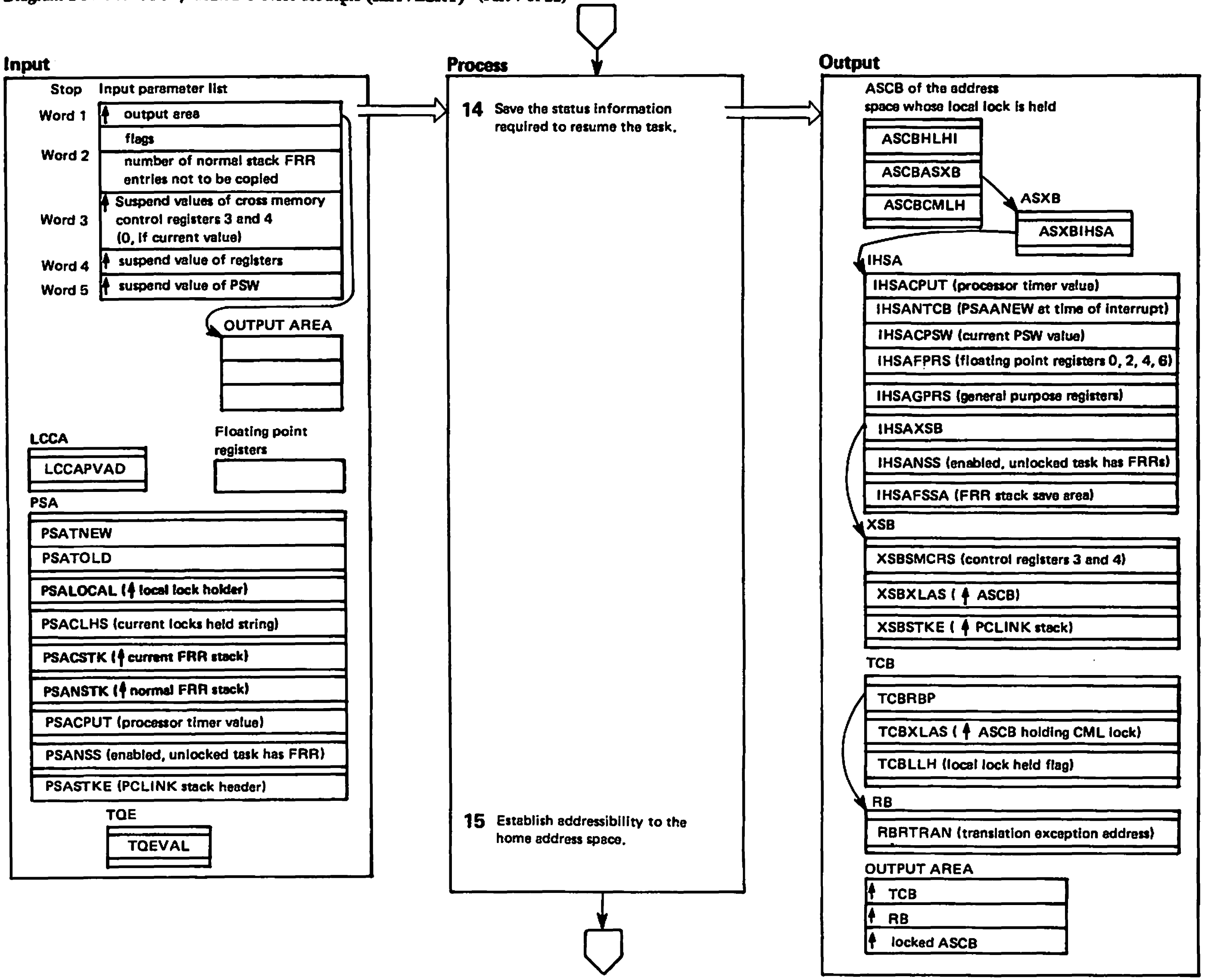
Diagram SUP-38. STOP/RESET Service Routine (IEAVESRT) (Part 6 of 22)

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

13 If the task has an active vector environment, IEAVESRT will:

- Store the vector activity count in the VSSAVAC.
- Save the vector registers in the task's VSSA.
- Place a X'20' value in STCBVAFN to indicate the vector registers are in the VSSA.
- Set bit 14 in control register 0 to zero to prevent execution of vector instructions.

Diagram SUP-38. STOP/RESET Service Routine (IEAVESRT) (Part 7 of 22)



Input

Stop Input parameter list

Word 1	output area
	flags
Word 2	number of normal stack FRR entries not to be copied
Word 3	Suspend values of cross memory control registers 3 and 4 (0, if current value)
Word 4	suspend value of registers
Word 5	suspend value of PSW

OUTPUT AREA

LCCA

- LCCAPVAD

Floating point registers

PSA

- PSATNEW
- PSATOLD
- PSALOCAL (local lock holder)
- PSACLHS (current locks held string)
- PSACSTK (current FRR stack)
- PSANSTK (normal FRR stack)
- PSACPUT (processor timer value)
- PSANSS (enabled, unlocked task has FRR)
- PSASTKE (PCLINK stack header)

TQE

- TQEVAL

Process

14 Save the status information required to resume the task.

15 Establish addressibility to the home address space.

Output

ASCB of the address space whose local lock is held

- ASCBHLHI
- ASCBASXB
- ASCBCMLH

ASXB

- ASXBIHSA

IHSA

- IHSACPUT (processor timer value)
- IHSANTCB (PSAANEW at time of interrupt)
- IHSACPSW (current PSW value)
- IHSAFPRS (floating point registers 0, 2, 4, 6)
- IHSAGPRS (general purpose registers)
- IHSAXSB
- IHSANSS (enabled, unlocked task has FRRs)
- IHSAFSSA (FRR stack save area)

XSB

- XSBSMCRS (control registers 3 and 4)
- XSBXLAS (ASCB)
- XSBSTKE (PCLINK stack)

TCB

- TCBRBP
- TCBXLAS (ASCB holding CML lock)
- TCBLLH (local lock held flag)

RB

- RBRTRAN (translation exception address)

OUTPUT AREA

- TCB
- RB
- locked ASCB

Diagram SUP-38. STOP/RESET Service Routine (IEAVESRT) (Part 8 of 22)

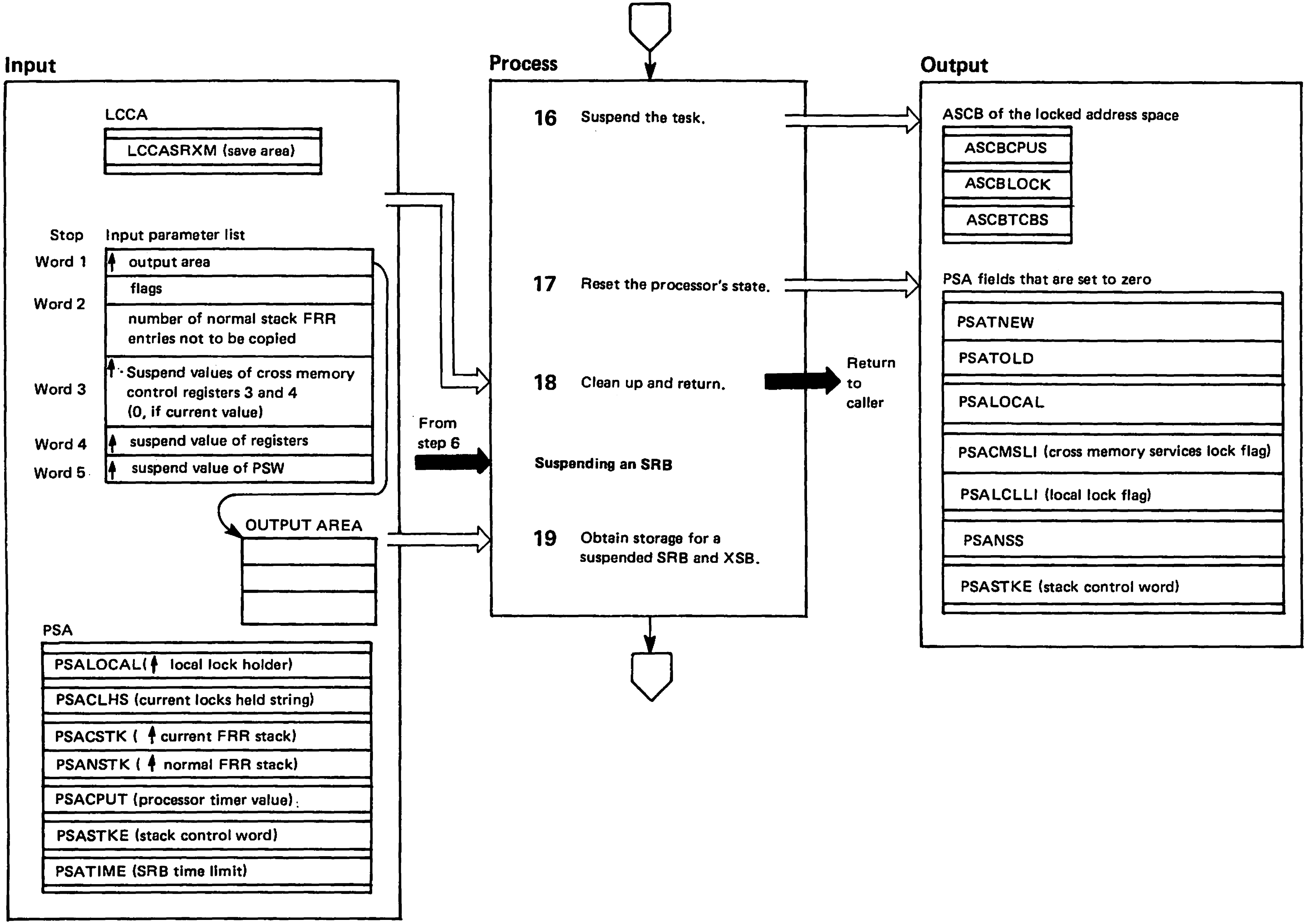
Extended Description	Module	Label
14 IEAVESRT saves the following status information in the locked address space's IHSA and the XSB chained from it.	IEAVESRT	LOCKTCB

- Cross memory control registers 3 and 4 in the XSB
- The PCLINK stack header in the XSB
- The address of the locked ASCB in the IHSA's XSB and TCB
- Current floating point registers 0, 2, 4, and 6 in the IHSA
- General purpose registers and the PSW at the time of the suspend request in the IHSA
- The processor timer value in the IHSA
- The entire normal FRR stack in the IHSA, or if the caller has specified a number of FRR entries not to be copied, saves the normal FRR stack minus the specified number of FRR entries.
- The value of the enabled, unlocked task FRR flag in the IHSA
- The current and old TCB addresses in the IHSA
- The LOCAL- or CML-lock-held indicator in the ASCB, and the local-lock-held indicator in the TCB
- The TCB address in the ASCB

IEAVESRT also stores a pointer to the TCB, a pointer to the RB, and a pointer to the locked ASCB into the output area. These values are returned to the caller at exit.

15 If the primary address space is not the home address space, IEAVESRT issues the CMSET SET macro to make the home address the primary address space.

Diagram SUP-38. STOP/RESET Service Routine (IEAVESRT) (Part 9 of 22)



Input

LCCA

- LCCASRXM (save area)

Stop Input parameter list

Word 1	↑ output area
	flags
Word 2	number of normal stack FRR entries not to be copied
Word 3	↑· Suspend values of cross memory control registers 3 and 4 (0, if current value)
Word 4	↑ suspend value of registers
Word 5	↑ suspend value of PSW

OUTPUT AREA

PSA

- PSALOCAL(↑ local lock holder)
- PSACLHS (current locks held string)
- PSACSTK (↑ current FRR stack)
- PSANSTK (↑ normal FRR stack)
- PSACPUT (processor timer value)
- PSASTKE (stack control word)
- PSATIME (SRB time limit)

Process

16 Suspend the task.

17 Reset the processor's state.

18 Clean up and return.

Return to caller

From step 6

Suspending an SRB

19 Obtain storage for a suspended SRB and XSB.

Output

ASCB of the locked address space

- ASCBCPUS
- ASCBLOCK
- ASCBTCBS

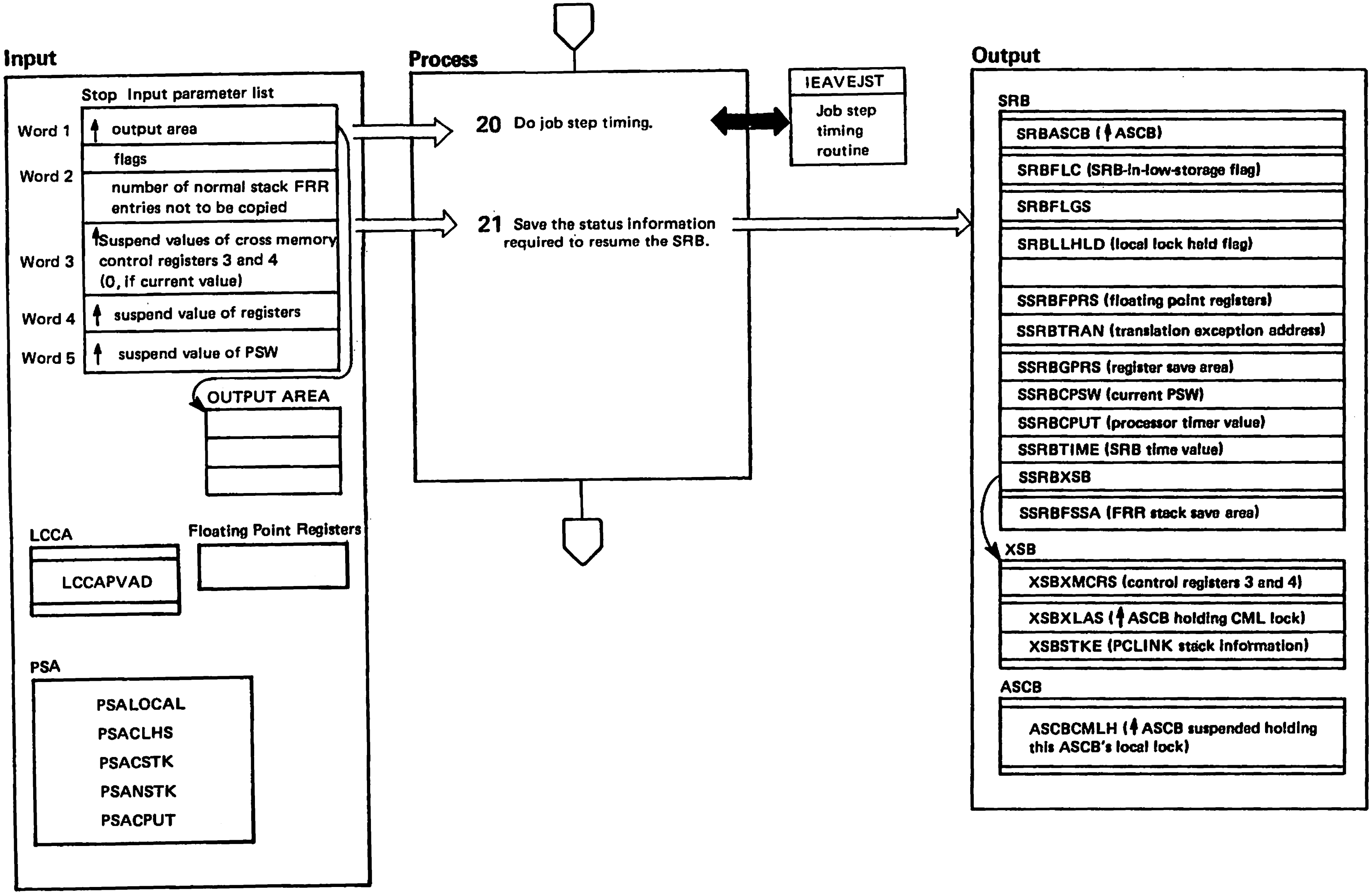
PSA fields that are set to zero

- PSATNEW
- PSATOLD
- PSALOCAL
- PSACMSLI (cross memory services lock flag)
- PSALCLLI (local lock flag)
- PSANSS
- PSASTKE (stack control word)

Diagram SUP-38. STOP/RESET Service Routine (IEAVESRT) (Part 10 of 22)

Extended Description	Module	Label
16 To suspend the task, IEAVESRT:	IEAVESRT	LOCKASCB
• Puts the suspend ID into the ASCB lockword • Subtracts one from the count of ready TCBs in the suspended task's home ASCB • Subtracts one from the count of active processor's in task mode		
If the count of TCB's requiring the local lock (ASCBTCBL) is zero, IEAVESRT takes a time stamp (via the STCK instruction) in ASCBEWST, and increases the current short wait count (ASCBWCT) by one.		
17 To reset the processor's state, IEAVESRT sets to zero the fields shown in the output section of the diagram.		
18 IEAVESRT restores the caller's cross memory status, deletes the FRR, and returns to the caller.	IEAVESRT	USERMODE
Suspending an SRB		
19 IEAVESRT issues a GETSSRB macro to obtain storage for an SSRB and XSB in which to save status.	IEAVESRT	SRBSPND

Diagram SUP-38. STOP/RESET Service Routine (IEAVESRT) (Part 11 of 22)



Input

Stop Input parameter list

- Word 1: ↑ output area
- Word 2: flags; number of normal stack FRR entries not to be copied
- Word 3: ↑Suspend values of cross memory control registers 3 and 4 (0, if current value)
- Word 4: ↑ suspend value of registers
- Word 5: ↑ suspend value of PSW

OUTPUT AREA

LCCA: LCCAPVAD

Floating Point Registers

PSA: PSALOCAL, PSACLHS, PSACSTK, PSANSTK, PSACPUT

Process

20 Do job step timing.

IEAVEJST: Job step timing routine

21 Save the status information required to resume the SRB.

Output

SRB

- SRBASCB (↑ASCB)
- SRBFLC (SRB-in-low-storage flag)
- SRBFLGS
- SRBLLHLD (local lock held flag)
- SSRBFPRS (floating point registers)
- SSRBTRAN (translation exception address)
- SSRBGPRS (register save area)
- SSRBCPSW (current PSW)
- SSRBCPUT (processor timer value)
- SSRBTIME (SRB time value)
- SSRBXSB
- SSRBFSSA (FRR stack save area)

XSB

- XSBXMCRS (control registers 3 and 4)
- XSBXLAS (↑ASCB holding CML lock)
- XSBSTKE (PCLINK stack information)

ASCB

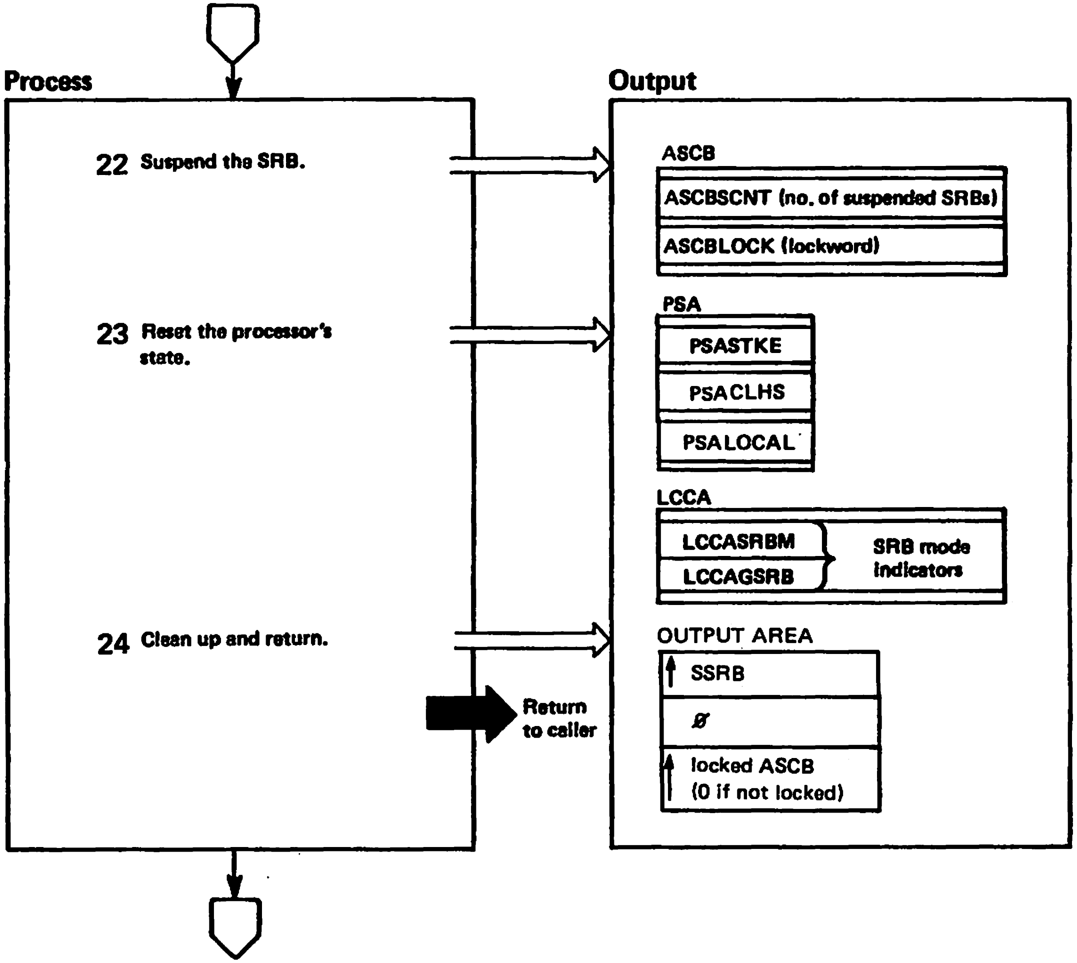
- ASCBCMLH (↑ASCB suspended holding this ASCB's local lock)

Extended Description	Module	Label
20 Call job step timing module (IEAVEJST) to calculate elapsed job step timing.		

21 IEAVESRT saves the following status information:

- Cross memory control registers 3 and 4 in the XSB
- The PSW at the time of the suspend request in the XSB
- The PCLINK stack header in the XSB
- The address of the ASCB holding the local lock in the XSB
- The current floating point registers in the SSRB
- General purpose registers and the PSW at the time of the suspend request in the SSRB
- The adjusted processor timer and SRB time values in the SSRB
- The entire normal FRR stack in the SSRB, or, if the caller has specified a number of FRR entries not to be copied (entry point IEAVSUSF), saves the normal stack minus the specified number of FRR entries.
- The CPU affinity information in the SRB
- The address of the home ASCB in the SRB
- The translation exception address in the SSRB
- The LOCAL- and CMS-lock-held indicators in the SRB and ASCB

Diagram SUP-38. STOP/RESET Service Routine (IEAVESRT) (Part 13 of 22)



Extended Description	Module	Label
25 IEAVESRT resets SSRBs, locked tasks, and unlocked tasks in different ways. Step 26 describes how an unlocked task is reset, step 27 a locked task, and step 28 an SSRB. The input parameter list points to the output area filled in by IEAVESRT when this unit of work was stopped.	IEAVESRT	RSETCOMM

If the reset type code (in the input parameter list) is not a zero (unconditional), or a four (conditional), or is an eight (page I/O error reset request), then the following reset error processing takes place:

- IEAVESRT establishes IEAVSCHF as the recovery routine for error processing designed to terminate the suspended SRB or TCB.

- If the caller is in secondary address mode, or if the reset ASCB (in the output area) is not the home ASCB, IEAVESRT establishes primary addressability to the reset address space. IEAVESRT obtains and initializes an SSRB to represent an SRB reset request. IEAVESRT schedules the SRB, deletes recovery and returns to the caller. (The SRB entry point is IEAVSRBR in module IEAVESRT.) Otherwise, if the local lock is held or if it cannot be obtained, IEAVESRT issues a CALLRTM TYPE=STERM to terminate the suspended SRB or TCB.

If the local lock was obtained, IEAVESRT releases it, deletes recovery, and returns control to the caller.

The STERM completion code is the reset type code (in the parameter list) except for the page I/O error case for which the completion code is set to X'08'.

Diagram SUP-38. STOP/RESET Service Routine (IEAVESRT) (Part 17 of 22)

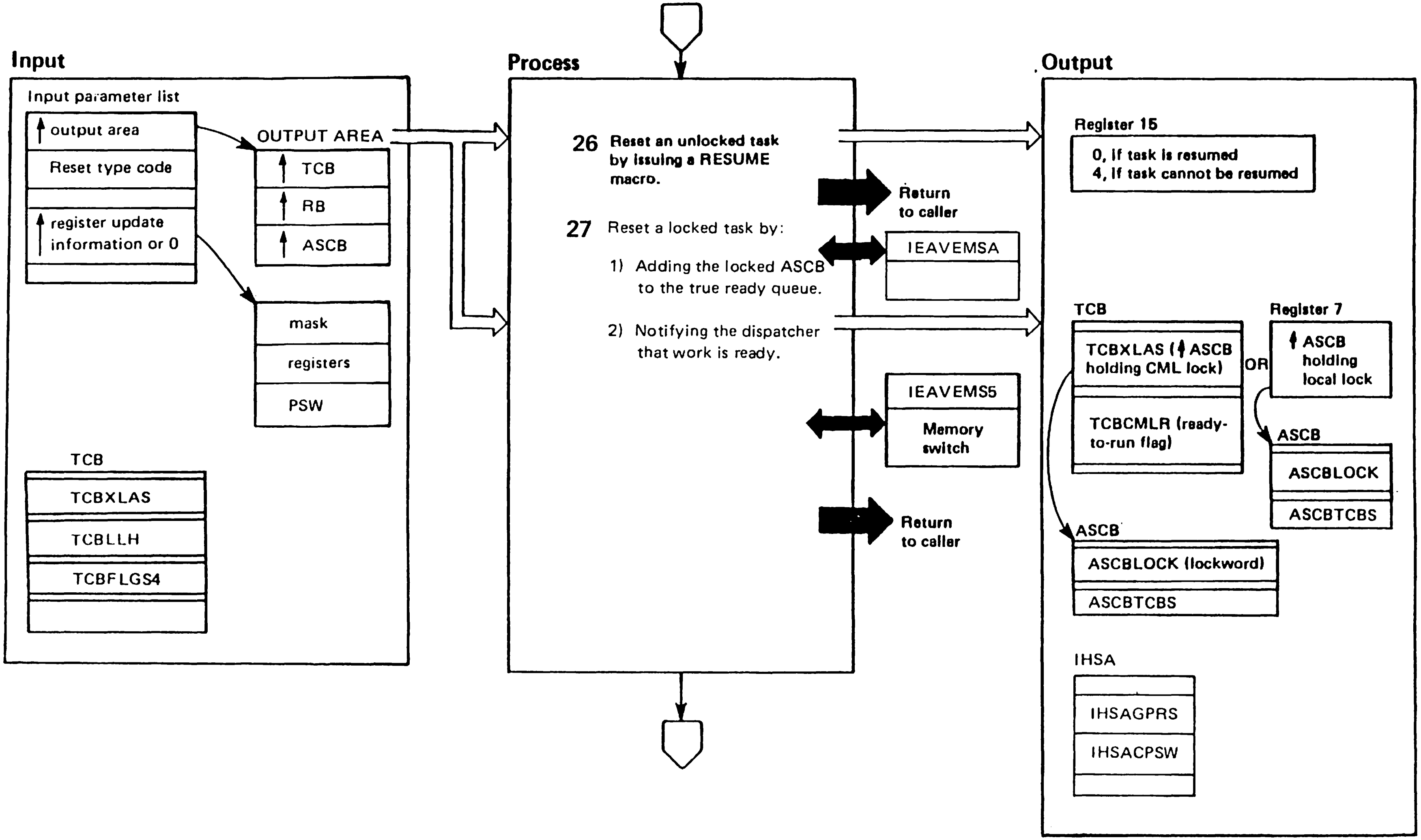
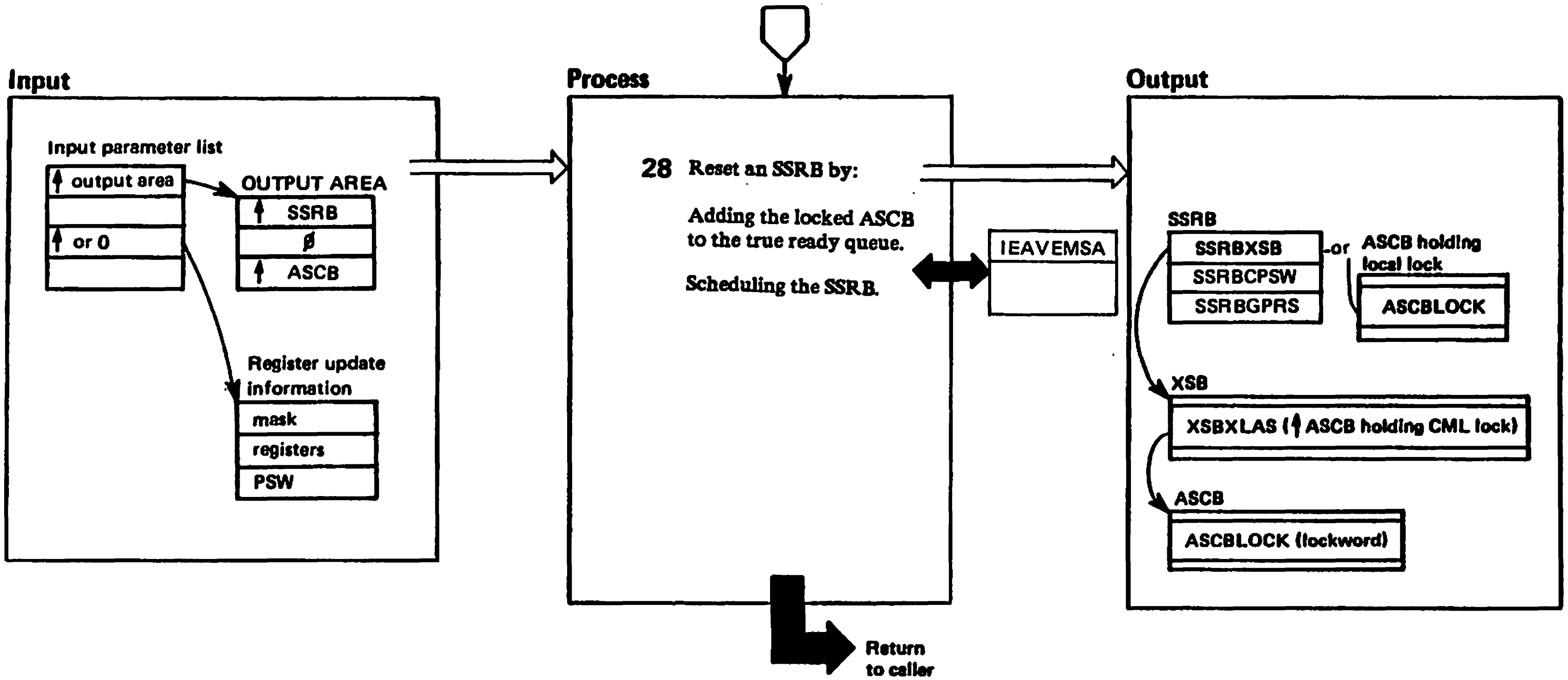


Diagram SUP-38. STOP/RESET Service Routine (IEAVESRT) (Part 18 of 22)

Extended Description	Module	Label
26 To reset an unlocked task, IEAVESRT:	IEAVESRT	NOLKTCB
• Establishes addressability to the input ASCB (the task's home address space) • Depending on the input code in register 6, issues either a conditional or an unconditional asynchronous RESUME macro • If the conditional RESUME macro fails, sets a return code of 4; otherwise, sets a return code of zero • Returns to the caller		
27 If the task is not locked (TCBLLH is zero) but the caller's input indicated locked, IEAVESRT abends the caller with a completion code of X'059' and a reason code of X'10'. Otherwise, to reset a task that holds the CML or LOCAL lock, IEAVESRT:	IEAVESRT	RSETCB
• Establishes addressability to the input ASCB • Establishes RESETFRR as the recovery routine for reset processing • If the lock held by the task does not contain the suspend ID, abends the caller with a completion code of X'059' and a reason code of X'14' • If the caller requests, update the resume registers and/or PSW with the values supplied by the caller. • If the TCB is dispatchable, IEAVESRT increases the count of ready-to-run TCBs (ASCBTCBS) by one • If the task holds the LOCAL lock, replaces the suspend ID in the ASCB lockword with an interrupt ID		
• If the task holds the CML lock, replaces the suspend ID in the CML-locked ASCB with a ready-to-run ID, and sets the ready-to-run flag in the TCB, calls Memory Switch to add the locked ASCB to the true ready queue.	IEAVESRT	RSETCML
• Calls the memory switch module (IEAVEMS5) to notify the dispatcher that work is ready • Deletes the FRR • Returns to the caller	IEAVESRT	RSETMEMS

Diagram SUP-38. STOP/RESET Service Routine (IEAVESRT) (Part 19 of 22)



28 Reset an SSRB by:

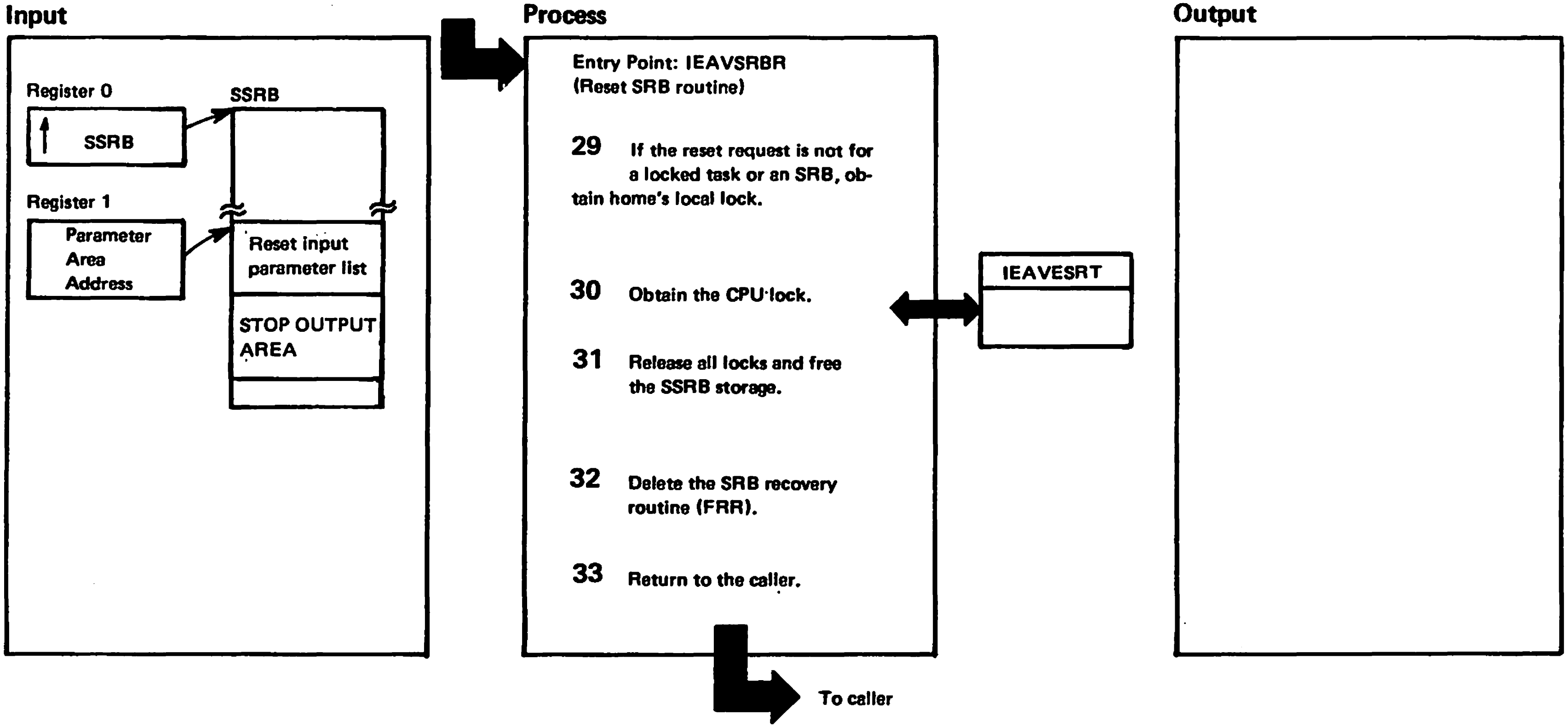
Adding the locked ASCB to the true ready queue.

Scheduling the SSRB.

Diagram SUP-38. STOP/RESET Service Routine (IEAVESRT) (Part 20 of 22)

Extended Description	Module	Label
28 To reset an SSRB: • If requested, update the resume registers and PSW (in SSRBGPRS and SSRBCPSW). • If the SSRB holds a CML or LOCAL lock, IEAVESRT replaces the suspend ID in the locked ASCB's lockword with a ready-to-run ID and adds the locked ASCB to the true ready queue. • IEAVESRT schedules the SSRB.	IEAVESRT	RSETSRB
Recovery for Suspend Processing When an error occurs while IEAVESRT is suspending a task or SRB, RTM gives control to STOPFRR, and entry point within IEAVESRT. STOPFRR records the error in the SDWA and records the FRR parameter area, the caller's ASID, the PSATOLD value, the caller's return address and the stop input parameter list in the SDWA's variable recording area. If a recursive or DAT error has occurred, or if the STOPFRR was entered as a global resource manager (SDWAGLBL is 1), STOPFRR requests percolation and returns to RTM. If IEAVESRT has finished processing the request (both the PSATOLD field and the LCCASRBM bits are zero), STOPFRR requests that RTM retry at label SFRRETRY. There IEAVESRT restores the caller's registers and returns to the caller. In all other cases, STOPFRR's actions depend on whether IEAVESRT was suspending an unlocked task, a locked task, or an SRB, and where processing the error occurred. If the PSATOLD field is nonzero, the local lock flag in the PSACLHS field is zero, and the TCBLLH flag has not yet been set, IEAVESRT was suspending an unlocked task. STOPFRR determines whether IEAVESRT altered the TCB's status. • If the status has not been altered (the RB wait count is zero), STOPFRR requests a retry at label SFRRETAB. There IEAVESRT restores the caller's registers and terminates the caller with X '059' abend and reason code X '08'. • If TCB status has been changed, STOPFRR resets the RB wait count to zero and adds one to the count of ready TCBs (ASCBTCBS) before requesting a retry at SFRRETAB. If IEAVESRT was suspending a locked task when the error occurred, STOPFRR determines whether IEAVESRT altered the processor's lock status.	IEAVESRT	STOPFRR
• If the lock status has not been changed (both the current TCB's local-lock-held flag, TCBLLH, and the locked ASCB's highest-lock-held field, ASCBHLHI, are zero), STOPFRR requests a retry at label SFRRETAB.	IEAVESRT	SRRTCBCM
• If the processor's lock state has changed, STOPFRR: – Puts the processor ID in the local lockword (only if TCBLLH=1) – Restores the PSANSS flag from the locked IHSA – Restores the PSASTKE and PSALOCAL from the value saved in the locked IHSA's XSB – Makes the PSACLHS field match the locked ASCB's ASCBHLHI field, and clears the ASCBHLHI field. – Adds one to the count of ready TCBs (ASCBTCBS) – Requests a retry at label SFRRETAB		
If IEAVESRT was suspending an SRB (LCCASRBM=1), recovery processing depends on whether IEAVESRT had changed the processor status and/or the SRB's local-lock-held field (SRBLLHLD). • If the processor status has not been changed (no SSRB has been obtained), STOPFRR requests a retry at label SFRRETAB to abend the caller.	IEAVESRT	SRRSRBM
• If the processor status has been changed, but the SRBLLHLD flag has not yet been set, STOPFRR: – Restores the PSASTKE from the value saved in the XSBSTKE field – Returns the SSRB/XSB to the SRB/SSRB pool – Requests a retry at label SFRRETAB • If the SRBLLHLD flag has been set, STOPFRR: – Puts the processor's ID into the lockword of the appropriate locked ASCB – Sets in the PSACLHS field the bits that are set in the SRBHLHI field – Restores the PSASTKE from the value saved in the XSBSTKE field – Restores PSALOCAL from the XSB (XSBXLAS) – Resets to zero the locked ASCBCMLH field of the ASCB – Returns the SSRB/XSB to the SSRB/XSB pool – Requests a retry at label SFRRETAB	IEAVESRT	SRRSRBLH

Diagram SUP-38. STOP/RESET Service Routine (IEAVESRT) (Part 21 of 22)



Entry Point: IEAVSRBR
(Reset SRB routine)

29 If the reset request is not for a locked task or an SRB, obtain home's local lock.

30 Obtain the CPU lock.

31 Release all locks and free the SSRB storage.

32 Delete the SRB recovery routine (FRR).

33 Return to the caller.

Diagram SUP-38. STOP/RESET Service Routine (IEAVESRT) (Part 22 of 22)

Extended Description	Module	Label
29 (Continued)		
Recovery for Reset Processing		
When an error occurs while IEAVESRT is resetting a locked task, RTM gives control to RESETFRR, an entry point within IEAVESRT. RESETFRR records the error in the SDWA and records the FRR parameter area, the caller's ASID, the PSATOLD value, the caller's return address, the reset input parameter list, and the stop output area in the SDWA's variable recording area. If a recursive or DAT error has occurred, RESETFRR percolates the error. Otherwise, RESETFRR attempts to finish resetting the locked task as described in step 26. It then requests a retry at label RFRRETRY, where IEAVESRT restores the caller's cross memory status and registers, and returns to the caller.	IEAVESRT	RESETFRR
Recovery for Reset Error Processing		
When an error occurs while IEAVESRT is processing an invalid RESET request, RTM gives control to IEAVSCHF, an entry point within IEAVESRT. IEAVSCHF records the error in the SDWA and records the FRR parameter area, the caller's ASID, the current task pointer (PSATOLD), and all the input parameters in the SDWA's variable recording area (VRA). If an SSRB pointer is available in the FRR parameter area (indicating that a reset schedule was in progress) then IEAVSCHF records the entire SRB in the VRA and frees the SSRB. If the CPU lock was obtained by the SRB routine, IEAVSCHF records this information in the VRA and frees the CPU lock. If the local lock was obtained by the SRB routine, IEAVSCHF records this information in the VRA and frees the local lock. The recovery routine then percolates to the next level of recovery.	IEAVESRT	IEAVSCHF

Extended Description	Module	Label
30 Check the output area filled in by STOP. If the RB address is zero (indicating that an SRB is requested) or bit 0 of the TCB address is one (indicating a locked TCB), then do not obtain home's local lock.	IEAVESRT	IEAVSRBR
31 Obtains the CPU lock to disable external and I/O interrupts.		
32 Releases the CPU lock and then the local lock. Converts the SRBID field to an SSRB ID and releases the SSRB storage.		
33 Deletes the FRR from the FRR stack.		

SUP-39. DYNAMIC SVC TABLE ENTRY INSTALLER (IEAVESTU)

IEAVESTU - MODULE DESCRIPTION

DESCRIPTIVE NAME: DYNAMIC SVC TABLE UPDATE SERVICE

FUNCTION:

The SVC Dynamic Update Service updates entries in the System SVC Table for "REPLACE/DELETE" requests. The old SVC Table Entry and other data about the update request is stored in the SVC Update Recording Table. It returns information from the SVC Table for "EXTRACT" requests.

ENTRY POINT: IEAVESTU

PURPOSE: Main entry point of the module. See Operation Section.

LINKAGE:
"SVCUPDTE" generates "BASSM" with register 1
pointing to the parameter list

CALLERS:
(via SVCUPDTE macro interface)
IEAVNP25,
IEAVNPS5,
IEAVNPST
Any supervisor state/key zero routine

INPUT:
- An SVCUPDTE Parameter list with the address passed in Register 1.

OUTPUT:
REPLACE/DELETE requests:
- SVC Table is updated.
- SVC Update Recording Table is updated.

EXTRACT requests:
- SVC Number is loaded into Register Zero.

EXIT NORMAL: Return to caller

EXIT ERROR: Return to caller

ENTRY POINT: IEAVESTE

PURPOSE:
Secondary entry point to extract the entry-point address of an LPA-resident routine

LINKAGE: BASSM with register 1 pointing to a parameter list

CALLERS:
IEAVNPS5
IEAVNPST

INPUT: - A Parameter area with the address passed in Register 1.

OUTPUT:
- Register 0 is loaded with the entry point address of the LPA routine.

EXIT NORMAL: Return to caller

EXIT ERROR: Return to caller

ENTRY POINT: IEAVESTR

PURPOSE:
IEAVESTR is a Functional Recovery Routine (FRR) which provides recovery from errors incurred while:
1. Attempting to update the SVC Table and the SVC Update Recording Table.

IEAVESTU - MODULE DESCRIPTION (Continued)

or

2. Attempting to GETMAIN storage for a register savearea.

LINKAGE: From Recovery Termination Manager (RTM)

CALLERS: None

INPUT: - See introductory comments to the FRR.

OUTPUT: - SDWA as modified by SETRP macro.

EXIT NORMAL: Return to RTM.

EXTERNAL REFERENCES:

ROUTINES:

IEAVENLU - Nucleus Lookup Routine
IEAVVMSR - LPA Directory Search Routine

CONTROL BLOCKS:

CDE	(Contents Directory Entry)	- (r)
CVT	(Communication Vector Table)	- (r)
FRRS	(FRR Routine Stack)	- (r,w)
LPDE	(Link Pack Directory Entry)	- (r)
PSA	(Prefix Save Area)	- (r)
SCVT	(Secondary CVT)	- (r)
SDWA	(System Diagnostic Work Area)	- (r,w)
SVC	(SVC Table entry mapping macro)	- (r,w)

TABLES:

SVC Table (IEAVSVCT)	- (r,w)
SVC Update Recording Table (IEAVSVCR)	- (w)

SERIALIZATION:

CDS Serializes updates to SVC Table
The CPU Lock is held while updating the SVC Table and the SVC Update Recording Table.

SUP-39. Dynamic SVC Table Entry Installer (IEAVESTU)

IEAVESTU - MODULE OPERATION

Upon entry, the routine validity checks the caller's parameter list. If the parameter list is invalid the module returns with a return code to indicate that the service request has failed.

If the function specified is "DELETE" or "REPLACE", data in the parameter list is converted into an SVC Table Entry.

If the function specified is "DELETE" then the entry point and attributes will be set for IGCERROR.

If the function specified is "REPLACE" with "EPNAME" determine the entry point address as follows:

> For Type 1, 2, or 6 (nucleus-resident) routines, IEAVESTU calls the Nucleus Lookup Routine (IEAVENLU) to obtain the entry point address. To invoke IEAVENLU, the CPU lock will be obtained, IEAVESTR will be established as an FRR and IEAVESTU will branch enter GETMAIN to obtain storage for a register savearea. The address of the caller's savearea, IEAVESTU's savearea and other update information, including the current contents of the SVC Table Entry will be saved in the FRR parameter area. When control is returned from IEAVENLU, the register savearea is freed, the FRR is deleted and the CPU lock is released.
>
> For Type 3 and 4 SVCs, IEAVESTU searchs active LPA CDEs and, if necessary, calls the Contents Supervisor LPA Directory Search Routine CSVSRCH (entry point, IEAVVMSR), to obtain the entry point address.
>
> If the EPA cannot be obtained, IEAVESTU will return to the caller with a return code to indicate that the service request has failed.

Once the entry point address for a valid "REPLACE" request is obtained processing proceeds as follows:

> Build a new SVC Table entry in registers.
>
> Set the assist bit if the SVC:
>
> > is not an ESR, and
> > is not APF Authorized, and
> > is not Non-Preemptive.
> >
> > And if the SVC:
> >
> > > is a TYPE 2, 3, or 4, or
> > > any TYPE SVC requiring no more than the Local Lock.
>
> Obtain the CPU Lock.
>
> Establish IEAVESTR as a recovery routine.
>
> Save the address of the caller's savearea and other update information, including the current contents of the SVC Table entry, in the FRR parameter area.
>
> Invoke the DATOFF Compare Double & Swap Routine (IEAVCDS) via DATOFF index, INDCDS to update the SVC Table entry. Update the corresponding entry in the SVC Update Recording Table.
>
> Delete the FRR.

IEAVESTU - MODULE OPERATION (Continued)

Release the CPU Lock.

If the function specified is "EXTRACT", processing proceeds as follows:

If the "EPNAMF" was specified IEAVESTU obtains the entry point address by calling the Nucleus Lookup Routine (IEAVENLU). To invoke IEAVENLU, the CPU lock will be obtained, IEAVESTR will be established as an FRR and IEAVESTU will branch enter GETMAIN to obtain storage for for a register savearea. The address of the caller's savearea, IEAVESTU's savearea and other update information, including the current contents of the SVC Table Entry will be saved in the FRR paramater area. If the routine is not found in the Nucleus, IEAVESTU searches active LPA CDESs. When control is returned from IEAVENLU, the register savearea is freed, the FRR is deleted and the CPU lock is released. If it is not found on the queue of CDEs, IEAVESTU calls the Contents Supervisor LPA Directory Search Routine, (IEAVVMSR). When the entry point address is obtained from one of these searches, IEAVESTU scans the SVC Table for an entry with a matching entry point address. If it is found the SVC number will be returned in register zero. If the "EXTRACT" fails, IEAVESTU will return with a non-zero return code.

When this module is called at entry point, IEAVESTE, the logic to search the LPA is executed as a separate service for SVC Table Initialization routines.

RECOVERY OPERATION:

The recovery environment is established whenever IEAVESTU obtains the CPU lock. This lock is obtained under two

SUP-39. Dynamic SVC Table Entry Installer (IEAVESTU)

IEAVESTU - DIAGNOSTIC AIDS

ENTRY POINT NAMES: IEAVESTU
IEAVESTE
IEAVESTR

MESSAGES: None

ABEND CODES: None

WAIT STATE CODES: None

RETURN CODES:

ENTRY POINT IEAVESTU:

EXIT NORMAL:

0

EXIT ERROR:

Decimal Code	Hexadecimal Code	Reason
04	04	Parameter List Version Level is invalid. SVC Number is not supplied for DELETE or REPLACE. Version Level 1 on parameter list for EXTRACT.
08	08	Extra parameters or RESERVED bits not zero for DELETE request
12	0C	SVC Type not equal to 1,2,3,4,5, or 6 for REPLACE request. Extra bits set on in ATTRIBUTE field or RESERVED byte not zero for REPLACE request. Extra bits set on in LOCK field for REPLACE request. SVC ENTRY POINT not halfword aligned for REPLACE request.
16	10	ENTRY POINT passed for a TYPE 5 REPLACE request. LOCK requested for a TYPE 6 REPLACE request. Global LOCK requested for a TYPE 3/4 REPLACE request. Neither ENTRY POINT nor EPNAME passed for a REPLACE request that is not TYPE 5. Both ENTRY POINT and EPNAME are passed for a REPLACE request. ENTRY POINT passed is zero for a REPLACE request. CMS LOCK requested without LOCAL LOCK for a REPLACE request.
20	14	Function is not recognized.
24	18	Attempt to Update ESR Entry.
28	1C	Unable to locate Entry Point Address

IEAVESTU - DIAGNOSTIC AIDS (Continued)

		for an EPNAME specification.
32	20	Neither ENTRY POINT nor EPNAME passed for EXTRACT request. Both ENTRY POINT and EPNAME passed for EXTRACT request. ENTRY POINT passed is not on halfword boundary for EXTRACT request. ENTRY POINT passed is zero for EXTRACT request. SVC number passed for EXTRACT request Extra parameters supplied or RESERVED bits not zero for EXTRACT request.
36	24	Unable to locate SVC Routine in the SVC Table for an "EXTRACT" request.
40	28	Program check suffered while attempting to update SVC Table.

ENTRY POINT IEAVESTE:

EXIT NORMAL:

0 = Successful

EXIT ERROR:

Decimal Code	Hexadecimal Code	Reason
04	04	Missing Required parameters.
28	1C	Unable to locate Entry Point Address for an EPNAME specification.

ENTRY POINT IEAVESTR: None

REGISTER CONTENTS ON ENTRY:

ENTRY POINT IEAVESTU:

1	Address of SVCUPDTE parameter list
13	Address of 18 word save area
14	Return Address
15	Entry Point Address

ENTRY POINT IEAVESTE:

1	Address of SVCUPDTE parameter list
13	Address of 18 word save area
14	Return Address
15	Entry Point Address

ENTRY POINT IEAVESTR: Irrelevant

REGISTER CONTENTS ON EXIT:

ENTRY POINT IEAVESTU:

EXIT NORMAL:

0	Unchanged for "REPLACE/DELETE" request SVC number for "EXTRACT" request
1-13	Unchanged
14	Return Address

IEAVESTU - DIAGNOSTIC AIDS (Continued)

15	Zero

ENTRY POINT IEAVESTE:

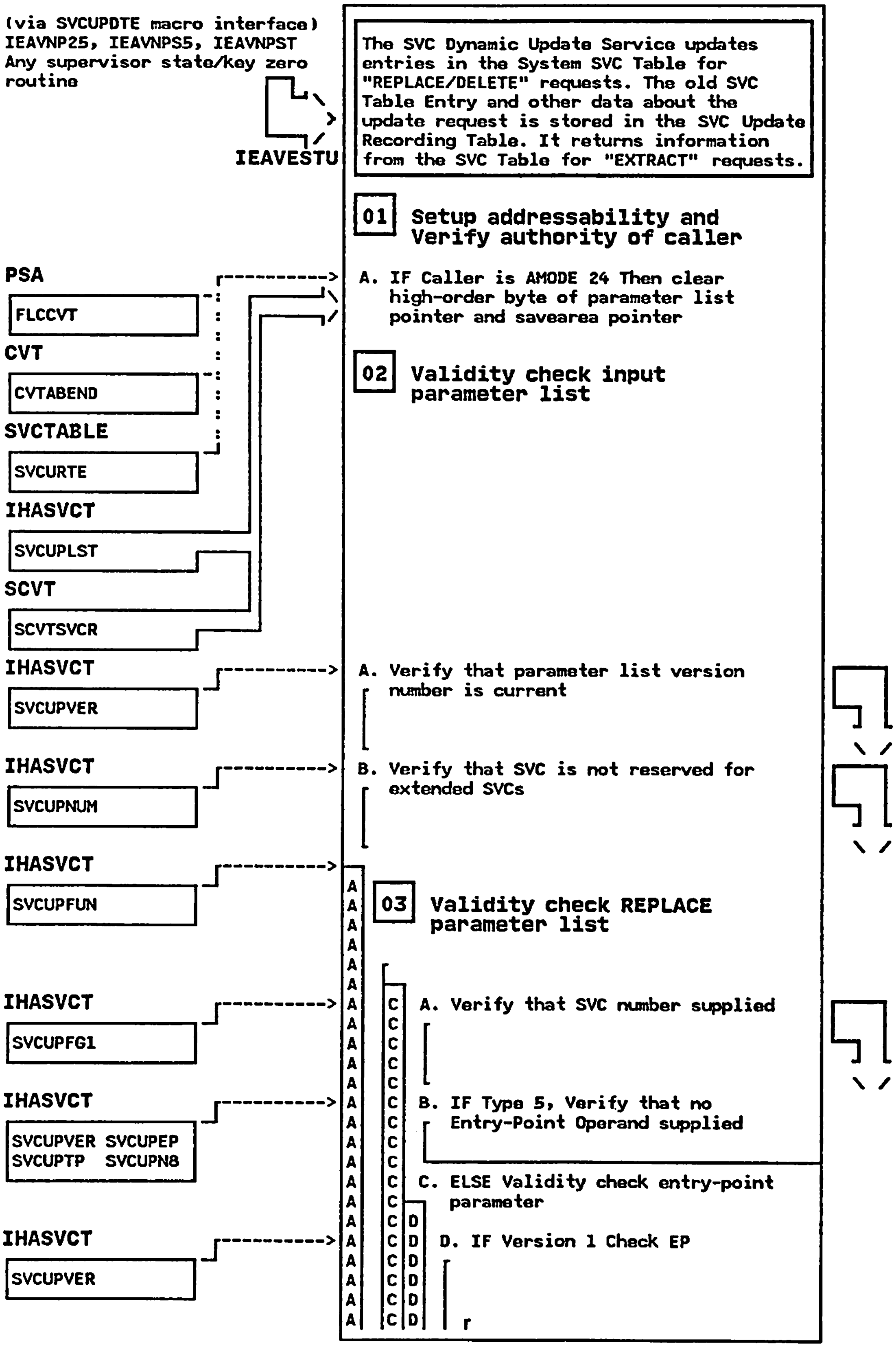
EXIT NORMAL:

0	Entry point address
1-13	Unchanged
14	Return Address
15	Zero

ENTRY POINT IEAVESTR: Irrelevant

SUP-39. Dynamic SVC Table Entry Installer (IEAVESTU)

IEAVESTU - DYNAMIC SVC TABLE UPDATE SERVICE STEP 01



(via SVCUPDTE macro interface)
IEAVNP25, IEAVNPS5, IEAVNPST
Any supervisor state/key zero
routine

IEAVESTU

The SVC Dynamic Update Service updates entries in the System SVC Table for "REPLACE/DELETE" requests. The old SVC Table Entry and other data about the update request is stored in the SVC Update Recording Table. It returns information from the SVC Table for "EXTRACT" requests.

PSA
FLCCVT
CVT
CVTABEND
SVCTABLE
SVCURTE
IHASVCT
SVCUPLST
SCVT
SCVTSVCR

01 Setup addressability and Verify authority of caller

A. IF Caller is AMODE 24 Then clear high-order byte of parameter list pointer and savearea pointer

02 Validity check input parameter list

IHASVCT
SVCUPVER

A. Verify that parameter list version number is current

IHASVCT
SVCUPNUM

B. Verify that SVC is not reserved for extended SVCs

IHASVCT
SVCUPFUN

03 Validity check REPLACE parameter list

IHASVCT
SVCUPFG1

A. Verify that SVC number supplied

IHASVCT
SVCUPVER SVCUPEP
SVCUPTP SVCUPN8

B. IF Type 5, Verify that no Entry-Point Operand supplied

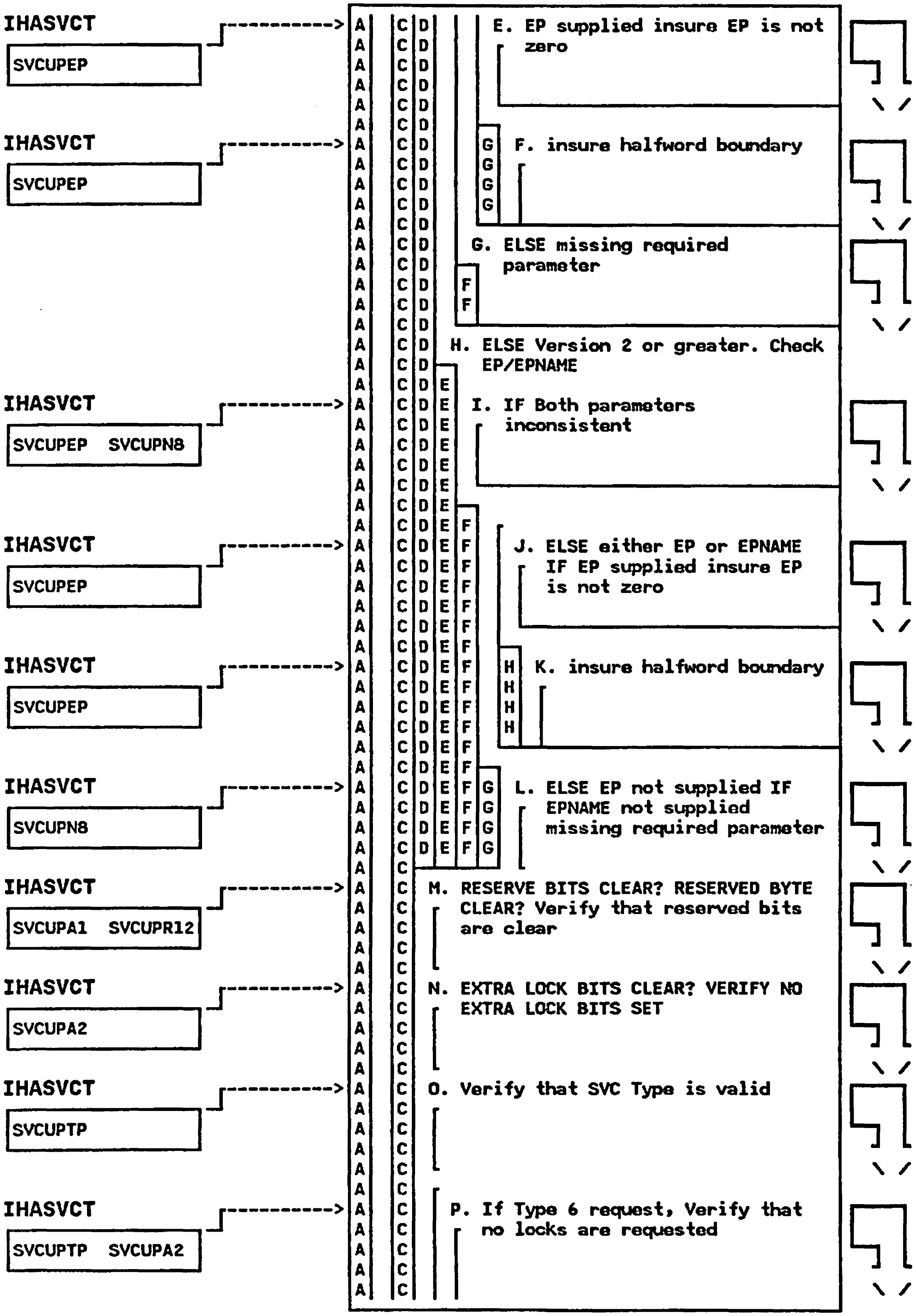
C. ELSE Validity check entry-point parameter

IHASVCT
SVCUPVER

D. IF Version 1 Check EP

SUP-39. Dynamic SVC Table Entry Installer (IEAVESTU)

IEAVESTU - DYNAMIC SVC TABLE UPDATE SERVICE STEP 03E



IHASVCT SVCUPEP

E. EP supplied insure EP is not zero

IHASVCT SVCUPEP

F. insure halfword boundary

G. ELSE missing required parameter

H. ELSE Version 2 or greater. Check EP/EPNAME

IHASVCT SVCUPEP SVCUPN8

I. IF Both parameters inconsistent

IHASVCT SVCUPEP

J. ELSE either EP or EPNAME IF EP supplied insure EP is not zero

IHASVCT SVCUPEP

K. insure halfword boundary

IHASVCT SVCUPN8

L. ELSE EP not supplied IF EPNAME not supplied missing required parameter

IHASVCT SVCUPA1 SVCUPR12

M. RESERVE BITS CLEAR? RESERVED BYTE CLEAR? Verify that reserved bits are clear

IHASVCT SVCUPA2

N. EXTRA LOCK BITS CLEAR? VERIFY NO EXTRA LOCK BITS SET

IHASVCT SVCUPTP

O. Verify that SVC Type is valid

IHASVCT SVCUPTP SVCUPA2

P. If Type 6 request, Verify that no locks are requested

IEAVESTU - DYNAMIC SVC TABLE UPDATE SERVICE STEP 04

IHASVCT: SVCUPTP SVCUPA2 SVCUPLL SVCUPCMS

04 Validity check DELETE parameter list

IHASVCT: SVCUPFG1

A. Verify that SVC number supplied

IHASVCT: SVCUPEP SVCUPTP SVCUPA1 SVCUPA2 SVCUPR12

B. "EP" NOT FILLED IN? SVC TYPE NULL? ATTRIBUTE FIELD NULL? LOCK FIELD NULL? RESERVED FIELD NULL? Verify that the rest of the parm list is clear

IHASVCT: SVCUPVER SVCUPN8

05 Validity check EXTRACT parameter list

IHASVCT: SVCUPVER

IHASVCT: SVCUPEP SVCUPN8

A. IF Both parameters inconsistent

IHASVCT: SVCUPEP

B. ELSE either EP or EPNAME IF EP supplied insure EP is not zero

IHASVCT: SVCUPEP

C. insure halfword boundary

IHASVCT: SVCUPN8

D. ELSE EP not supplied IF EPNAME not supplied missing required parameter

IHASVCT: SVCUPFG1

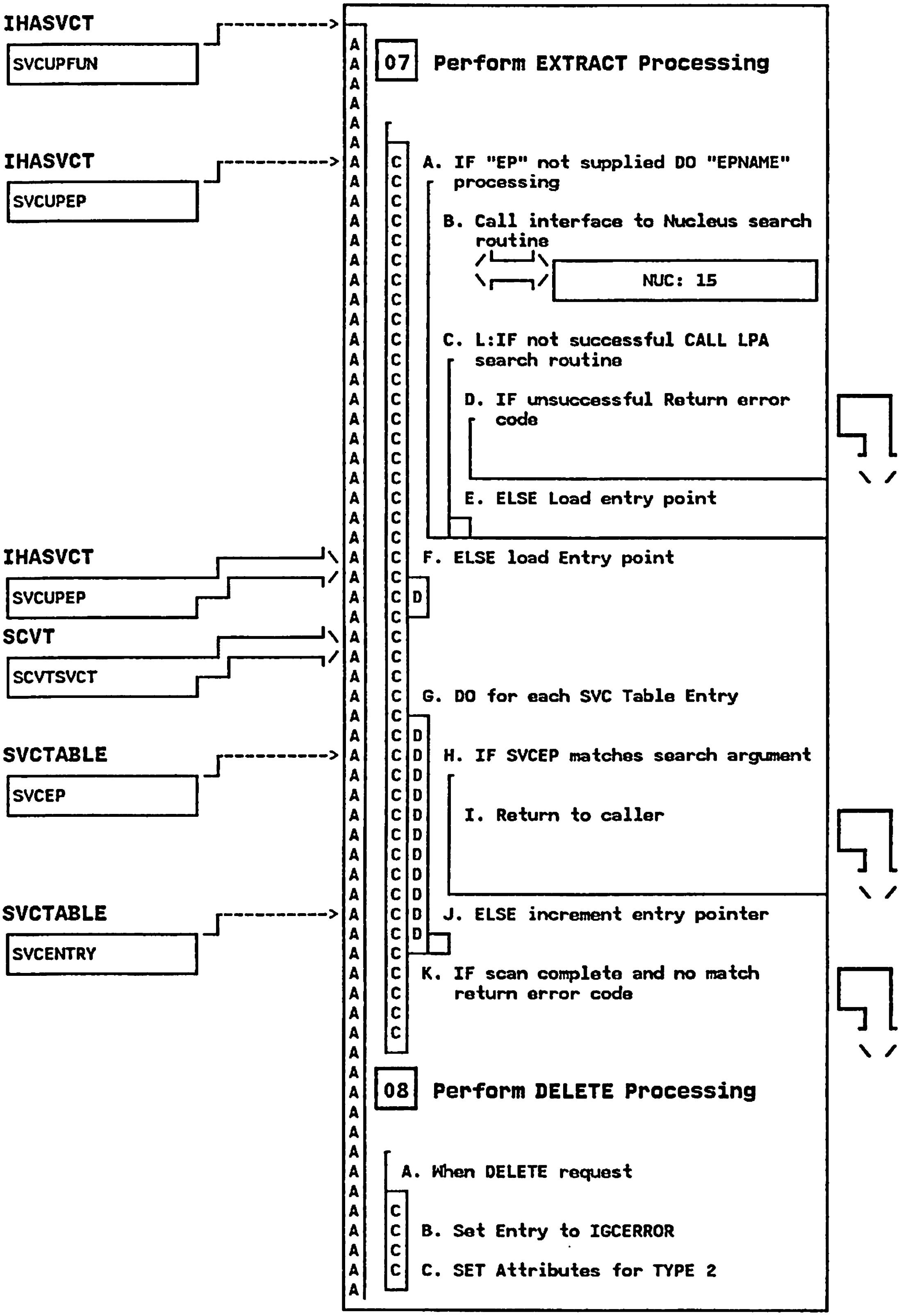
E. Verify that no SVC number is supplied

IHASVCT: SVCUPTP SVCUPA1 SVCUPA2 SVCUPR12

F. SVC TYPE NULL? ATTRIBUTE FIELD NULL? LOCK FIELD NULL? RESERVED FIELD NULL? Verify that extra bits are clear

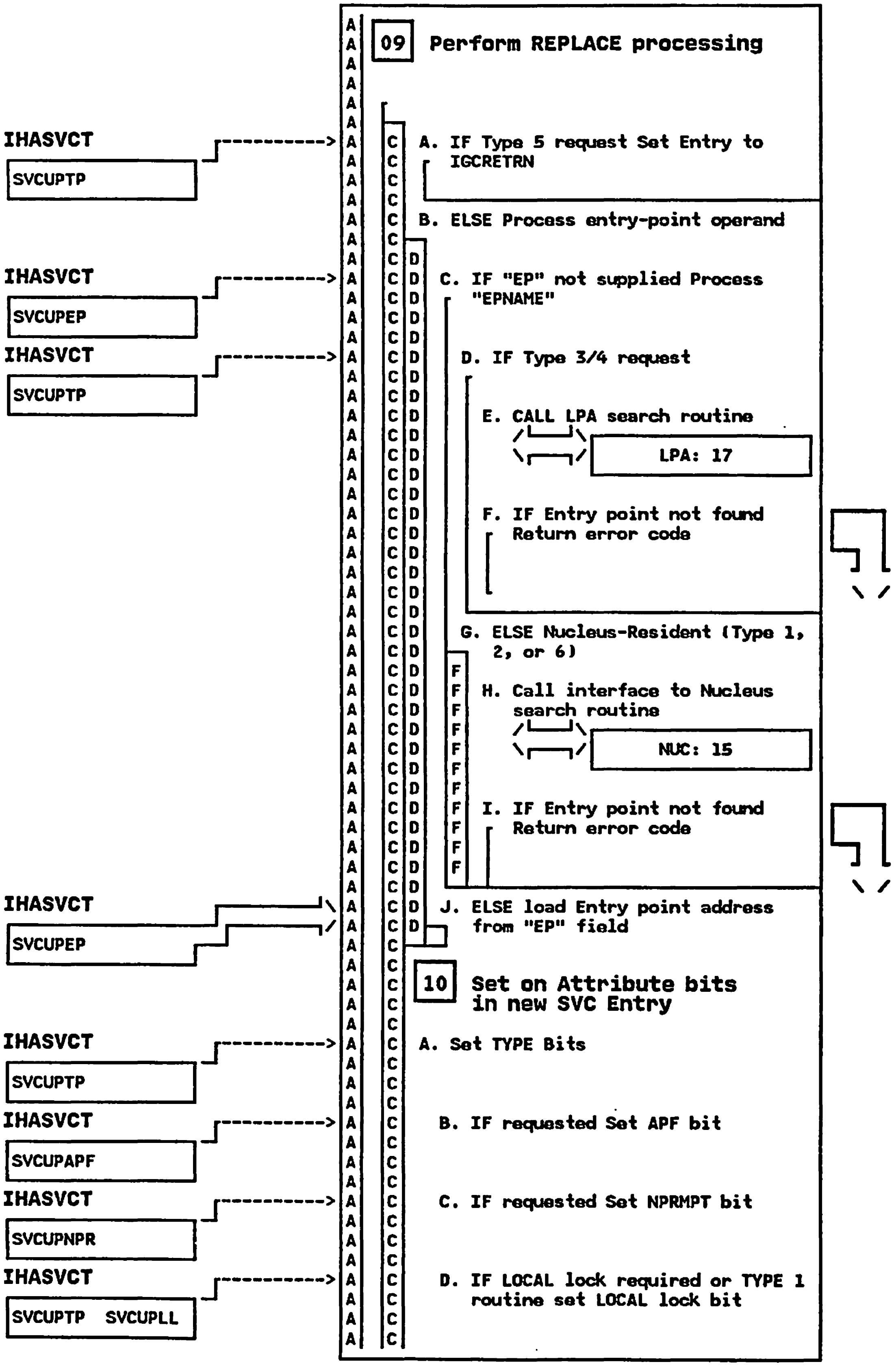
06 OTHERWISE Return code for invalid funtion

IEAVESTU - DYNAMIC SVC TABLE UPDATE SERVICE   STEP 07



```
07 Perform EXTRACT Processing

   A. IF "EP" not supplied DO "EPNAME"
      processing
      B. Call interface to Nucleus search
         routine
      C. L:IF not successful CALL LPA
         search routine
         D. IF unsuccessful Return error
            code
         E. ELSE Load entry point
   F. ELSE load Entry point
   G. DO for each SVC Table Entry
      H. IF SVCEP matches search argument
         I. Return to caller
      J. ELSE increment entry pointer
   K. IF scan complete and no match
      return error code

08 Perform DELETE Processing

   A. When DELETE request
   B. Set Entry to IGCERROR
   C. SET Attributes for TYPE 2
```

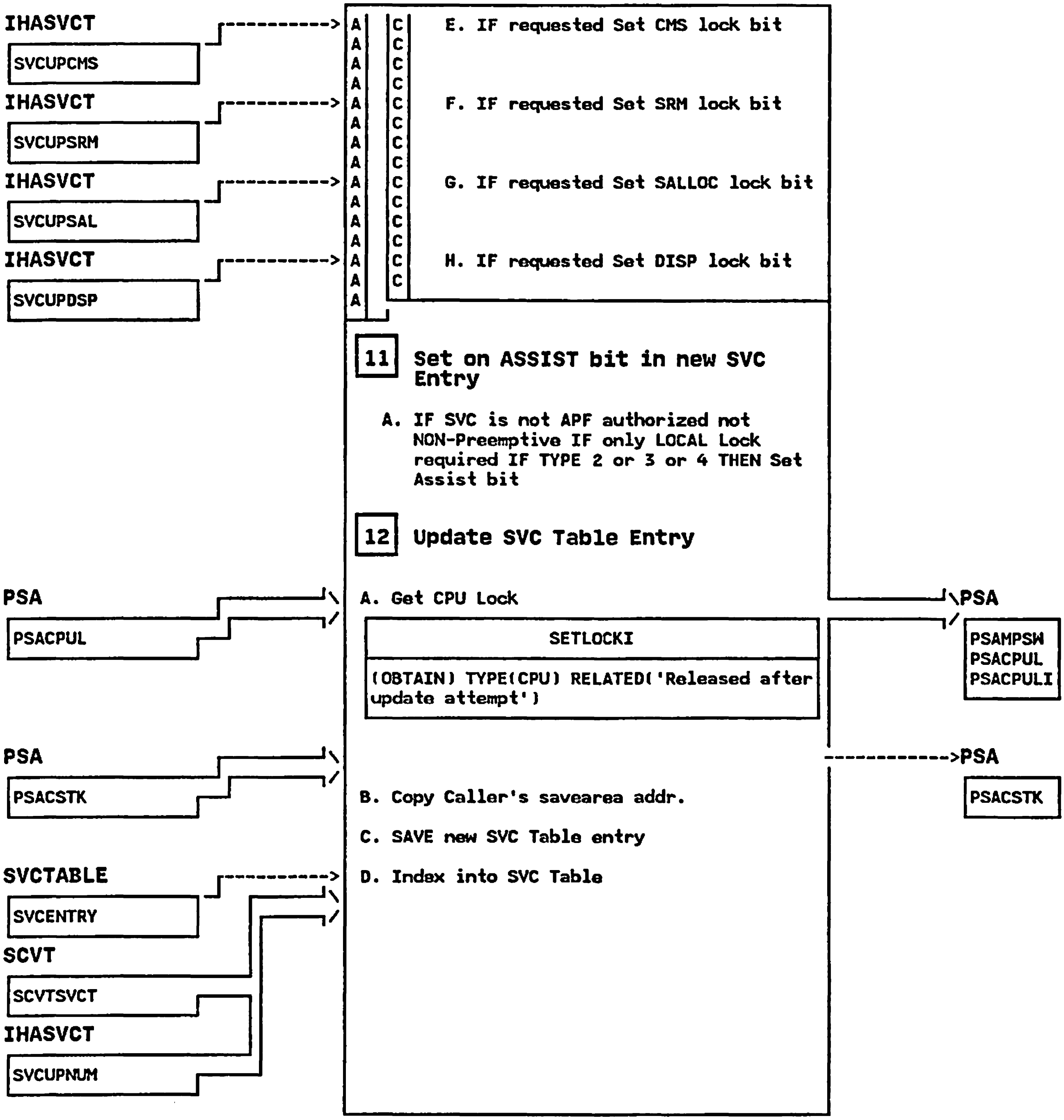
IEAVESTU - DYNAMIC SVC TABLE UPDATE SERVICE STEP 09



```
09 Perform REPLACE processing

   A. IF Type 5 request Set Entry to
      IGCRETRN
   B. ELSE Process entry-point operand
      C. IF "EP" not supplied Process
         "EPNAME"
         D. IF Type 3/4 request
            E. CALL LPA search routine
            F. IF Entry point not found
               Return error code
         G. ELSE Nucleus-Resident (Type 1,
            2, or 6)
            H. Call interface to Nucleus
               search routine
            I. IF Entry point not found
               Return error code
      J. ELSE load Entry point address
         from "EP" field

10 Set on Attribute bits
   in new SVC Entry

   A. Set TYPE Bits
      B. IF requested Set APF bit
      C. IF requested Set NPRMPT bit
      D. IF LOCAL lock required or TYPE 1
         routine set LOCAL lock bit
```

SUP-39. Dynamic SVC Table Entry Installer (IEAVESTU)

IEAVESTU - DYNAMIC SVC TABLE UPDATE SERVICE STEP 10E



E. IF requested Set CMS lock bit

F. IF requested Set SRM lock bit

G. IF requested Set SALLOC lock bit

H. IF requested Set DISP lock bit

11 Set on ASSIST bit in new SVC Entry

A. IF SVC is not APF authorized not NON-Preemptive IF only LOCAL Lock required IF TYPE 2 or 3 or 4 THEN Set Assist bit

12 Update SVC Table Entry

A. Get CPU Lock

SETLOCKI
(OBTAIN) TYPE(CPU) RELATED('Released after update attempt')

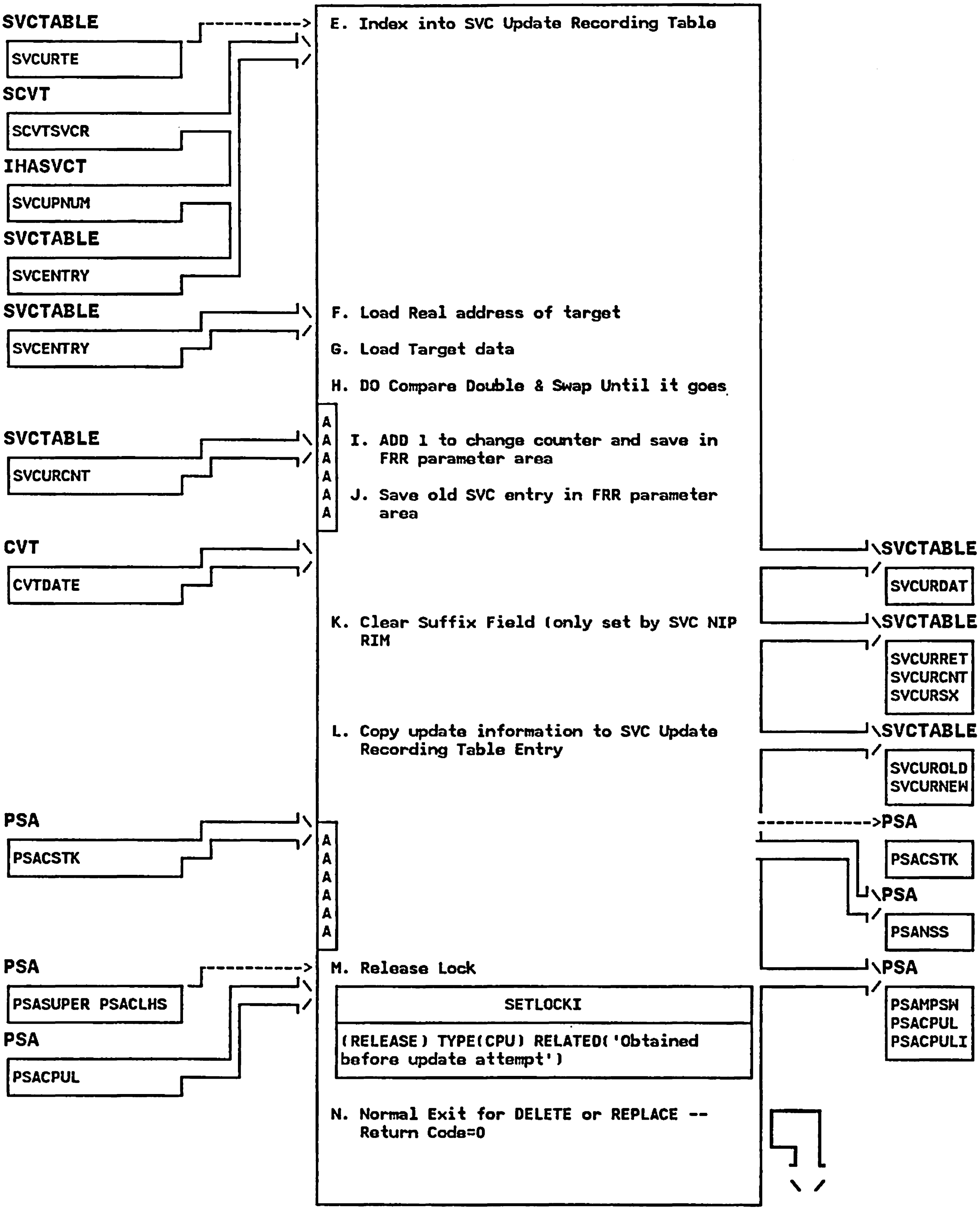
B. Copy Caller's savearea addr.

C. SAVE new SVC Table entry

D. Index into SVC Table

SUP-39. Dynamic SVC Table Entry Installer (IEAVESTU)

IEAVESTU - DYNAMIC SVC TABLE UPDATE SERVICE STEP 12E



E. Index into SVC Update Recording Table

F. Load Real address of target

G. Load Target data

H. DO Compare Double & Swap Until it goes

I. ADD 1 to change counter and save in FRR parameter area

J. Save old SVC entry in FRR parameter area

K. Clear Suffix Field (only set by SVC NIP RIM

L. Copy update information to SVC Update Recording Table Entry

M. Release Lock

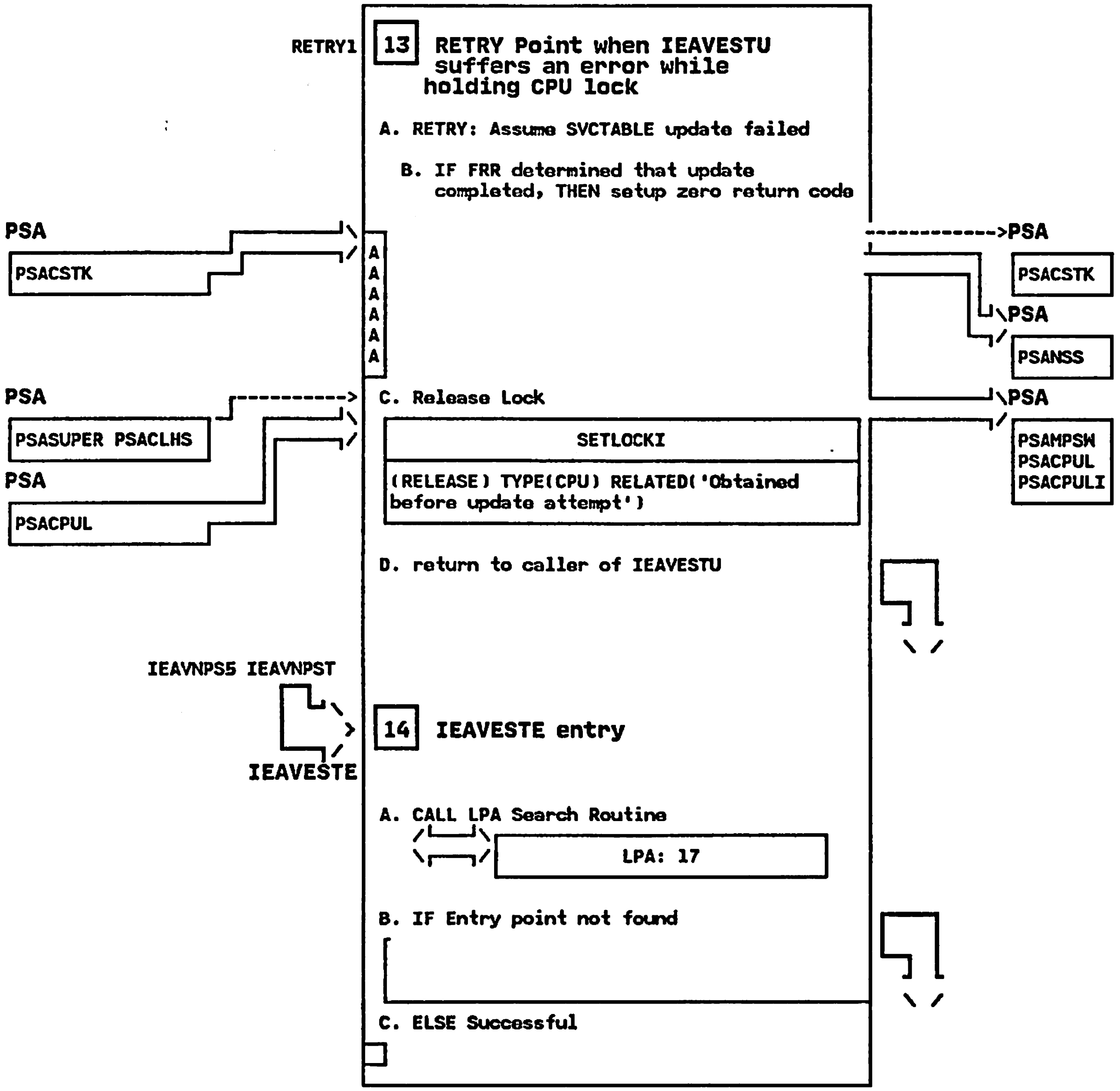
SETLOCKI

(RELEASE) TYPE(CPU) RELATED('Obtained before update attempt')

N. Normal Exit for DELETE or REPLACE -- Return Code=0

SUP-39. Dynamic SVC Table Entry Installer (IEAVESTU)

IEAVESTU - DYNAMIC SVC TABLE UPDATE SERVICE STEP 13



13 RETRY Point when IEAVESTU suffers an error while holding CPU lock

A. RETRY: Assume SVCTABLE update failed

B. IF FRR determined that update completed, THEN setup zero return code

C. Release Lock

SETLOCKI

(RELEASE) TYPE(CPU) RELATED('Obtained before update attempt')

D. return to caller of IEAVESTU

14 IEAVESTE entry

A. CALL LPA Search Routine

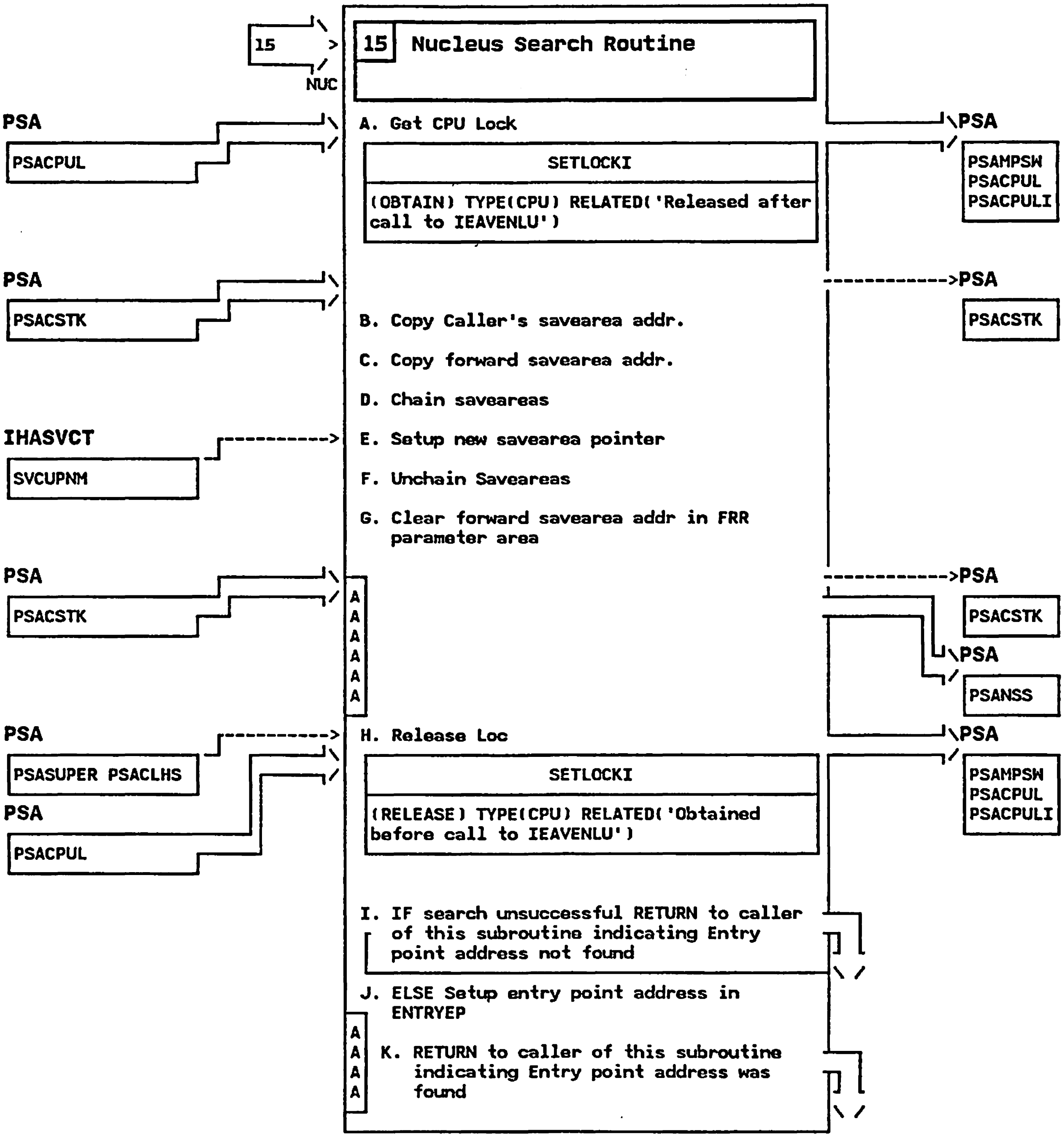
LPA: 17

B. IF Entry point not found

C. ELSE Successful

SUP-39. Dynamic SVC Table Entry Installer (IEAVESTU)

IEAVESTU - DYNAMIC SVC TABLE UPDATE SERVICE STEP 15



15 Nucleus Search Routine

A. Get CPU Lock

SETLOCKI

(OBTAIN) TYPE(CPU) RELATED('Released after call to IEAVENLU')

B. Copy Caller's savearea addr.

C. Copy forward savearea addr.

D. Chain saveareas

E. Setup new savearea pointer

F. Unchain Saveareas

G. Clear forward savearea addr in FRR parameter area

H. Release Loc

SETLOCKI

(RELEASE) TYPE(CPU) RELATED('Obtained before call to IEAVENLU')

I. IF search unsuccessful RETURN to caller of this subroutine indicating Entry point address not found

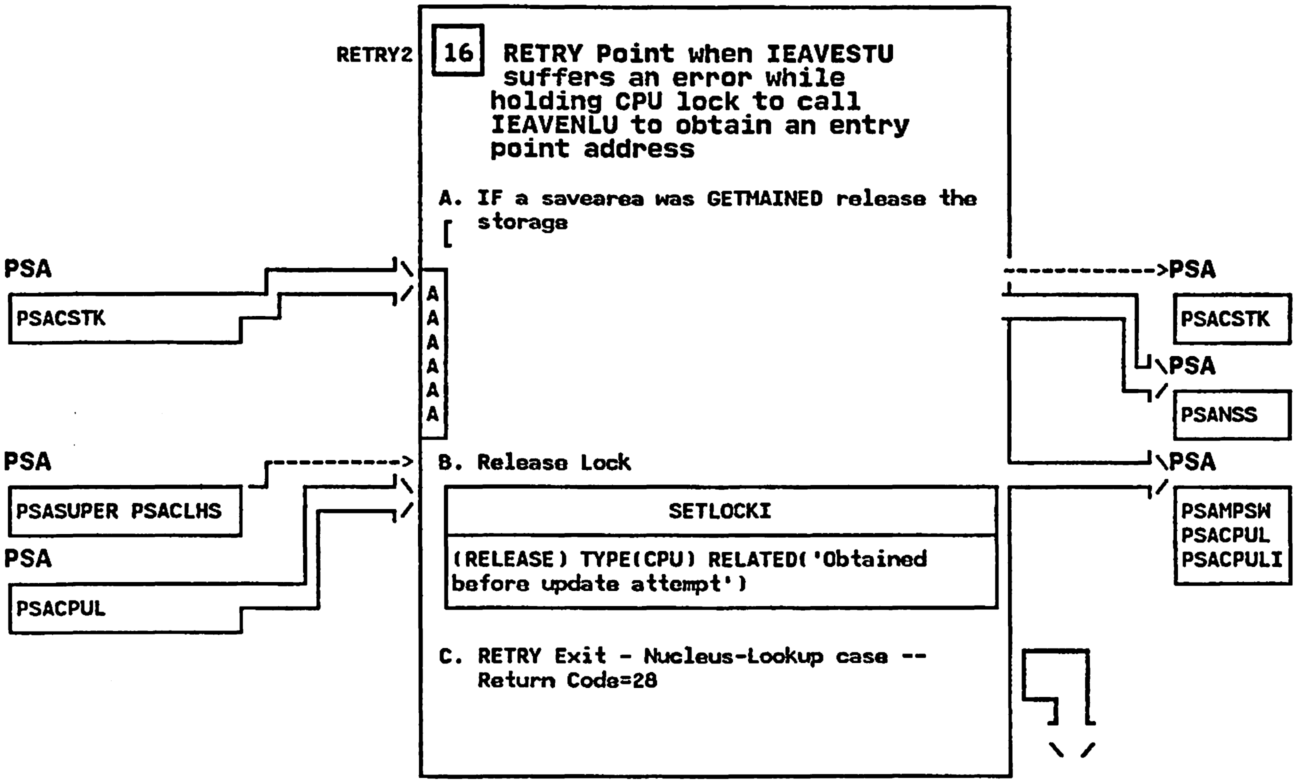
J. ELSE Setup entry point address in ENTRYEP

K. RETURN to caller of this subroutine indicating Entry point address was found

SUP-39. Dynamic SVC Table Entry Installer (IEAVESTU)

IEAVESTU - DYNAMIC SVC TABLE UPDATE SERVICE

STEP 16



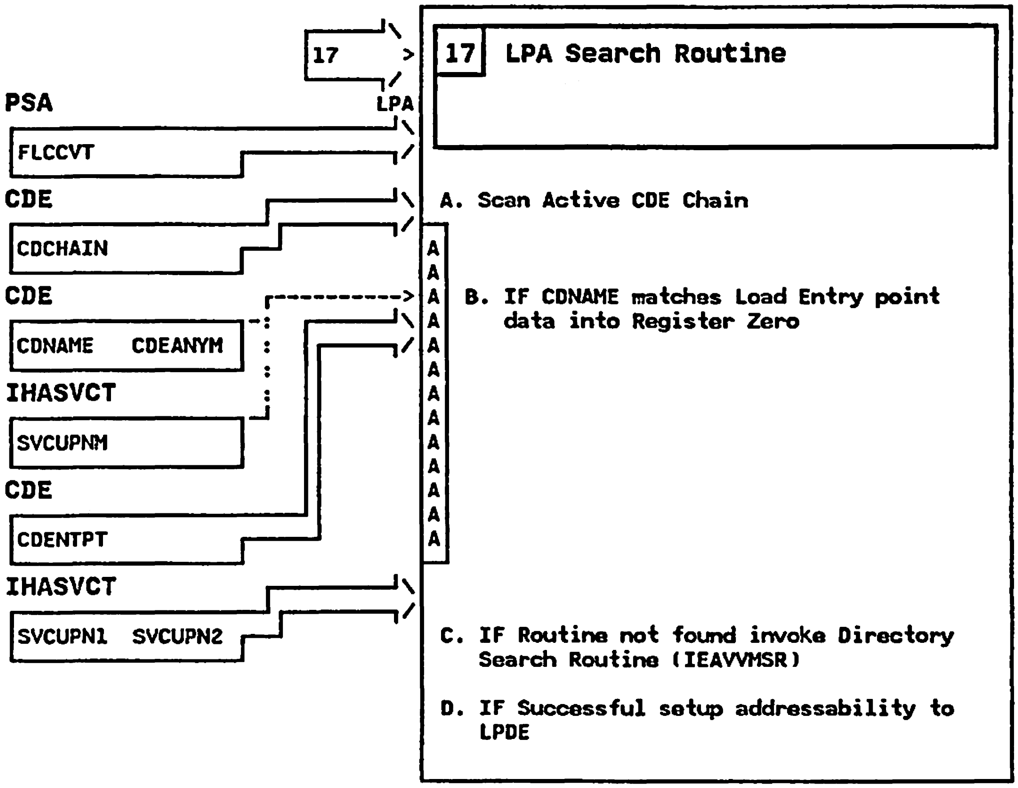
16 RETRY Point when IEAVESTU suffers an error while holding CPU lock to call IEAVENLU to obtain an entry point address

A. IF a savearea was GETMAINED release the storage

B. Release Lock

SETLOCKI
(RELEASE) TYPE(CPU) RELATED('Obtained before update attempt')

C. RETRY Exit - Nucleus-Lookup case -- Return Code=28



17 LPA Search Routine

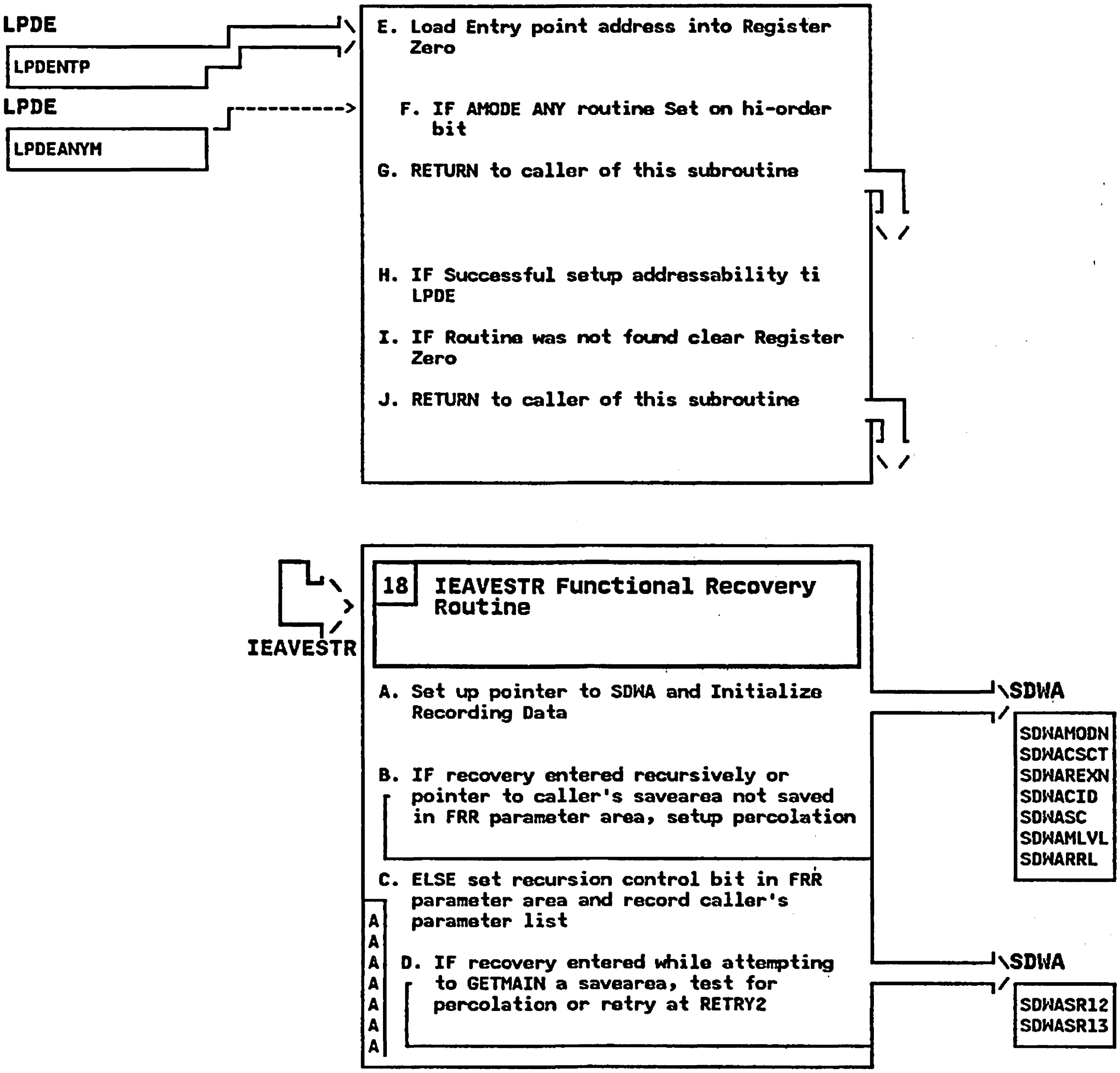
A. Scan Active CDE Chain

B. IF CDNAME matches Load Entry point data into Register Zero

C. IF Routine not found invoke Directory Search Routine (IEAVVMSR)

D. IF Successful setup addressability to LPDE

IEAVESTU - DYNAMIC SVC TABLE UPDATE SERVICE STEP 17E



E. Load Entry point address into Register Zero

F. IF AMODE ANY routine Set on hi-order bit

G. RETURN to caller of this subroutine

H. IF Successful setup addressability ti LPDE

I. IF Routine was not found clear Register Zero

J. RETURN to caller of this subroutine

18 IEAVESTR Functional Recovery Routine

A. Set up pointer to SDWA and Initialize Recording Data

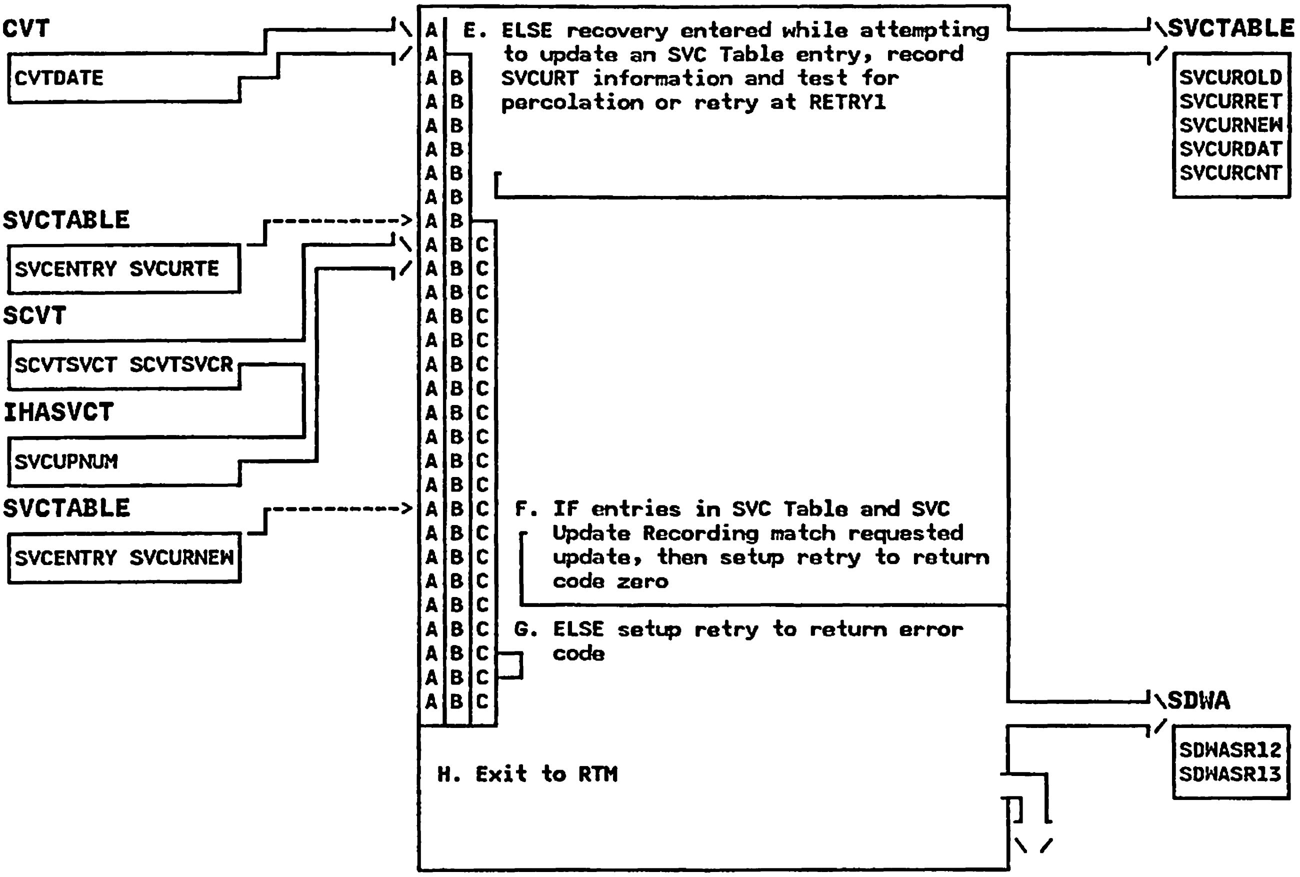
B. IF recovery entered recursively or pointer to caller's savearea not saved in FRR parameter area, setup percolation

C. ELSE set recursion control bit in FRR parameter area and record caller's parameter list

D. IF recovery entered while attempting to GETMAIN a savearea, test for percolation or retry at RETRY2

SUP-39. Dynamic SVC Table Entry Installer (IEAVESTU)

IEAVESTU - DYNAMIC SVC TABLE UPDATE SERVICE STEP 18E



E. ELSE recovery entered while attempting to update an SVC Table entry, record SVCURT information and test for percolation or retry at RETRY1

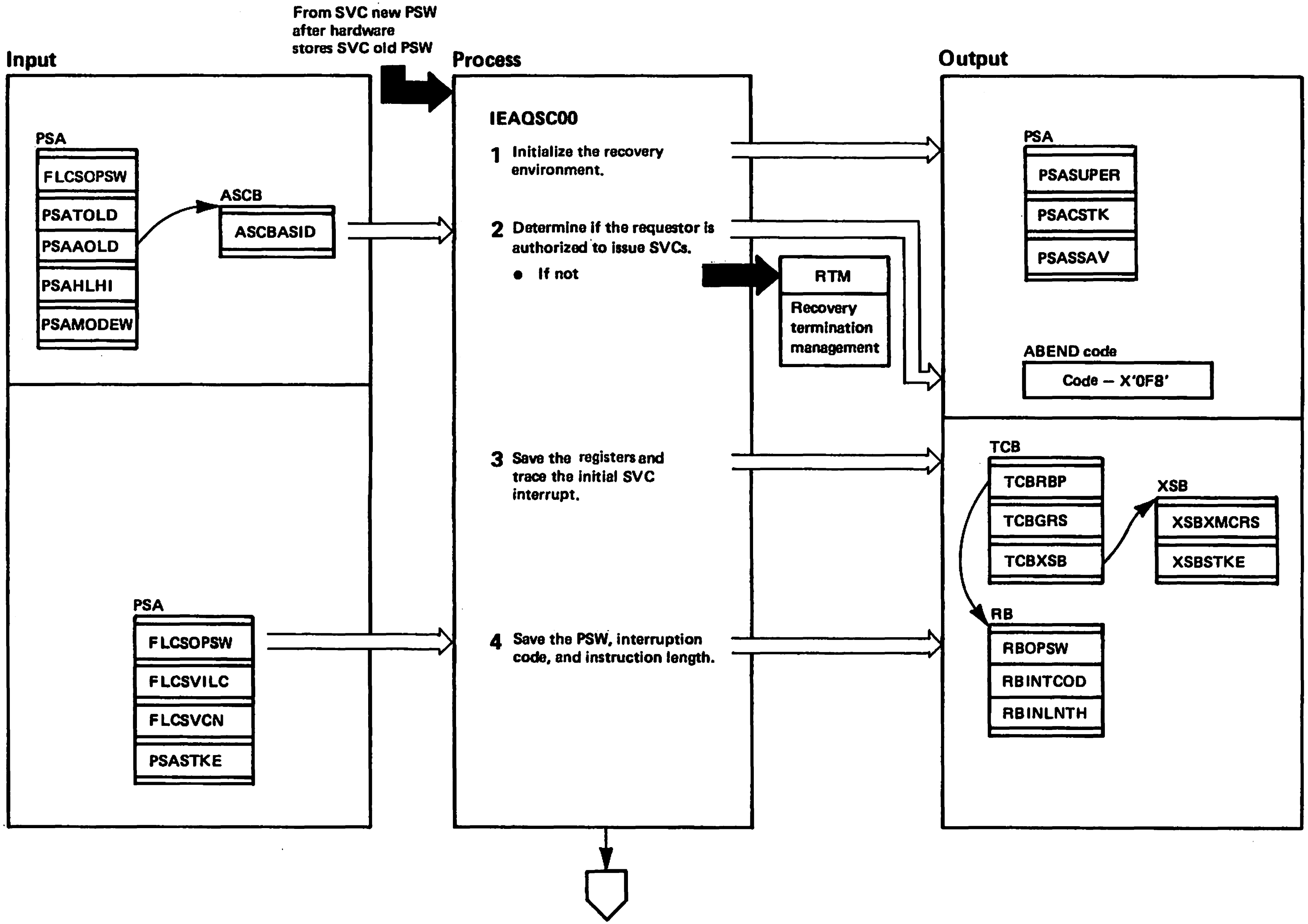
F. IF entries in SVC Table and SVC Update Recording match requested update, then setup retry to return code zero

G. ELSE setup retry to return error code

H. Exit to RTM

This page left blank.

Diagram SUP-40. SVC Interrupt Handler (IEAVESVC) (Part 1 of 14)



1 Initialize the recovery environment.

2 Determine if the requestor is authorized to issue SVCs.

- If not

3 Save the registers and trace the initial SVC interrupt.

4 Save the PSW, interruption code, and instruction length.

Diagram SUP-40. SVC Interrupt Handler (IEAVESVC) (Part 2 of 14)

Extended Description	Module	Label
The SVC interrupt handler sets up the proper operating environment for a requested SVC (supervisor call) by obtaining any necessary locks and initializing registers. The SVC interrupt handler routes control to the appropriate SVC routine after setting up the operator's environment.	IEAVESVC	IEAQSC00

1 The SVC first level interrupt handler (FLIH) copies the SVC code to PSALSVCI (this field is used as a check against recursive ABEND loops), saves the current FRR stack pointer (PSACSTK) in the PSASSAV field, makes the super FRR stack current, and sets the SVC super bit (PSASVC) to 1.

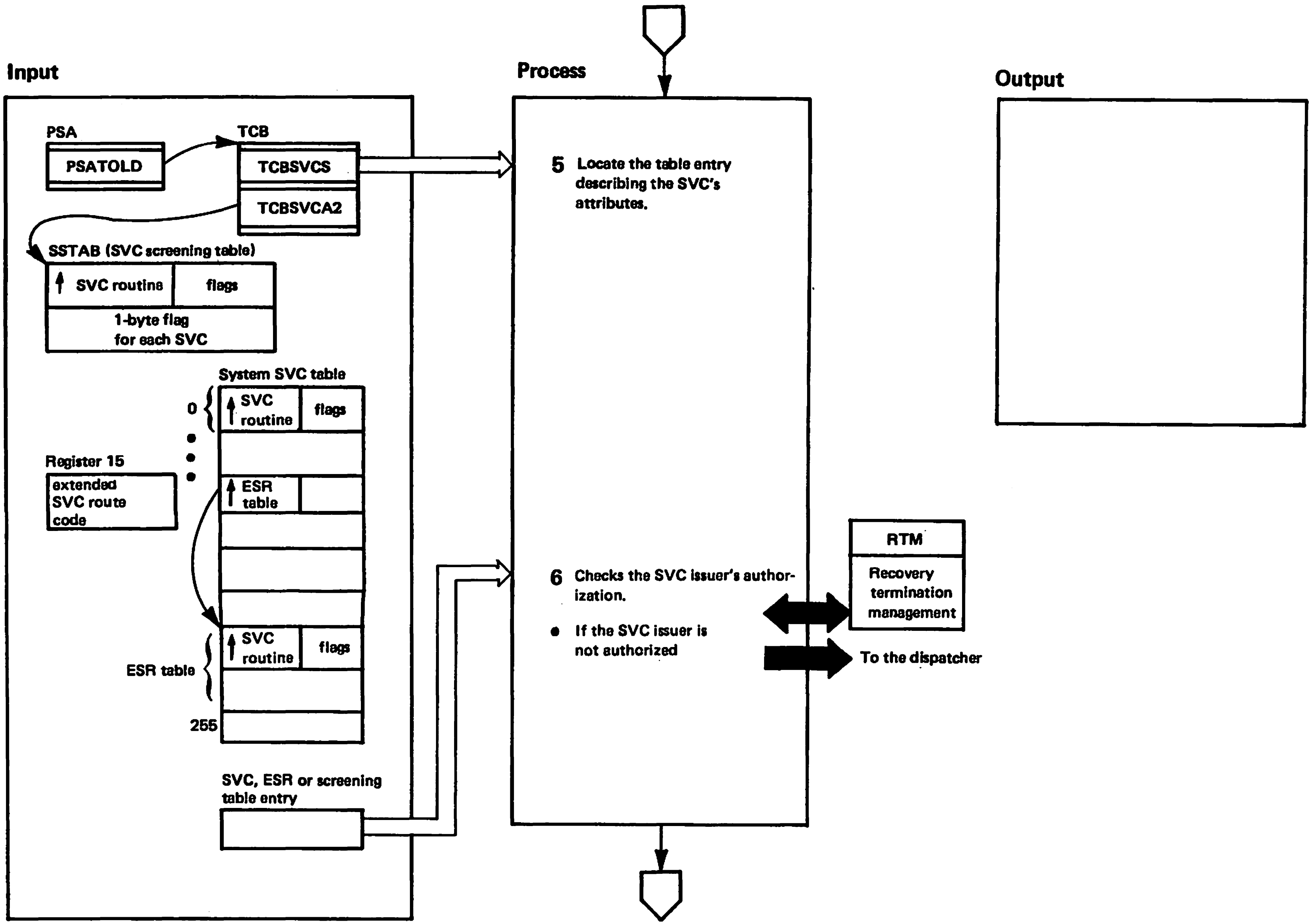
2 Programs in the disabled state, in cross memory mode, in enabled unlocked task mode with an FRR, in non-task mode, or holding locks cannot issue SVCs. If one does, the SVC FLIH clears PSALSVCI,,and passes control toRTM, which then terminates the caller with an ABEND code of X'0F8'.

3 For type 1 and type 6 SVCs the SVC FLIH saves the SVC issuer's registers in the TCBGRS field; for type 2, 3 and 4 SVCs, it saves the registers in the SVRB obtained by the SVC FLIH. It traces the SVC interrupt. Additionally, the SVC FLIH uses the generalized trace facility.

4 The SVC FLIH saves:

- The PSW in the RBOPSW field.
- The interrupt code in the RBINTCOD field.
- The instruction length in the RBINLNTH field.
- Cross memory control registers 3 and 4 in the XSBXMCRS field.
- The PCLINK stack header in the XSBSTKE field.

Diagram SUP-40. SVC Interrupt Handler (IEAVESVC) (Part 3 of 14)



5 Locate the table entry describing the SVC's attributes.

6 Checks the SVC issuer's authorization.

- If the SVC issuer is not authorized

Diagram SUP-40. SVC Interrupt Handler (IEAVESVC) (Part 4 of 14)

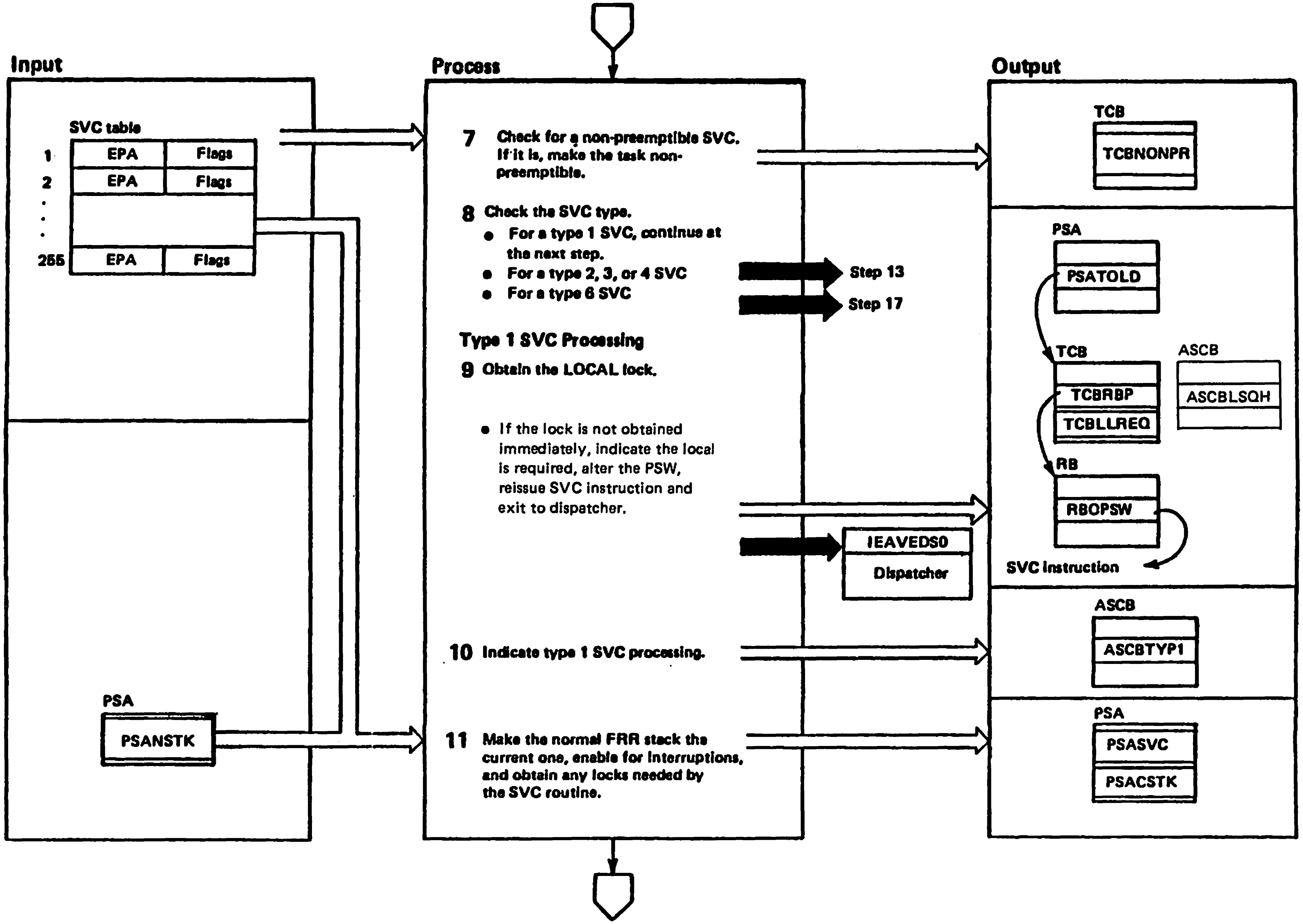
Extended Description **Module** **Label**

5 Each SVC has an entry describing its attributes (entry point of the SVC routine, addressing mode of the SVC routine, type, whether the SVC routine requires APF authorization or locks) in one of three tables: the SVC screening table, the system SVC table, or the extended SVC routine (ESR) table. IEAVESVC locates the entry to user later when establishing the environment required to invoke the service routine. To locate the appropriate entry, IEAVESVC does the following in the order specified.

- If SVC screening is active (TCBSVCS=1), IEAVESVC determines whether the SVC issuer is allowed to issue the SVC (that is, whether the SVC is being screened). To make the determination, IEAVESVC uses the SVC number to index into the task's SVC screening table (pointed to by the TCBSVCA2 field), which contains a flag for every SVC indicating whether the task can issue the SVC. If the SVC cannot be issued, IEAVESVC processes instead the SVC request described in the first doubleword of the screening table. The doubleword contains the information normally obtained from the system SVC table.
- If SVC can be issued, IEAVESVC uses the SVC number to locate the appropriate entry in the system SVC table. Unless the attributes indicate that it is an extended SVC (either SVC 109, 116, 122, or 137) (SVCESR=1), IEAVESVC uses this information when preparing to invoke the service routine.
- If an extended SVC is being processed, the first word of the SVC table entry points to an ESR table. IEAVESVC uses the extended SVC route code in register 15 to locate the appropriate entry in the ESR table.

6 If the table entry information indicates that the SVC can be issued only by users with APF authorization (SVCAPF=d), the SVC FLIH issues a TESTAUTH macro to determine if the user is authorized. If the TESTAUTH return code is nonzero, the user is not authorized. In this case, the SVC FLIH calls RTM to abnormally terminate the user with ABEND code X'047'. It then exits to the dispatcher at entry point IEA0DS1.

Diagram SUP-40. SVC Interrupt Handler (IEAVESVC) (Part 5 of 14)



7 Check for a non-preemptible SVC. If it is, make the task non-preemptible.

8 Check the SVC type.

- For a type 1 SVC, continue at the next step.
- For a type 2, 3, or 4 SVC
- For a type 6 SVC

Type 1 SVC Processing

9 Obtain the LOCAL lock.

- If the lock is not obtained immediately, indicate the local is required, alter the PSW, reissue SVC instruction and exit to dispatcher.

10 Indicate type 1 SVC processing.

11 Make the normal FRR stack the current one, enable for interruptions, and obtain any locks needed by the SVC routine.

Diagram SUP-40: SVC Interrupt Handler (IEAVESVC) (Part 6 of 14)

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

7 If the SVC is non-preemptible (SVCNP=1), the SVC FLIH makes the task non-preemptible (TCBNONPR=1).

8 Based on the SVC type, the SVC FLIH branches to the appropriate processing routine. Note that steps 11 and 12 show processing common to SVC types 1, 2, 3, and 4.

SVC type	*Steps*
1	9-12
2, 3, 4	13, 14, 11, 12 (in that order)
6	17-19

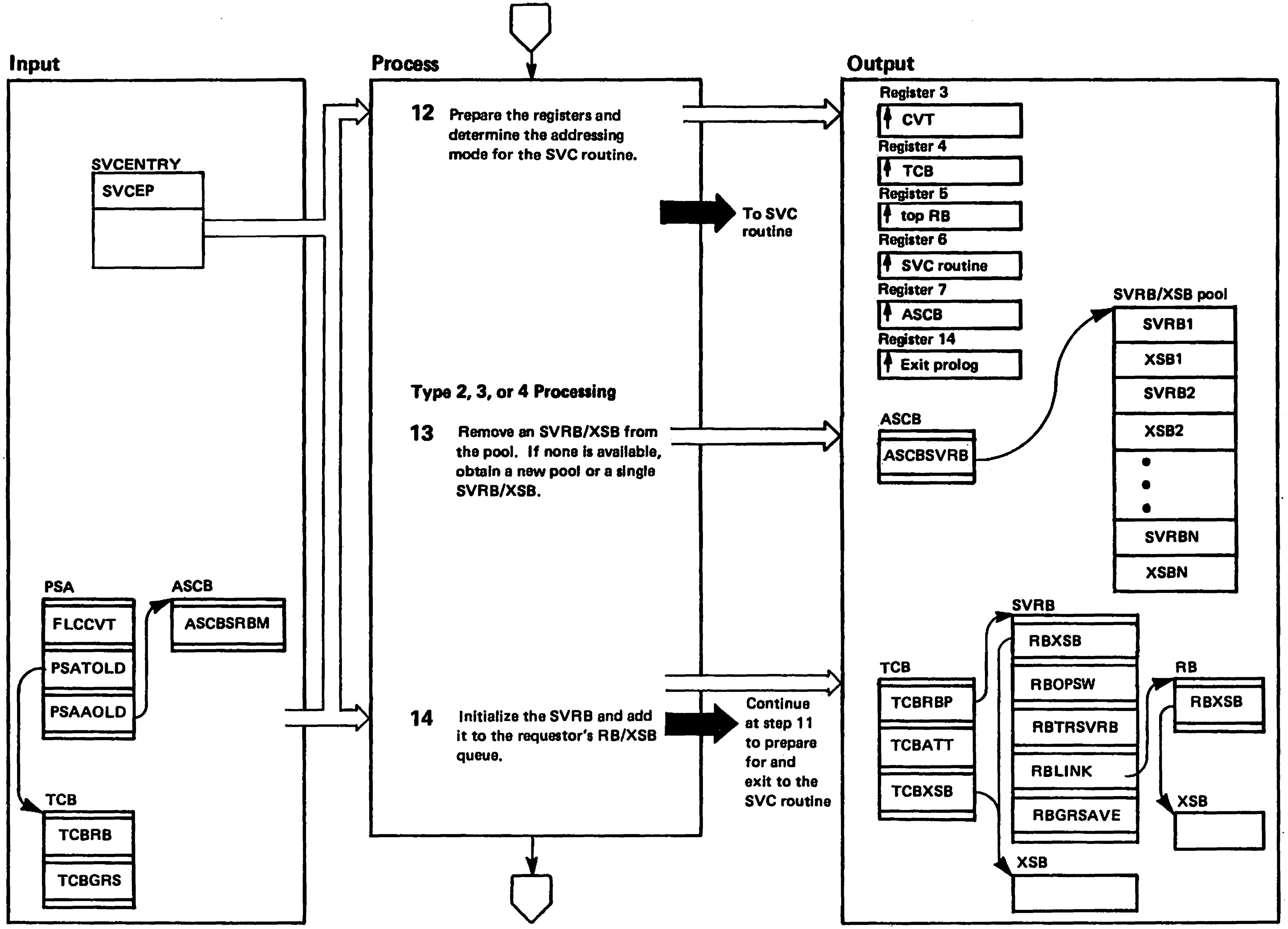
Type 1 SVC Processing

9 To process type 1 SVCs, the SVC FLIH obtains the local lock. The local lock request is made conditionally, since the SVC FLIH cannot be suspended (see lock manager routine, IEAVESLK). If the local lock is obtained, operation continues at step 10.
Otherwise, the SVC FLIH sets the high order bit of ASCBLSQH to indicate suspended on the local lock. This causes the lock manager to get control when the lock is released. The SVC FLIH then changes the RBOPSW in the requester's RB to point to the SVC instruction so that, when the requester is redispatched, the SVC is reissued. The SVC FLIH passes control to the dispatcher at entry point IEA0DS1 to redispatch the requester when the lock is available.

10 The SVC FLIH indicates type 1 processing in the ASCBFLG1 field, bit ASCBTYP1.

11 The SVC FLIH turns off the SVC super bit (PSASVC=0), makes the normal FRR stack the current one, and enables the PSW for I/O and external interrupts. The SVC FLIH now sets the operating environment for the requested SVC routine. First, the SVC FLIH obtains any locks that the SVC routine needs, as indicated in the SVC table entry.

Diagram SUP-40. SVC Interrupt Handler (IEAVESVC) (Part 7 of 14)



12 Prepare the registers and determine the addressing mode for the SVC routine.

To SVC routine

Type 2, 3, or 4 Processing

13 Remove an SVRB/XSB from the pool. If none is available, obtain a new pool or a single SVRB/XSB.

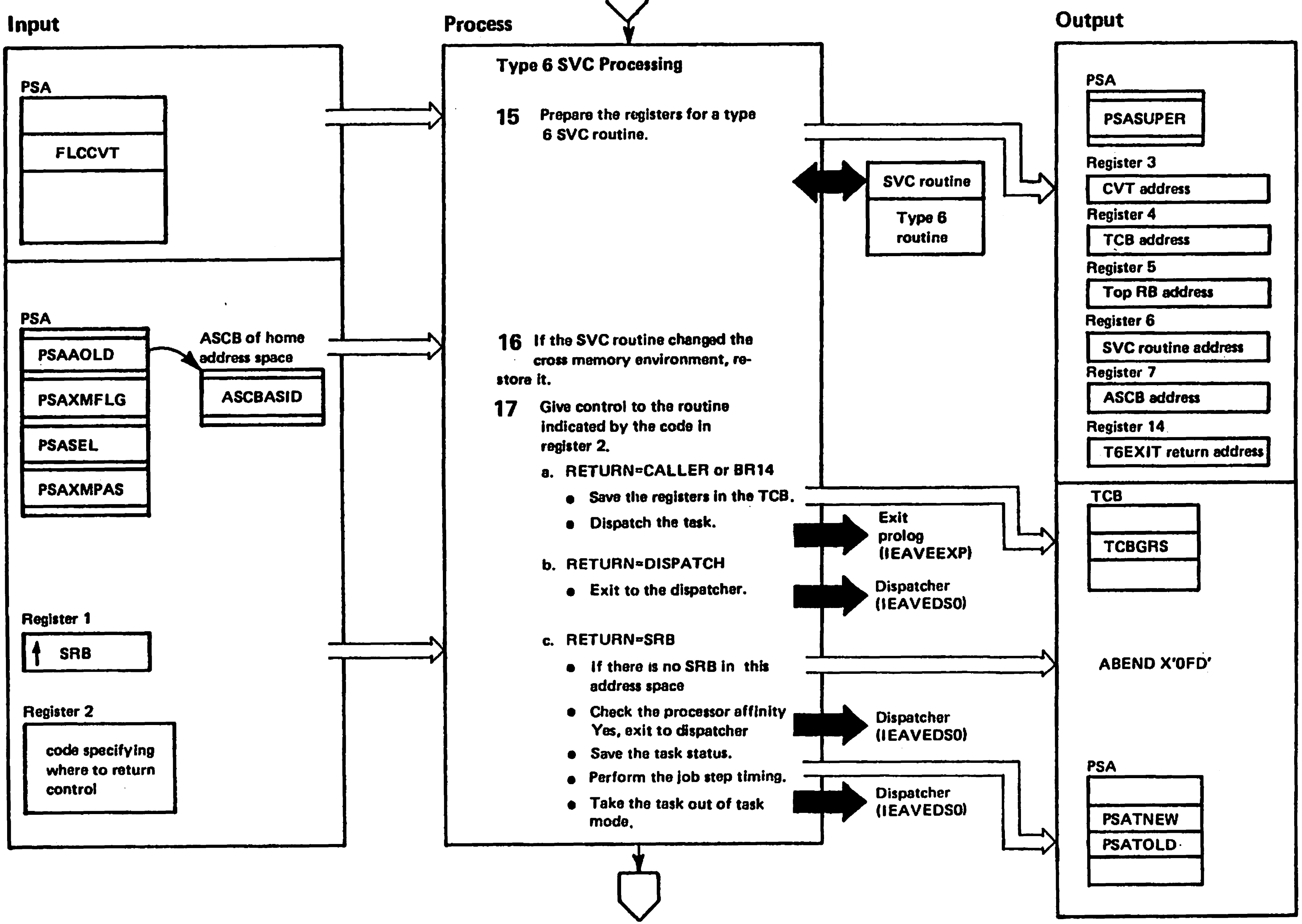
14 Initialize the SVRB and add it to the requestor's RB/XSB queue.

Continue at step 11 to prepare for and exit to the SVC routine

Diagram SUP-40. SVC Interrupt Handler (IEAVESVC) (Part 8 of 14)

Extended Description	Module	Label
12 THE SVC FLIH sets the proper values in input registers used by the SVC routine, and gives the SVC routine control using the entry point address in the SVC table entry. Registers 0, 1, 13 and 15 contain the same value as when the requester issued the SVC. If the high order bit of the entry point address (SVCEP) is a 1, the SVC receives control in 31 bit addressing mode; otherwise, the SVC receives control in 24 bit addressing mode.		
Types 2, 3, and 4 SVC Processing		
13 Because type 2, 3, and 4 SVC routines can be interrupted, the SVC FLIH obtains an SVRB/XSB pair in which to save status if interrupted. The SVC FLIH attempts to remove an SVRB/XSB from the pool chained from the ASCBSVRB field. If no pool elements are available (ASCBSVRB=0), the SVC FLIH issues a GETMAIN to expand the pool. If that GETMAIN fails, IEAVESVC attempts to get a single SVRB/XSB pair. If that fails, IEAVESVC calls RTM with ABEND code X'0F9', reason code 0. If the pool is obtained, IEAVESVC initializes it, queues it to the ASCBSVRB pool, and removes one of the available SVRB/XSB pairs.		
14 The SVC FLIH: • Initializes the SVRB. • Moves the issuer's registers from the TCBGRS field into the RBGRSAVE field in the SVRB. • Copies the ASCBSRBM value into the first byte of the RBOPSW to properly propagate PER monitoring. • If the SVC is type 3 or 4, indicates so by setting the RBTRSVRB bit to one. • Adds the SVRB/XSB to the task's RB/XSB queue. • Sets the TCBATT bit to one to suppress attentions.		

Diagram SUP-40. SVC Interrupt Handler (IEAVESVC) (Part 9 of 14)



Type 6 SVC Processing

15 Prepare the registers for a type 6 SVC routine.

16 If the SVC routine changed the cross memory environment, restore it.

17 Give control to the routine indicated by the code in register 2.

a. RETURN=CALLER or BR14

- Save the registers in the TCB.
- Dispatch the task.

b. RETURN=DISPATCH

- Exit to the dispatcher.

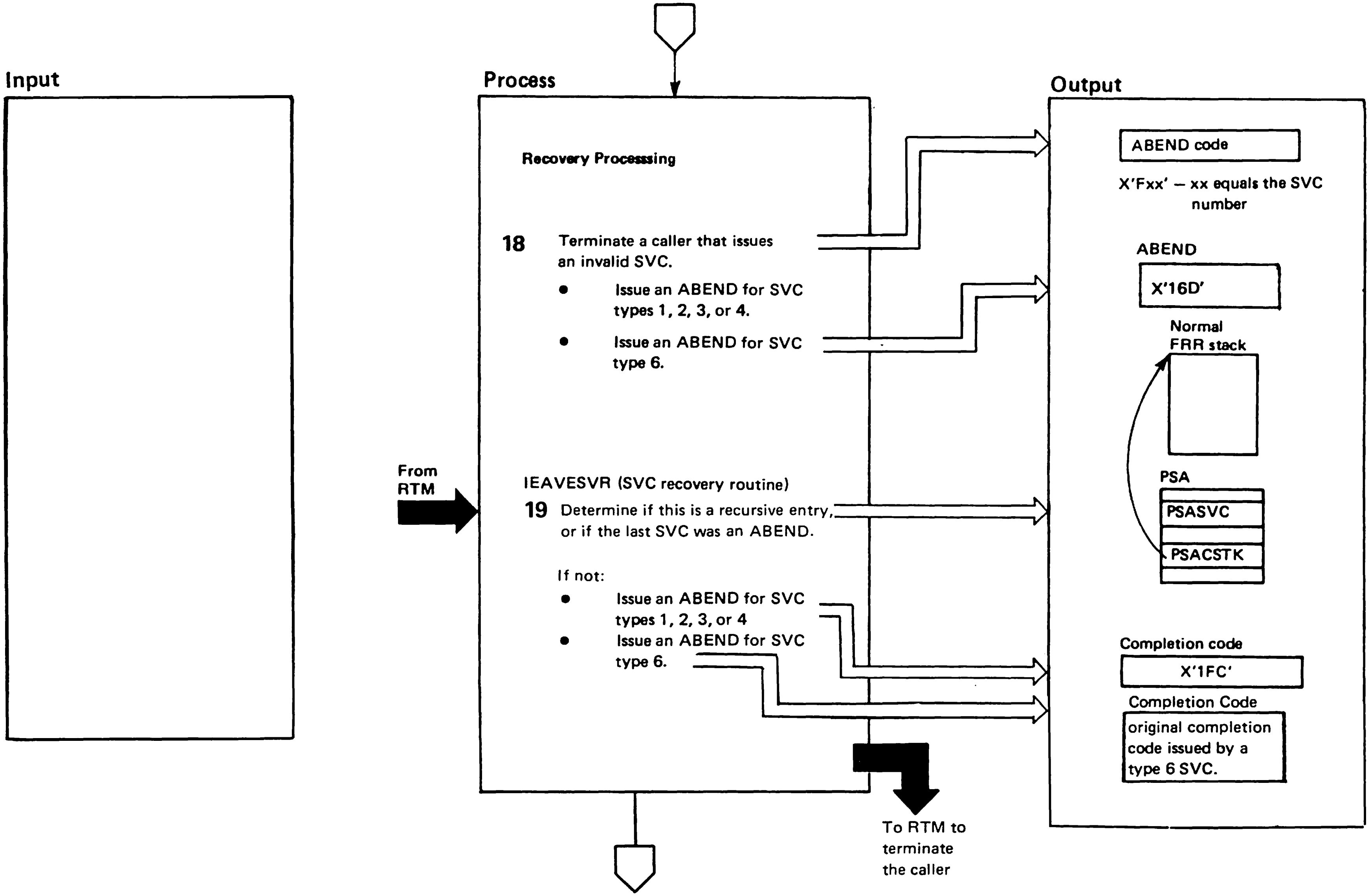
c. RETURN=SRB

- If there is no SRB in this address space
- Check the processor affinity Yes, exit to dispatcher
- Save the task status.
- Perform the job step timing.
- Take the task out of task mode.

Diagram SUP-40. SVC Interrupt Handler (IEAVESVC) (Part 10 of 14)

Extended Description	Module	Label
Type 6 SVC Processing		
15 The type 6 SVC processor sets the type 6 super bit (PSATYPE6=1) to indicate that a type 6 SVC is in control, and sets up the input registers for the type 6 SVC routine. It then branches to the routine. When the routine finishes executing, it returns control to this point in the SVC FLIH.		TYPE6SVC
16 Because the SVC FLIH executes with the primary and secondary address spaces equal to the home address space (PASID=SASID=HASID), after the SVC routine returns, the SVC FLIH checks whether this cross memory state still exists: ● If the SVC routine was executing in secondary mode, the SVC FLIH issues a SAC instruction to reestablish primary addressing mode. ● If the SVC routine changed the cross memory state, the SVC FLIH issues a CMSET SET macro to reestablish the home address space as both the primary and secondary address space. ● If the PCLINK stack is not empty, the SVC FLIH issues a PCLINK PURGE macro to free all the stack elements.		IEAVET6E and IEAVTY6C
17 Register 2 contains a code indicating to whom the SVC FLIH is to give control. The return options are: a. RETURN=CALLER or BR14. The SVC IH saves registers 0, 1, and 15 in the TCB and exits to the exit prolog, which directly re-dispatches the task. b. RETURN=DISPATCH. The SVC FLIH gives control to the dispatcher at entry point IEA0DS1. c. RETURN=SRB. The SVC FLIH checks whether there is an SRB scheduled for this address space. If there is no SRB, it issues an ABEND. If there is an SRB, the SVC FLIH saves the task status (floating point registers and timing data), performs the job step timing functions, takes the task out of task mode, and calls the global SRB dispatcher routine (entry point DSSRBRTN in the dispatcher) to directly dispatch the specified SRB.		IEAVET6E

Diagram SUP-40. SVC Interrupt Handler (IEAVESVC) (Part 11 of 14)



Recovery Processing

18 Terminate a caller that issues an invalid SVC.

- Issue an ABEND for SVC types 1, 2, 3, or 4.
- Issue an ABEND for SVC type 6.

IEAVESVR (SVC recovery routine)

19 Determine if this is a recursive entry, or if the last SVC was an ABEND.

If not:

- Issue an ABEND for SVC types 1, 2, 3, or 4
- Issue an ABEND for SVC type 6.

ABEND code

X'Fxx' – xx equals the SVC number

ABEND

X'16D'

Normal FRR stack

PSA

PSASVC

PSACSTK

Completion code

X'1FC'

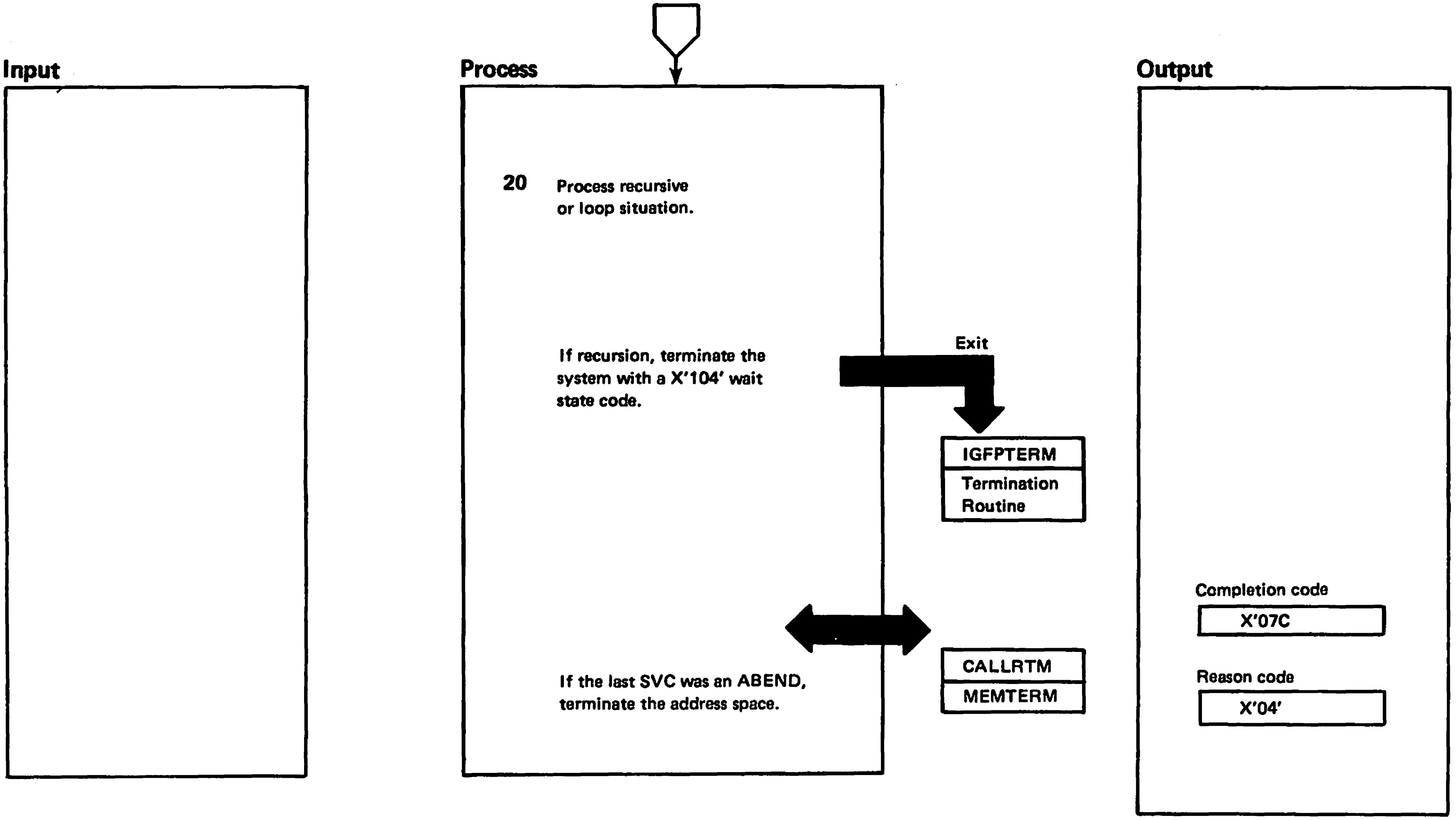
Completion Code

original completion code issued by a type 6 SVC.

Diagram SUP-40. SVC Interrupt Handler (IEAVESVC) (Part 12 of 14)

Extended Description	Module	Label
Recovery Processing		
18 The IGCERROR entry point receives control when the requester issues an SVC not listed in the SVC table. This routine terminates the requester with a code of X'Fxx', where xx is the number of the invalid SVC. The IGXERROR entry point receives control when the requester issues an invalid extended SVC. This routine terminates the requester with a completion code of X'16D'.		IGCERROR
19 The SVC FLIH FRR (functional recovery routine) tests for a recursive entry and checks field PSALSVCI to determine if the last SVC processed was an ABEND. (This field is set by IEAVESC and is used to handle a recursive ABEND situation.) If both tests fail, IEAVESVR clears the SVC indicator in the PSA, and sets the FRR stack pointer to the normal stack. If the SVC type was type 1, 2, 3, or 4, then this routine terminates the caller with a completion code of X'1FC'. If the SVC type was 6, it passes the original completion code issued by the type 6 SVC to the caller.	IEAVESVR	

Diagram SUP-40. SVC Interrupt Handler (IEAVESVC) (Part 13 of 14)



Input

Process

20 Process recursive or loop situation.

If recursion, terminate the system with a X'104' wait state code.

Exit

IGFPTERM

Termination Routine

If the last SVC was an ABEND, terminate the address space.

CALLRTM

MEMTERM

Output

Completion code

X'07C

Reason code

X'04'

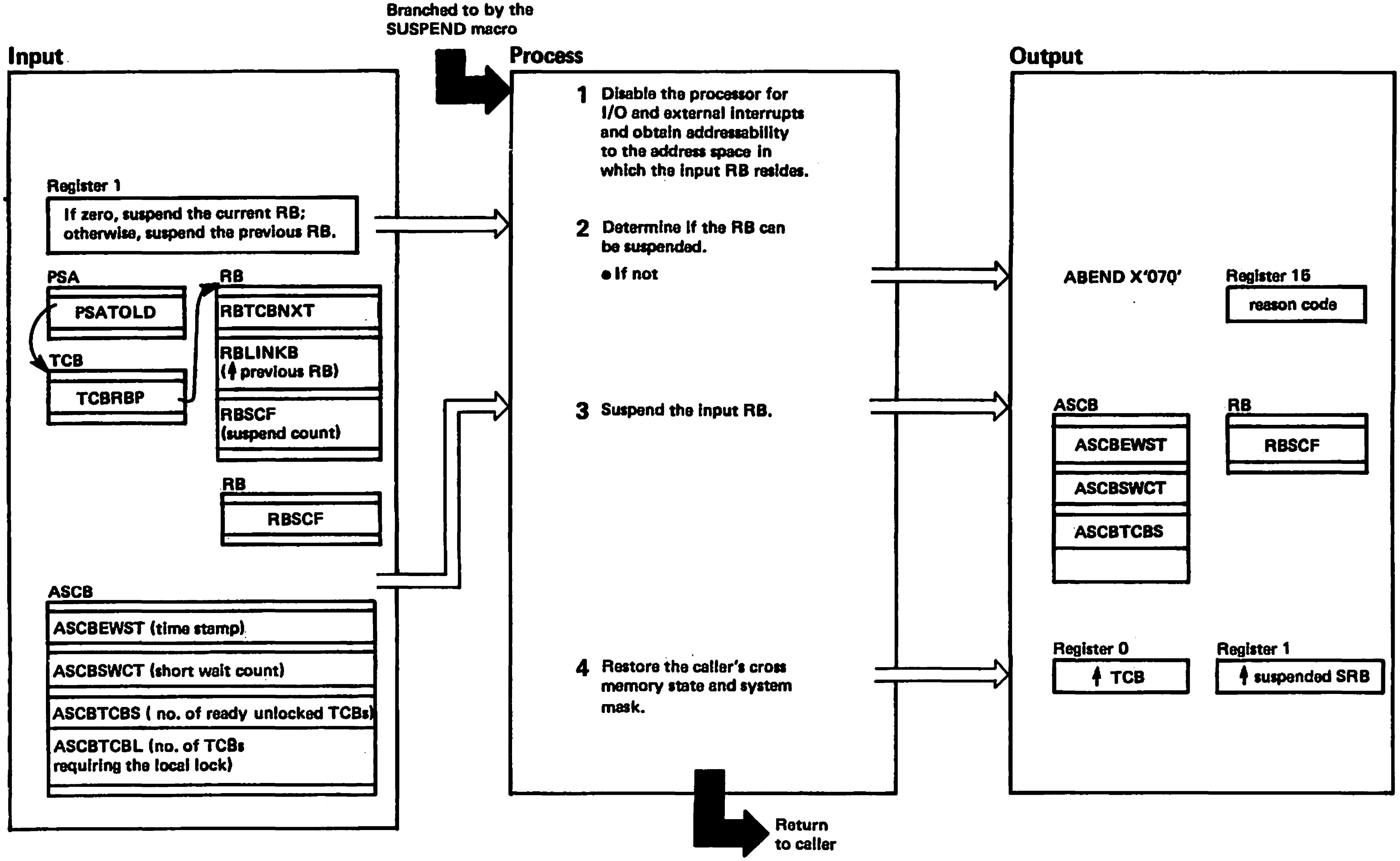
Diagram SUP-40. SVC Interrupt Handler (IEAVESVC) (Part 14 of 14)

Extended Description

20 Process recursive or loop situation.

- If recursion is detected, the system terminates. System termination prints an IEA740W message (Supervisor unable to recover from SVC D loop) at the console.
- If the last SVC was an ABEND, IEAVESVR terminates the address space. The MEMTERM completion cole is X'07C' with a reason code of X'04'.

Diagram SUP-41. SUSPEND Service Routine (IEAVETCL) (Part 1 of 2)



Branched to by the SUSPEND macro

Input

Register 1

If zero, suspend the current RB; otherwise, suspend the previous RB.

PSA

PSATOLD

TCB

TCBRBP

RB

RBTCBNXT

RBLINKB (↑ previous RB)

RBSCF (suspend count)

RB

RBSCF

ASCB

ASCBEWST (time stamp)

ASCBSWCT (short wait count)

ASCBTCBS (no. of ready unlocked TCBs)

ASCBTCBL (no. of TCBs requiring the local lock)

Process

1 Disable the processor for I/O and external interrupts and obtain addressability to the address space in which the input RB resides.

2 Determine if the RB can be suspended.

- If not

3 Suspend the input RB.

4 Restore the caller's cross memory state and system mask.

Return to caller

Output

ABEND X'070'

Register 15

reason code

ASCB

ASCBEWST

ASCBSWCT

ASCBTCBS

RB

RBSCF

Register 0

↑ TCB

Register 1

↑ suspended SRB

Diagram SUP-41. SUSPEND Service Routine (IEAVETCL) (Part 2 of 2)

Extended Description	Module	Label
When a SUSPEND macro is issued, IEAVEGLU receives control at entry point IEAVSPND, converts to 31 bit addressing mode and branches to IEAVETCL at entry point IEAVSPN1. (Return will be made directly to the issuer of the SUSPEND macro.) The function performed at IEAVSPN1 is to place an RB in a suspended state.		
1 IEAVETCL issues a CMSET SET macro to obtain addressability to the home address space of the RB being suspended. This is the address space pointed to by PSAAOLD.	IEAVETCL	IEAVSPN1

2 IEAVETCL cannot suspend the RB if:

- The RB suspend count (RBSCF) is nonzero.
- The caller requested that the previous RB be suspended, but no previous RB exists.

In either case, IEAVETCL terminates the caller with ABEND X'070' and reason code 48 or 52, respectively.

3 When suspending the current RB, IEAVETCL:

- Subtracts one from the count of unlocked, ready TCBs (ASCBTCBS). (This is done because the TCB is not dispatchable when the top RB is suspended.) If this makes the count of ready TCBs (both unlocked TCBs and those requiring the local lock) zero, IEAVETCL adds one to the short wait count (ASCBSWCT) and time stamps the wait time.
- Sets the RB's suspend count to one.

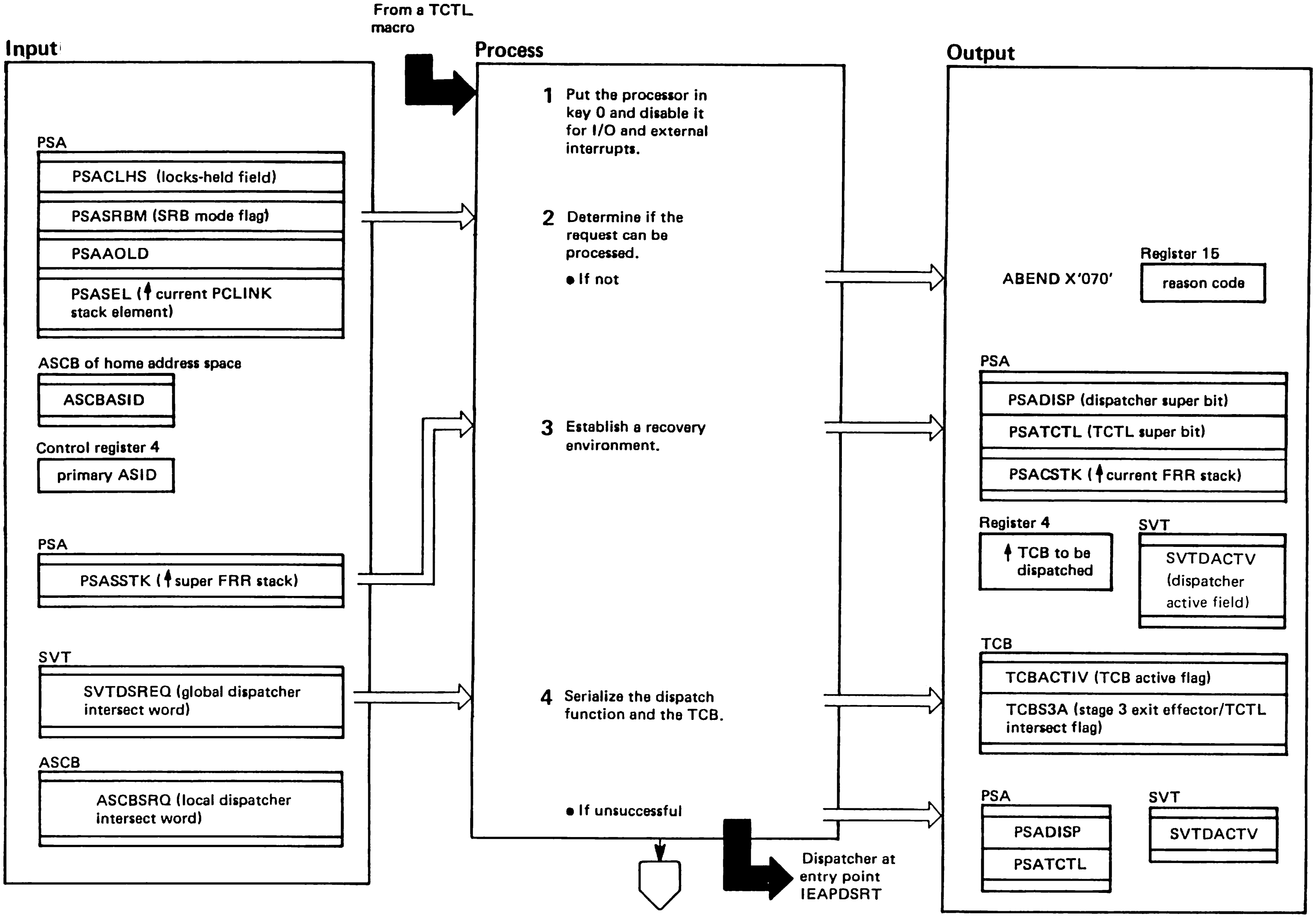
To suspend the previous RB, IEAVETCL sets the RB's suspend count to one. Since the TCB is still dispatchable, no additional processing is required.

4 IEAVETCL issues a CMSET RESET macro to restore the caller's cross memory state. It puts the target TCB's address into register 0, the suspended RB's address into register 1, and returns to the caller.

Recovery Processing

When entered to process a SUSPEND macro, IEAVETCL does not establish its own recovery environment.

Diagram SUP-42. TCTL Service Routine (IEAVETCL) (Part 1 of 6)


From a TCTL macro
Input
PSA
PSACLHS (locks-held field)
PSASRBM (SRB mode flag)
PSAAOLD
PSASEL (↑ current PCLINK stack element)
ASCB of home address space
ASCBASID
Control register 4
primary ASID
PSA
PSASSTK (↑ super FRR stack)
SVT
SVTDSREQ (global dispatcher intersect word)
ASCB
ASCBSRQ (local dispatcher intersect word)
Process
Output
ABEND X'070'
Register 15
reason code
PSA
PSADISP (dispatcher super bit)
PSATCTL (TCTL super bit)
PSACSTK (↑ current FRR stack)
Register 4
↑ TCB to be dispatched
SVT
SVTDACTV (dispatcher active field)
TCB
TCBACTIV (TCB active flag)
TCBS3A (stage 3 exit effector/TCTL intersect flag)
PSA
PSADISP
PSATCTL
SVT
SVTDACTV
Dispatcher at entry point IEAPDSRT

1 Put the processor in key 0 and disable it for I/O and external interrupts.

2 Determine if the request can be processed.

- If not

3 Establish a recovery environment.

4 Serialize the dispatch function and the TCB.

- If unsuccessful

Diagram SUP-42. TCTL Service Routine (IEAVETCL) (Part 2 of 6)

Extended Description	Module	Label
When a TCTL macro is issued, IEAVEGLU receives control at entry point IEAVTCTL, converts to 31 bit addressing mode, and branches to IEAVETCL at entry point IEAVTCT1. IEAVETCL transfers control directly to the specified TCB. SRBs that dispatch an associated TCB on exit use this service as a performance path. The RESUME service routine also invokes IEAVETCL when an SRB issues a RESUME macro requesting that control be given to the resumed TCB/RB after the resume function is completed. In this case, the RESUME service routine branch enters IEAVETCL at entry point JOINTCT. At entry, register 4 points to the TCB to be given control.		
1 IEAVETCL sets PSW key zero and disables the processor for I/O and external interrupts.	IEAVETCL	IEAVTCT1
2 For the request to be valid, the caller must be an unlocked SRB in home addressing mode with an empty PCLINK stack. (IEAVETCL will flush the PCLINK stack if it is not empty.) If it is not, IEAVETCL terminates the caller with ABEND X'070' and an appropriate reason code. (See *System Codes* for specific reason codes.)		
3 IEAVETCL makes the dispatcher-I/O-SVC super FRR stack the current one and sets to one the dispatcher and TCTL super bits (PSADISP and PSATCTL). As a result, if the TCTL service routine issues an ABEND, the supervisor's recovery routine (IEAVESPR) routes control to IEAVETCR, an FRR entry point within IEAVETCL. (See "Recovery Processing" for a description of IEAVETCR.)		
4 To serialize the dispatch function, IEAVETCL: • Puts the processor's logical address in the appropriate dispatcher active byte (SVTDACTV). This prevents STATUS from changing the TCB's dispatchability. • Ensures that no other functions are intersecting with the dispatcher. If either the global or local intersect word (SVTDSREQ or ASCBSRQ) is nonzero, IEAVETCL cannot perform the TCTL function. It clears the dispatcher active and super bits and returns to the dispatcher's SRB exit routine (entry point IEAPDSRT).	IEAVETCL	
IEAVETCL attempts to serialize the TCB with the dispatcher, STATUS, and the stage 3 exit effector by setting the TCB active and stage 3 exit effector/TCTL intersect flags (TCBACTIV and TCBS3A). It uses a compare and swap instruction to do so. If the instruction fails because the intersect flags are already on, IEAVETCL cannot complete TCTL processing. It clears the dispatcher active and super bits and returns to the dispatcher's SRB exit routine (entry point IEAPDSRT).		GETXSCT3

Diagram SUP-42. TCTL Service Routine (IEAVETCL) (Part 3 of 6)

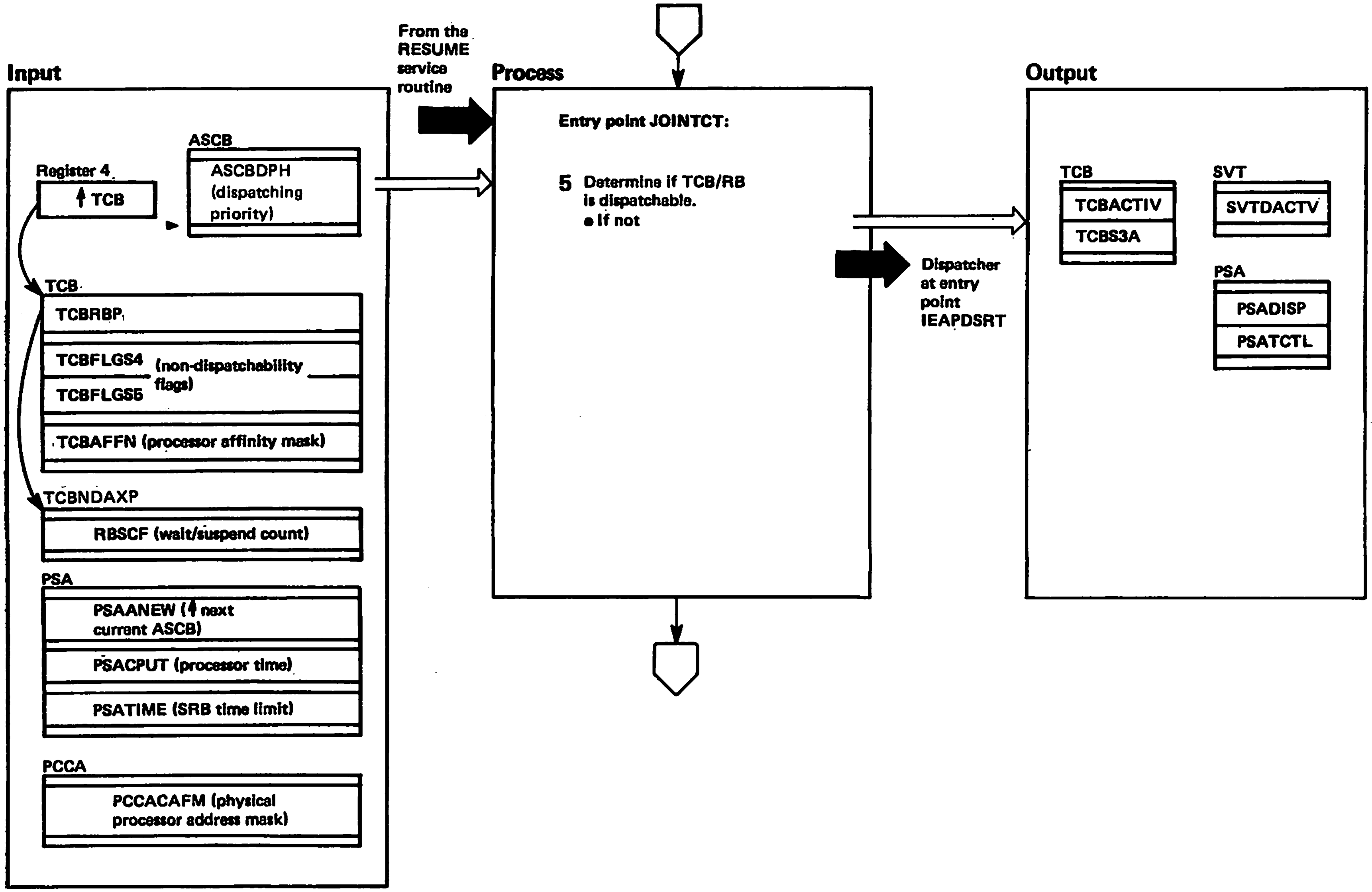

From the RESUME service routine
Input
Register 4
↑ TCB
ASCB
ASCBDPH (dispatching priority)
TCB
TCBRBP
TCBFLGS4
TCBFLGS5
(non-dispatchability flags)
TCBAFFN (processor affinity mask)
TCBNDAXP
RBSCF (wait/suspend count)
PSA
PSAANEW (↑ next current ASCB)
PSACPUT (processor time)
PSATIME (SRB time limit)
PCCA
PCCACAFM (physical processor address mask)
Process
Entry point JOINTCT:
5 Determine if TCB/RB is dispatchable.
● If not
Dispatcher at entry point IEAPDSRT
Output
TCB
TCBACTIV
TCBS3A
SVT
SVTDACTV
PSA
PSADISP
PSATCTL

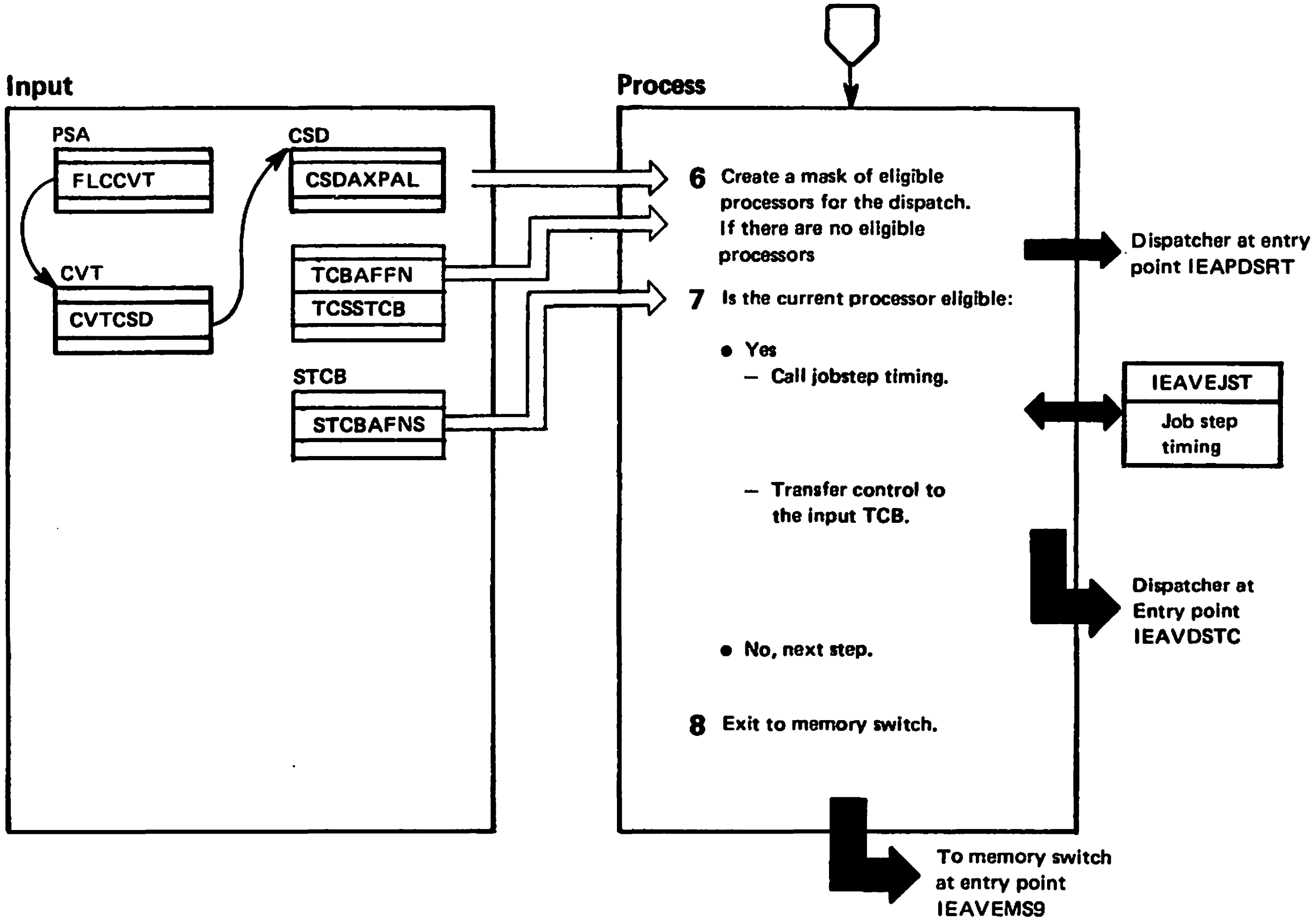
Diagram SUP-42. TCTL Service Routine (IEAVETCL) (Part 4 of 6)

Extended Description	Module	Label
5 IEAVETCL checks the following conditions to determine whether the TCB/RB can be dispatched.		JOINTCT

- No TCB dispatchability flags are set
- The top RB is not waiting or suspended
- The TCB does not have affinity to a processor other than the one executing IEAVETCL
- The address space pointed to by the PSAANEW field does not have higher priority than the current address space

If any of these conditions is not met, IEAVETCL cannot transfer control to the TCB/RB. It clears the TCB intersect flags, the dispatcher active flag, and the super bits, and returns to the dispatcher's SRB exit routine (entry point IEAPDSRT).

Diagram SUP-42. TCTL Service Routine (IEAVETCL) (Part 5 of 6)


Input
PSA
FLCCVT
CVT
CVTCSD
CSD
CSDAXPAL
TCBAFFN
TCSSTCB
STCB
STCBAFNS
Process
Dispatcher at entry point IEAPDSRT
IEAVEJST
Job step timing
Dispatcher at Entry point IEAVDSTC
To memory switch at entry point IEAVEMS9

6 Create a mask of eligible processors for the dispatch. If there are no eligible processors

7 Is the current processor eligible:

- Yes
  - Call jobstep timing.
  - Transfer control to the input TCB.
- No, next step.

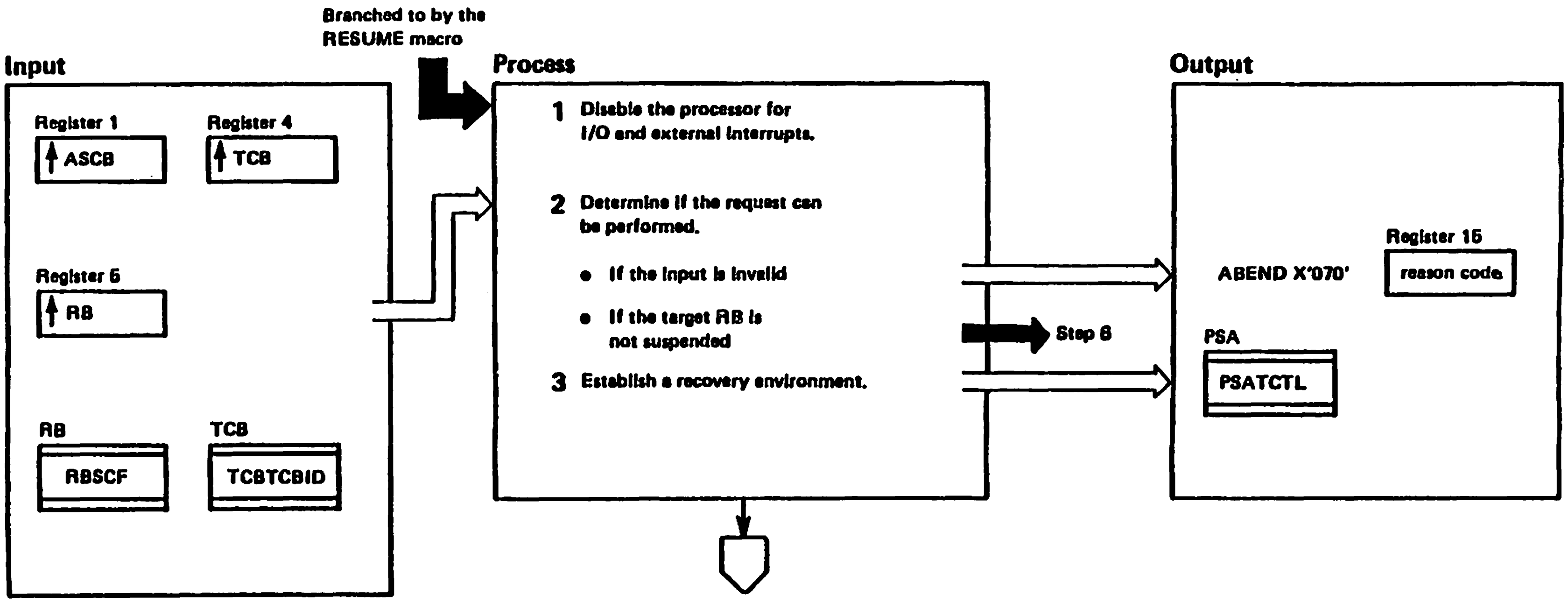
8 Exit to memory switch.

Diagram SUP-42. TCTL Service Routine (IEAVETCL) (Part 6 of 6)

Extended Description	Module	Label
6 Create a mask of eligible processors for the dispatch of the input TCB. If the task requires a processor with a vector facility, the mask is based on CSDCPUVF and STCBAFNS. If the resulting mask is 0, exit immediately to the dispatcher at IEAPDSRT, the SRB exit address. Otherwise, store the mask in TCBAFFN. If a vector facility is not required, the mask is based on TCBAFFN, TCBNDAXP and CSDAXPAL.		
7 The physical processor address mask (PCCACAFM), is logically ANDed with the eligible processor mask computed in step 6. If the current CPU is eligible, 1. ● Jobstep timing is invoked at entry point DSJSTSRB 2. ● IEAVETCL does the following: ● –Sets PSAANEW to the ASCB address of the home address space ● –Sets to zero the processor's SRB mode flags, SVTDACTV, TCTL, and dispatcher superbits ● –Branches to dispatcher entry point IEAVDSTC to give control to the specified TCB.		
8 IEAVETCL performs the following: ● Sets up the interface to memory switch using TCBAFFN and TCBNDAXP or STCBVAFN and STCBAFNS. ● Sets the TCTL superbits, TCBACTIV and TCBS3A to zero. ● Sets the SVTDACTV intersect byte for the current processor to zero. ● Sets up register 14 with the SRB return address in the dispatcher. ● Gives control to memory switch entry point (IEAVEMS9).		

Extended Description	Module	Label
Recovery Processing		
When an error occurs while IEAVETCL is performing TCTL services, RTM gives control to the supervisor control FRR (IEAVESPR). IEAVESPR routes control to IEAVETCL	IEAVESPR	
at entry point IEAVETCR. IEAVETCR:	IEAVETCL	IEAVETCR
● Records diagnostic information in the SDWA and the SDWA variable recording area		
● Sets to zero the TCBACTIV and TCBS3A flags in the input TCB, the executing processor's dispatcher active byte (SVTDACTV), and the TCTL and dispatcher super bits		TCRACTV
● Returns to IEAVESPR, which retries at label IEAVEDSE within IEAVEDSR.		

Diagram SUP-43. RESUME Service Routine (IEAVETCL) (Part 1 of 6)



1 Disable the processor for I/O and external interrupts.

2 Determine if the request can be performed.

- If the input is invalid
- If the target RB is not suspended

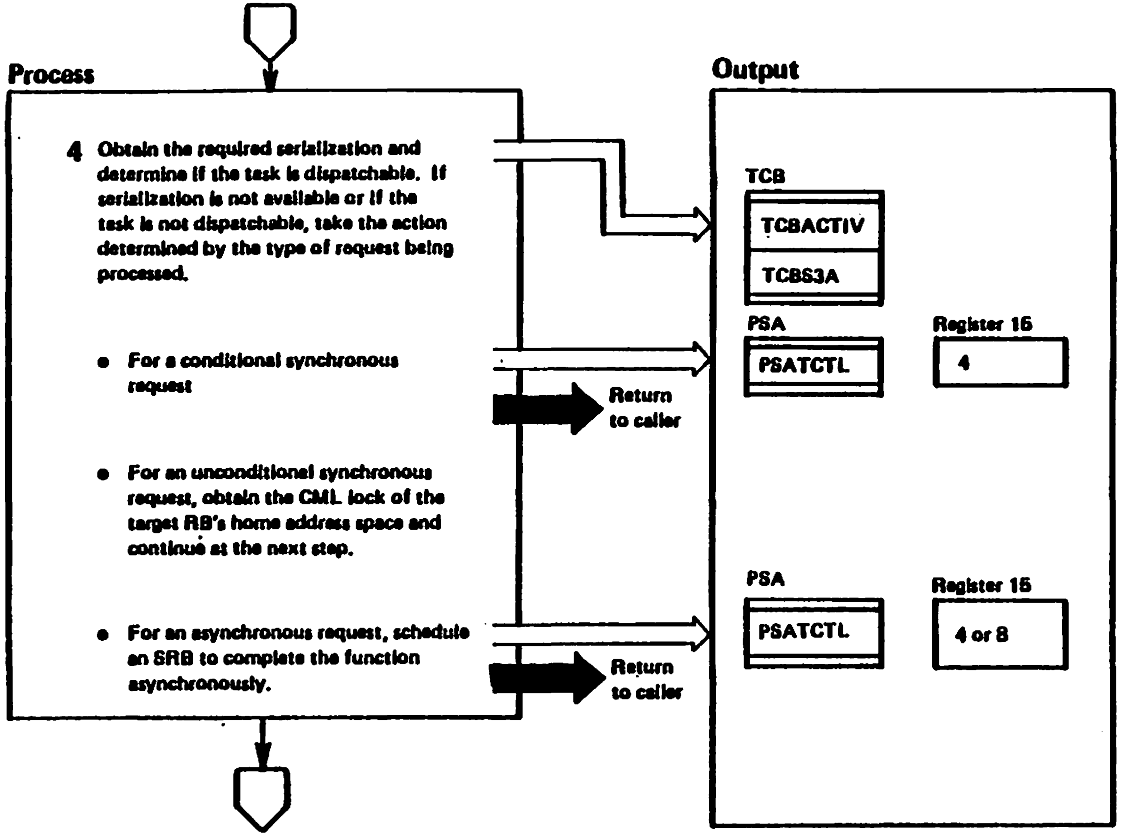
3 Establish a recovery environment.

Diagram SUP-43. RESUME Service Routine (IEAVETCL) (Part 2 of 6)

Extended Description	Module	Label
When a RESUME macro is issued, IEAVEGLU receives control. It converts to 31-bit addressing mode and branches to IEAVETCL at the appropriate entry point. (See SUP-36 for the correspondence between the RESUME entry points in IEAVEGLU and IEAVETCL.) IEAVETCL receives control to remove a specified RB from a suspended state. IEAVETCL tries to do this synchronously. If unsuccessful because the required serialization is not available, IEAVETCL's actions depend on the type of RESUME macro being processed. The types and IEAVETCL's corresponding actions are listed below. Note that the request type refers to the action IEAVETCL takes *after* it has determined that it cannot perform the resume function synchronously without further serialization.		
● A conditional synchronous RESUME — Bypasses the resume function and returns to the caller. ● An unconditional synchronous RESUME — Obtains the local lock of the target address space and completes the resume function before returning control to the caller. ● A conditional asynchronous RESUME — Conditionally obtains an SRB and schedules it to complete the resume function asynchronously. If no SRB is available, IEAVETCL does not complete the function (no GETMAIN is issued). ● An unconditional asynchronous RESUME — Obtains an SRB and schedules it to complete the resume function asynchronously. IEAVETCL might have to issue a GETMAIN to allocate the SRB.		
At entry, register 5 points to the RB to be resumed (the target RB), register 4 points to the TCB from which the RB is chained, and register 1 contains the ASCB address of the target RB's home address space.		
1 IEAVETCL disables the processor for external and I/O interrupts to prevent the address space in which the target RB resides from being swapped out.		COMMON

Extended Description	Module	Label
2 IEAVETCL checks the resume request to determine if:		
● The caller has established current addressability to the address space in which the suspended RB and task reside. If not, IEAVETCL issues ABEND X'070' with reason code 20.		CHKASCB
● The TCB acronym is valid. If it is not, IEAVETCL issues ABEND X'070' with reason code 16. ● The RB suspend count (RBSCF) is nonzero. If the suspend count is zero, the RB is not suspended and IEAVETCL exits at step 6. If the suspend count is greater than 1, IEAVETCL issues ABEND X'070' with reason code X'56'.		
IEAVETCL also checks whether the RB's address is valid by making a dummy reference to the RB. If the RB address is invalid, the reference causes a program interrupt.		
3 IEAVETCL puts an FRR on the current FRR stack and sets to one the TCTL super bit. This allows the RESUME/TCTL recovery routine (IEAVETCR) to receive control disabled when an error occurs. See "Recovery Processing" for a description of IEAVETCR.		

Diagram SUP-43. RESUME Service Routine (IEAVETCL) (Part 3 of 6)



Process

4 Obtain the required serialization and determine if the task is dispatchable. If serialization is not available or if the task is not dispatchable, take the action determined by the type of request being processed.

- For a conditional synchronous request
- For an unconditional synchronous request, obtain the CML lock of the target RB's home address space and continue at the next step.
- For an asynchronous request, schedule an SRB to complete the function asynchronously.

Diagram SUP-43. RESUME Service Routine (IEAVETCL) (Part 4 of 6)

Extended Description	Module	Label
4 IEAVETCL attempts to serialize the TCB with the dispatcher, STATUS, and stage 3 exit effector by setting the TCB active and stage 3 exit effector intersect flags (TCBACTIV and TCBS3A). It uses a compare and swap (CS) instruction to do so. If successful and if the TCB is dispatchable (TCBFLGS4=0), IEAVETCL continues processing at the next step. If the CS instruction fails because the intersect flags are already set or if the CS instruction is successful but the TCB is not dispatchable (TCBFLGS4≠0), IEAVETCL cannot perform the resume function synchronously without further serialization. One exception to this is when the TCB is active on this processor. In this case, the task issued a RESUME for an RB in itself and serialization is not required unless the TCB is nondispatchable. (When the TCB is nondispatchable, serialization other than the intersect flags is required because the QUIESCE function might be preparing to swap out the target address space.) If either the intersect flags are already set or the TCB is nondispatchable, IEAVETCL's actions depend on the type of resume macro being processed and are described below. If IEAVETCL was able to set the intersect flags, it resets them to zero before taking the action described.	IEAVETCL	GETXSCT1
• *Conditional synchronous requests:* IEAVETCL does not complete the resume function. It deletes the FRR, resets the TCTL bit, restores the caller's system mask, and returns to the caller with a return code of 4 to indicate that the resume function was not performed.	IEAVETCL	RCSC
• *Unconditional synchronous requests:* IEAVETCL obtains further serialization. If the caller does not already hold the CML lock of the address space in which the input RB resides, IEAVETCL obtains it. If the caller holds any other CML lock, IEAVETCL issues ABEND X'070' with reason code 44. After obtaining the lock, IEAVETCL updates the resume registers and PSW, if requested by the caller to do so. IEAVETCL resets the specified RB's suspend count to zero, then tries again to obtain the intersect flags (TCBACTIV and TCBS3A). If successful, and if the CML lock was not already held at entry, IEAVETCL releases it. If the TCB is dispatchable (TCBFLGS4=0), IEAVETCL continues at step 5. Otherwise, processing continues at step 6.	IEAVETCL	NOTCOND
4 (continued)		
• *Conditional asynchronous requests:* IEAVETCL makes no further serialization attempts, but it tries to perform the resume function asynchronously. It issues a conditional request for an SRB. If one can be obtained from the supervisor's SRB pool using a GETSRB macro, IEAVETCL initializes and schedules the SRB to enter this module at entry point IEAVRSRB, where an unconditional synchronous RESUME macro is issued. After scheduling the SRB, IEAVETCL: — Turns off the TCTL super bit — Deletes the FRR — Sets a return code of 4 to indicate that an SRB was scheduled to perform the resume function asynchronously — Restores the caller's system mask — Returns to the caller If no SRBs are available, IEAVETCL cannot do the resume. It performs the cleanup functions described above, except that it sets a return code of 8 to indicate that the resume function was not completed.	IEAVETCL	RSCA
• *Unconditional asynchronous requests:* IEAVETCL performs the same functions as it does for conditional asynchronous requests, except that it obtains the SRB unconditionally.	IEAVETCL	RSUA

Diagram SUP-43. RESUME Service Routine (IEAVETCL) (Part 5 of 6)

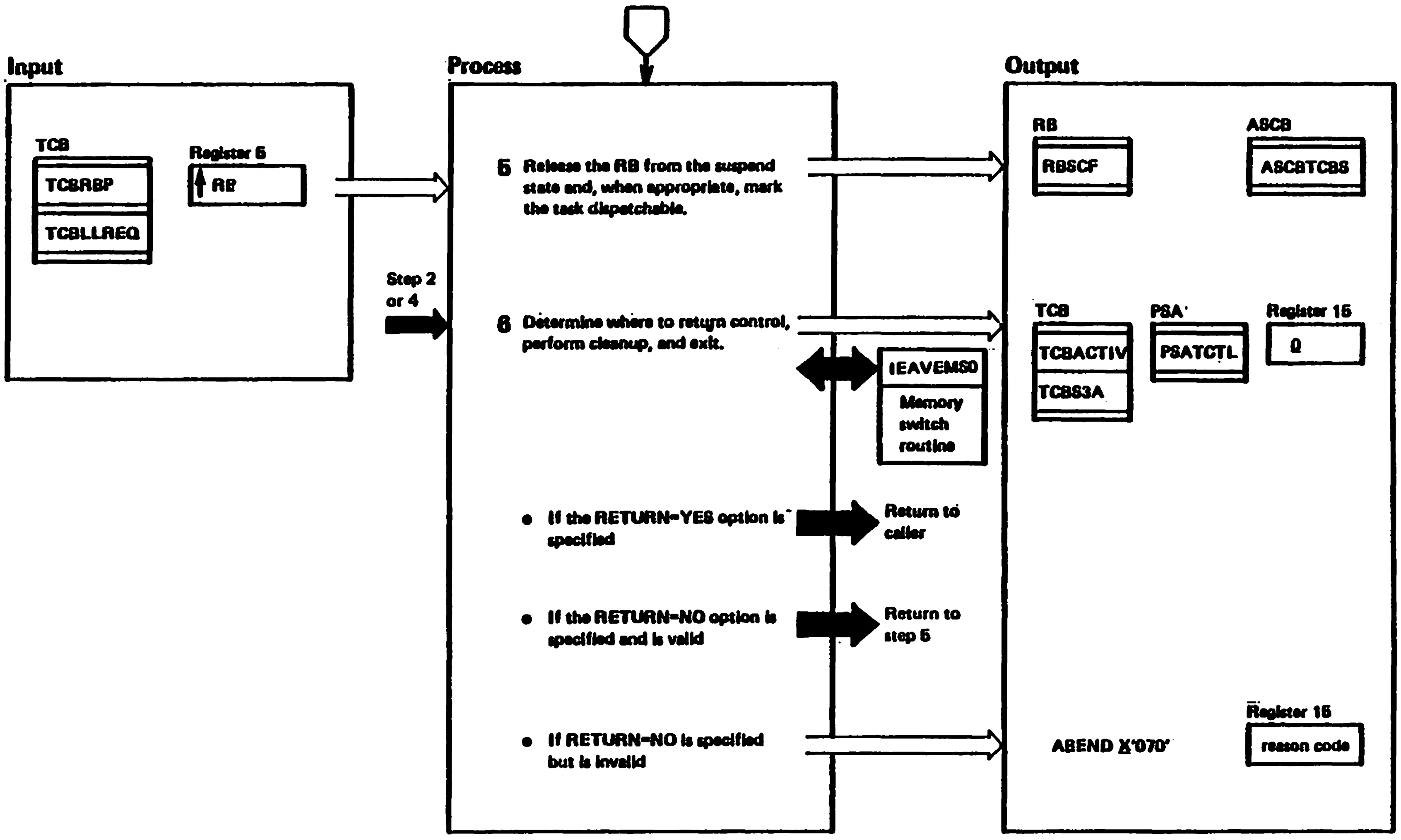
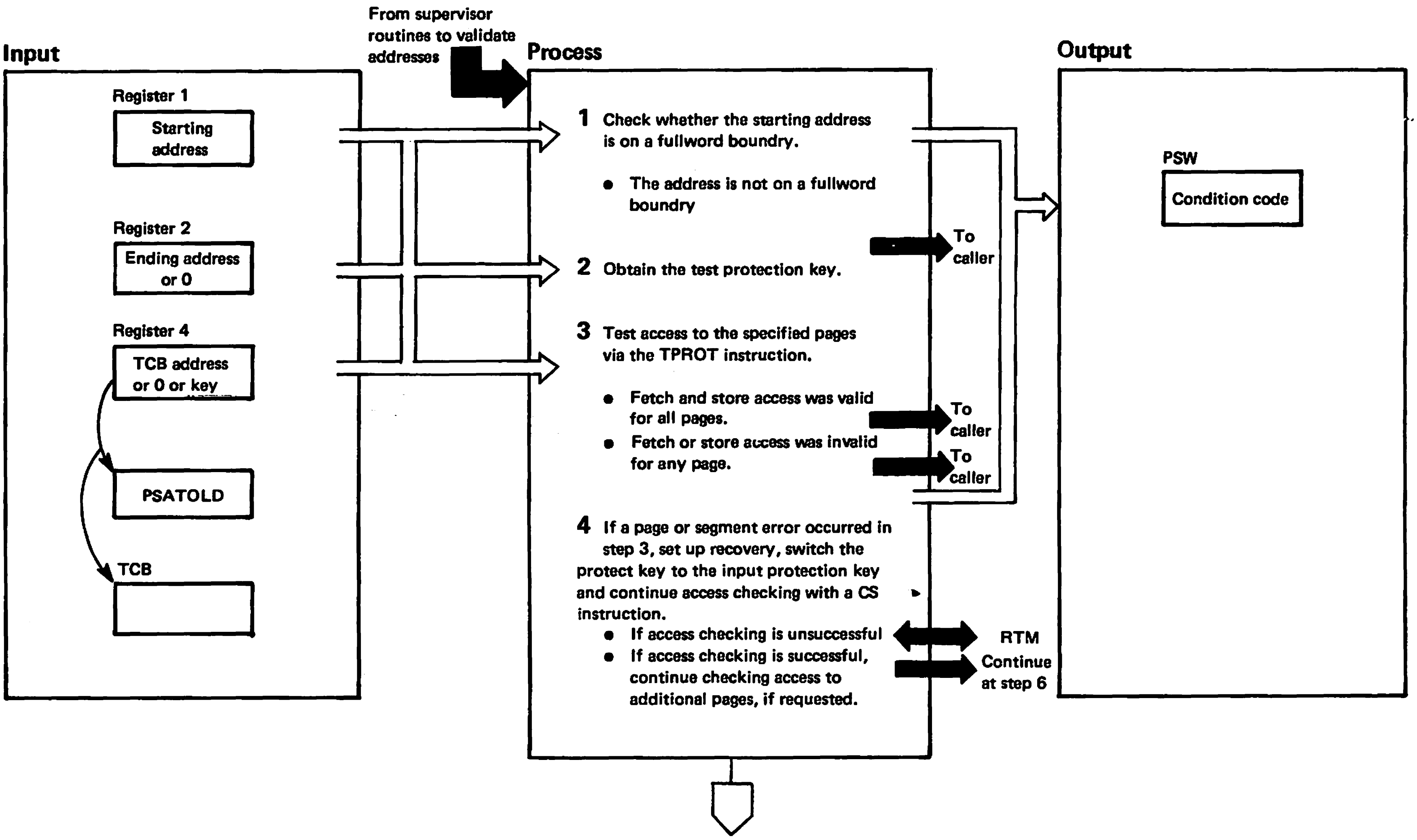

Input
TCB
TCBRBP
TCBLLREQ
Register 6
RB
Step 2 or 4
Process
5 Release the RB from the suspend state and, when appropriate, mark the task dispatchable.
6 Determine where to return control, perform cleanup, and exit.
IEAVEMS0
Memory switch routine
• If the RETURN=YES option is specified
Return to caller
• If the RETURN=NO option is specified and is valid
Return to step 5
• If RETURN=NO is specified but is invalid
Output
RB
RBSCF
ASCB
ASCBTCBS
TCB
TCBACTIV
TCBS3A
PSA
PSATCTL
Register 15
0
ABEND X'070'
Register 15
reason code

Diagram SUP-43. RESUME Service Routine (IEAVETCL) (Part 6 of 6)

Extended Description	Module	Label
5 IEAVETCL releases the RB from the suspend state by setting the RB suspend count (RBSCF) to zero.		MAINLINE
If the task does not require a local lock (TCBLLREQ=0), and if the RB released from the suspend state is the current RB (TCBRBP = the target RB address), the task is now dispatchable. IEAVETCL updates the resume registers and PSW if the caller requests. IEAVETCL adds one to the count of ready, unlocked TCBs (ASCBTCBS).		
6 The caller can specify that, after completing the resume function, IEAVETCL either return to the caller or transfer control to the resumed TCB/RB. If IEAVETCL is to return control to the caller, it:		
● Turns off the intersect flags ● Turns off the TCTL super bit ● Deletes the FRR ● If the resumed RB is the current one, and if the task is dispatchable, calls IEAVEMS0 to notify other processors that work is ready (entry point IEAVEMS9) ● Sets a return code of 0 ● Restores the caller's status ● Branches to the caller		
If the caller requested that IEAVETCL exit to the resumed TCB/RB (the RETURN=NO option was specified), IEAVETCL determines if the request is valid. The RETURN=NO option can be specified only by callers executing in SRB mode and in home mode. IEAVETCL terminates all other requestors with ABEND X'070' and reason code 12 (not in SRB mode) or 24 (not in home mode). If the PCLINK stack is not empty, it is purged. If the caller holds any locks, IEAVETCL abends the caller with reason code 28. If the request is valid, control is passed to the TCTL service routine at label JOINTCT to attempt to give control to the resumed task.		NORTN

Extended Description	Module	Label
Recovery Processing:		
When an error occurs while IEAVETCL is processing a RESUME macro, RTM gives control to IEAVETCL's FRR (IEAVETCR). The FRR:		IEAVETCR
● Records diagnostic information in the SDWA. ● Establishes a nested FRR (entry point RSFRR2E within IEAVETCL) to protect this routine against errors in external routines (SDUMP, FREESRB) and errors in references to private storage. ● Adds one to the count of ready TCBs (ASCBTCBS). If IEAVETCL had not yet decreased the count, this makes it high. ● Issues an SDUMP of the resume TCB's address space. ● If IEAVETCL had set the TCBACTIV and TCBS3A flags, resets them to zero. ● If IEAVETCL had obtained an SRB, frees it. ● Sets the TCTL super bit to zero. ● Issues a SETRP to percolate the error and free the local lock (if IEAVETCL obtained it).		
If an error occurs while IEAVETCR is executing, RTM gives control to entry point RSFRR2E. This FRR:		RSFRR2G
● Records diagnostic information in the SDWA. ● Sets the TCTL super bit to zero. ● Issues a SETRP to percolate the error and free the local lock (if IEAVETCL obtained it).		

Diagram SUP-44. Validity Check Processing (IEAVEVAL) (Part 1 of 4)



1 Check whether the starting address is on a fullword boundry.

- The address is not on a fullword boundry

2 Obtain the test protection key.

3 Test access to the specified pages via the TPROT instruction.

- Fetch and store access was valid for all pages.
- Fetch or store access was invalid for any page.

4 If a page or segment error occurred in step 3, set up recovery, switch the protect key to the input protection key and continue access checking with a CS instruction.

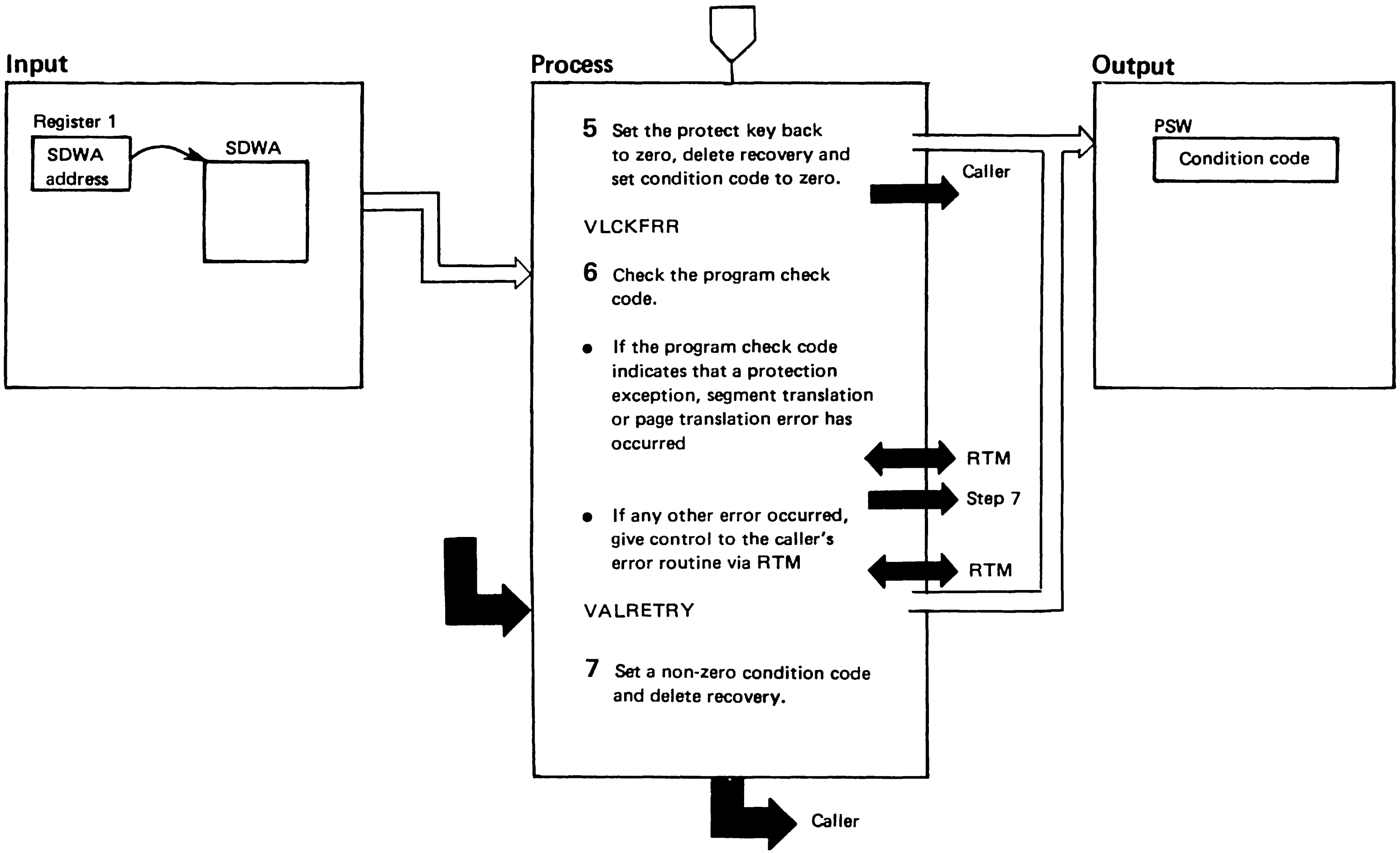
- If access checking is unsuccessful
- If access checking is successful, continue checking access to additional pages, if requested.

Diagram SUP-44. Validity Check Processing (IEAVEVAL) (Part 2 of 4)

Extended Description	Module	Label
The validity check processing determines whether an address or address range belongs in the key of a specified program. Supervisor service routines branch into validity check, giving as input the address (or range) of the area being checked.		
1 Validity check gives control back to the caller for a starting address not on a fullword boundry. Validity check passes a non-zero condition code (in the PSW) to the caller.	IEAVEVAL	IEA0VL01
2 Validity check obtains a protection key from one of three sources: ● Passed as input by the caller or ● From a TCB (TCBPKR) whose address is passed as input by the caller or ● From the current TCB (via PSATOLD).		IEA0VL00

Extended Description	Module	Label
3 Validity check uses the selected protect key to test the storage key of the specified page(s) via the TPROT instruction. If both storing and fetching is allowed on all pages within the selected range, validity check will return to the caller with a condition code of zero. If either storing or fetching is not allowed on any page, a non-zero return code is returned to the caller.		
4 If the TPROT instruction condition code indicates that a page or segment translation exception was encountered, the following processing is done: ● A recovery routine (FRR) is established to intercept any program checks ● The protection key is switched to the selected input protection key ● A compare and swap instruction is used to validate the storage area. The compare and swap (CS) instruction will do both a fetch and store into the specified address. If the check is successful, validity check loops to check the requested address range, if necessary. A program check error will result if the compare and swap instruction referred to an invalid address, resulting in the recovery routine gaining control via RTM. RTM gives control to the recovery routine at step 6, entry point VLCKFRR.		

Diagram SUP-44. Validity Check Processing (IEAVEVAL) (Part 3 of 4)



5 Set the protect key back to zero, delete recovery and set condition code to zero.

VLCKFRR

6 Check the program check code.

- If the program check code indicates that a protection exception, segment translation or page translation error has occurred
- If any other error occurred, give control to the caller's error routine via RTM

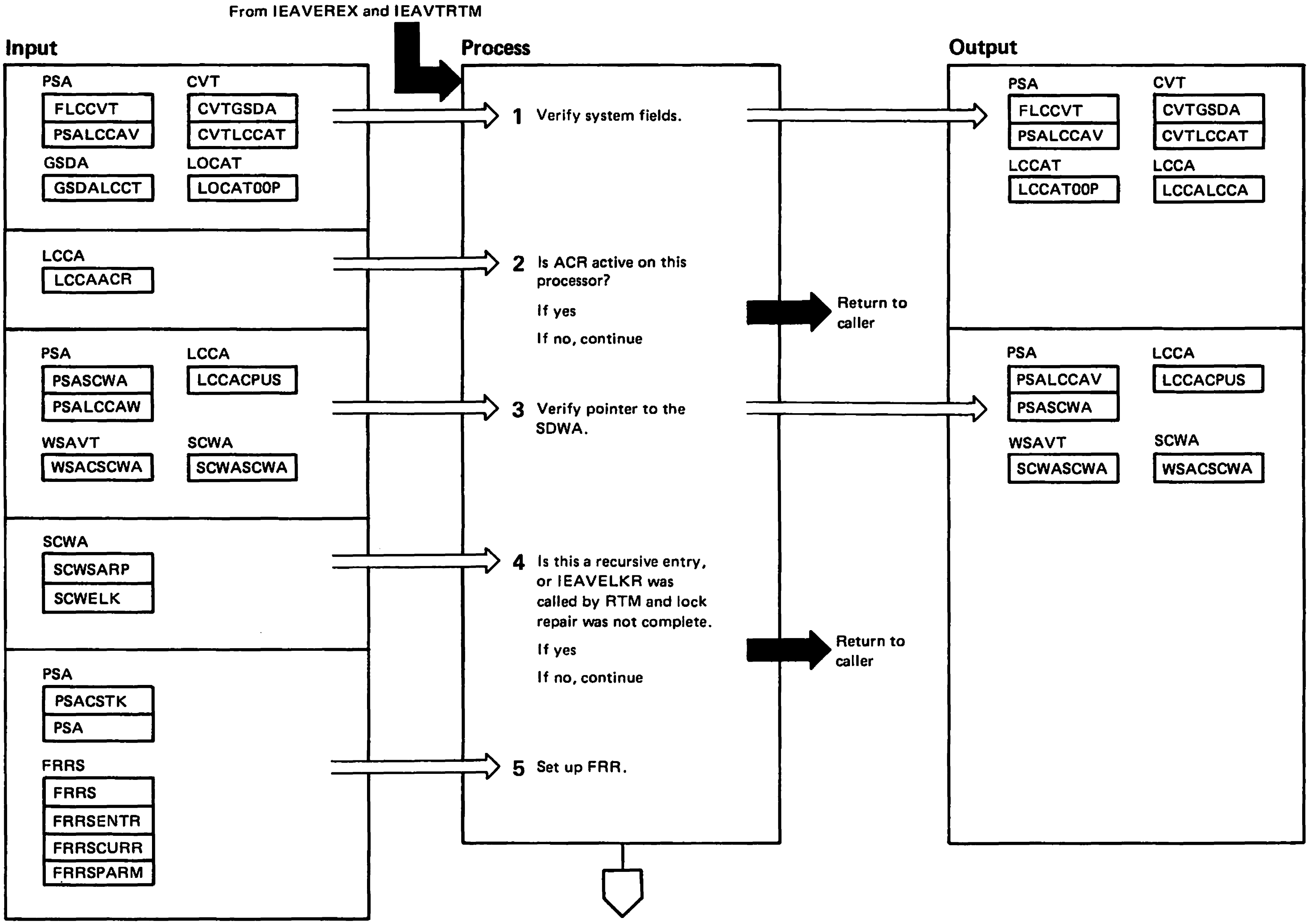
VALRETRY

7 Set a non-zero condition code and delete recovery.

Diagram SUP-44. Validity Check Processing (IEAVEVAL) (Part 4 of 4)

Extended Description	Module	Label
5 Validity check sets a condition code of 0 indicating a valid address. The protection key is changed back to 0, and control returns to the caller.		
6 RTM gives control to validity check at entry point VLCKFRR if a program check occurred. The validity check recovery routine determines whether an expected program check occurred—either a 0C4, 0D0, or 0D1. If one of the three expected error occurred, control goes to RTM to retry at entry point VALRETRY. Otherwise, control goes to RTM to give the caller's error routine control.		VLCKFRR
7 RTM reenters validity check at entry point VALRETRY. Here, validity check sets the condition code to a non-zero, and returns to the caller.		VALRETRY

Diagram SUP-45. Supervisor Analysis Router Routine (IEAVESAR) (Part 1 of 6)



1 Verify system fields.

2 Is ACR active on this processor?

If yes

If no, continue

3 Verify pointer to the SDWA.

4 Is this a recursive entry, or IEAVELKR was called by RTM and lock repair was not complete.

If yes

If no, continue

5 Set up FRR.

Diagram SUP-45. Supervisor Analysis Router Routine (IEAVESAR) (Part 2 of 6)

Extended Description	Module	Label

IEAVESAR refreshes critical control block pointers and calls component analysis routines that continue to refresh/repair process.

1 IEAVESAR verifies system fields that are required to continue processing in this module. Refreshes the CVT pointer. IEAVESAR repairs the PSALCCAV and verifies the GSDA, LCCT, or the PSALCCAV.

If the PSALCCAV pointer and the LCCATOOP entry for this processor both fail LRA checks, then IEAVESAR sets the processor in an X'083' disabled wait.

Otherwise, if needed, IEAVESAR refreshes the PCCATOOP, the PSALCCAV, and LCCA ID, then returns to the subroutine caller.

2 IEAVESAR checks the status of ACR. If ACR is active on this processor, IEAVESAR returns to the caller. Otherwise, continues at the next step.

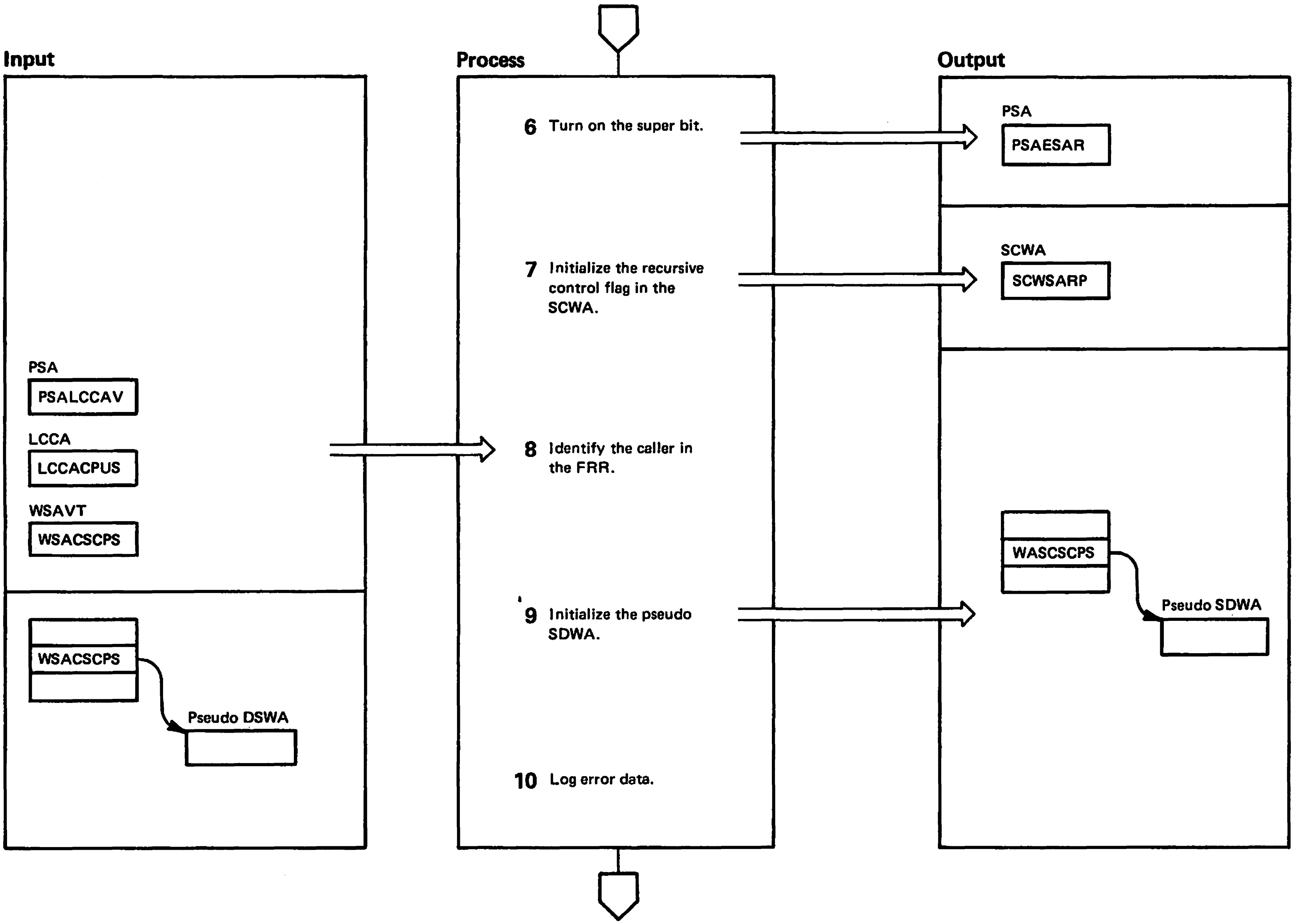
3 IEAVESAR verifies the PSASCWA.

- If both the PSASCWA and the LCCAPUS fail LRA checks, then sets the processor in a X'083' disabled wait.
- If both the PSASCWA and the WSACSCWA fail LRA checks, then sets the processor in a X'083' disabled wait.

4 If this is a recursive entry into IEAVESAR, or IEAVELKR was called directly by RTM, and lock repair processing returns to the caller.

5 IEAVESAR sets up a functional recovery routine (FRR).

Diagram SUP-45. Supervisor Analysis Router Routine (IEAVESAR) (Part 3 of 6)



6 Turn on the super bit.

7 Initialize the recursive control flag in the SCWA.

8 Identify the caller in the FRR.

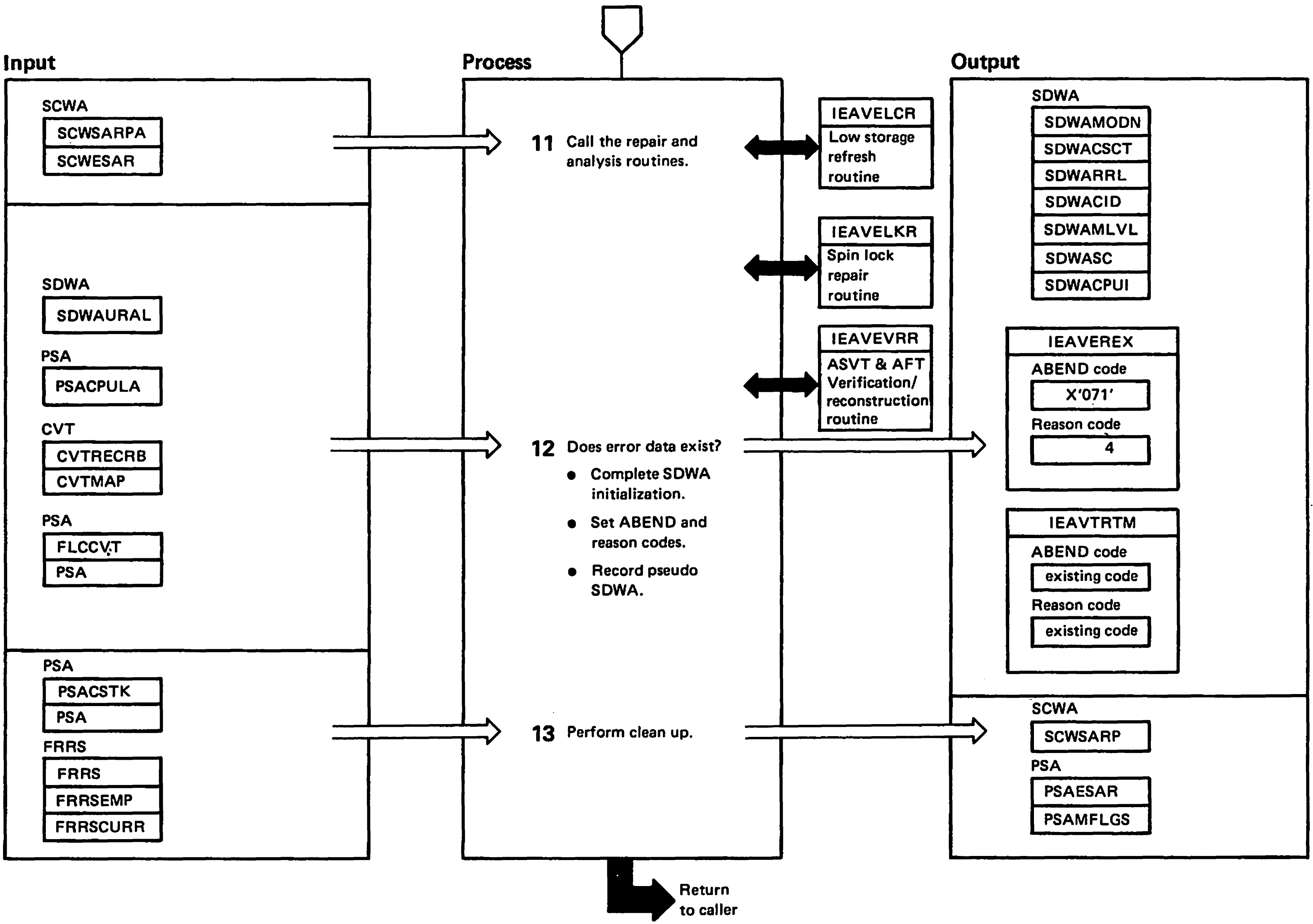
9 Initialize the pseudo SDWA.

10 Log error data.

Diagram SUP-45. Supervisor Analysis Router Routine (IEAVESAR) (Part 4 of 6)

Extended Description	Module	Label
6 IEAVESAR turns on the super bit.		
7 IEAVESAR initializes a recursive control flag in the SCWA.		
8 IEAVESAR identifies the caller in the FRR parameter area.		
9 IEAVESAR initializes the pseudo SDWA in the processor-related work area.		
10 If error data was collected prior to the initialization of the pseudo SDWA, IEAVESAR moves error data, with control block name and field name, the VRA of the pseudo SDWA.		

Diagram SUP-45. Supervisor Analysis Router Routine (IEAVESAR) (Part 5 of 6)



Input

SCWA
SCWSARPA
SCWESAR

SDWA
SDWAURAL
PSA
PSACPULA
CVT
CVTRECRB
CVTMAP
PSA
FLCCVT
PSA

PSA
PSACSTK
PSA
FRRS
FRRS
FRRSEMP
FRRSCURR

Process

11 Call the repair and analysis routines.

IEAVELCR
Low storage refresh routine

IEAVELKR
Spin lock repair routine

IEAVEVRR
ASVT & AFT Verification/ reconstruction routine

12 Does error data exist?
- Complete SDWA initialization.
- Set ABEND and reason codes.
- Record pseudo SDWA.

13 Perform clean up.

Return to caller

Output

SDWA
SDWAMODN
SDWACSCT
SDWARRL
SDWACID
SDWAMLVL
SDWASC
SDWACPUI

IEAVEREX
ABEND code
X'071'
Reason code
4

IEAVTRTM
ABEND code
existing code
Reason code
existing code

SCWA
SCWSARP
PSA
PSAESAR
PSAMFLGS

Diagram SUP-45.. Supervisor Analysis Router Routine (IEAVESAR) (Part 6 of 6)

Extended Description	Module	Label
11 IEAVESAR initializes the necessary parameters and registers, calls the repair/analysis routines. IEAVESAR calls the low storage refresh routine, the spin lock repair lock routine, and the ASVT and AFT verification/ reconstruction routine.	IEAVELCR IEAVELKR IEAVEVRR	

12 If error data exists in the pseudo SDWA, then IEAVESAR:

- Completes the necessary SDWA initialization.
- Sets the ABEND and reason codes.
  - If the caller is IEAVEREX then the ABEND code is X'071' and the reason code is 4.
  - If the caller is IEAVTRTM then the existing ABEND and reason codes are propagated.
- Records the pseudo SDWA in SYS1.LOGREC using the RECORD function.

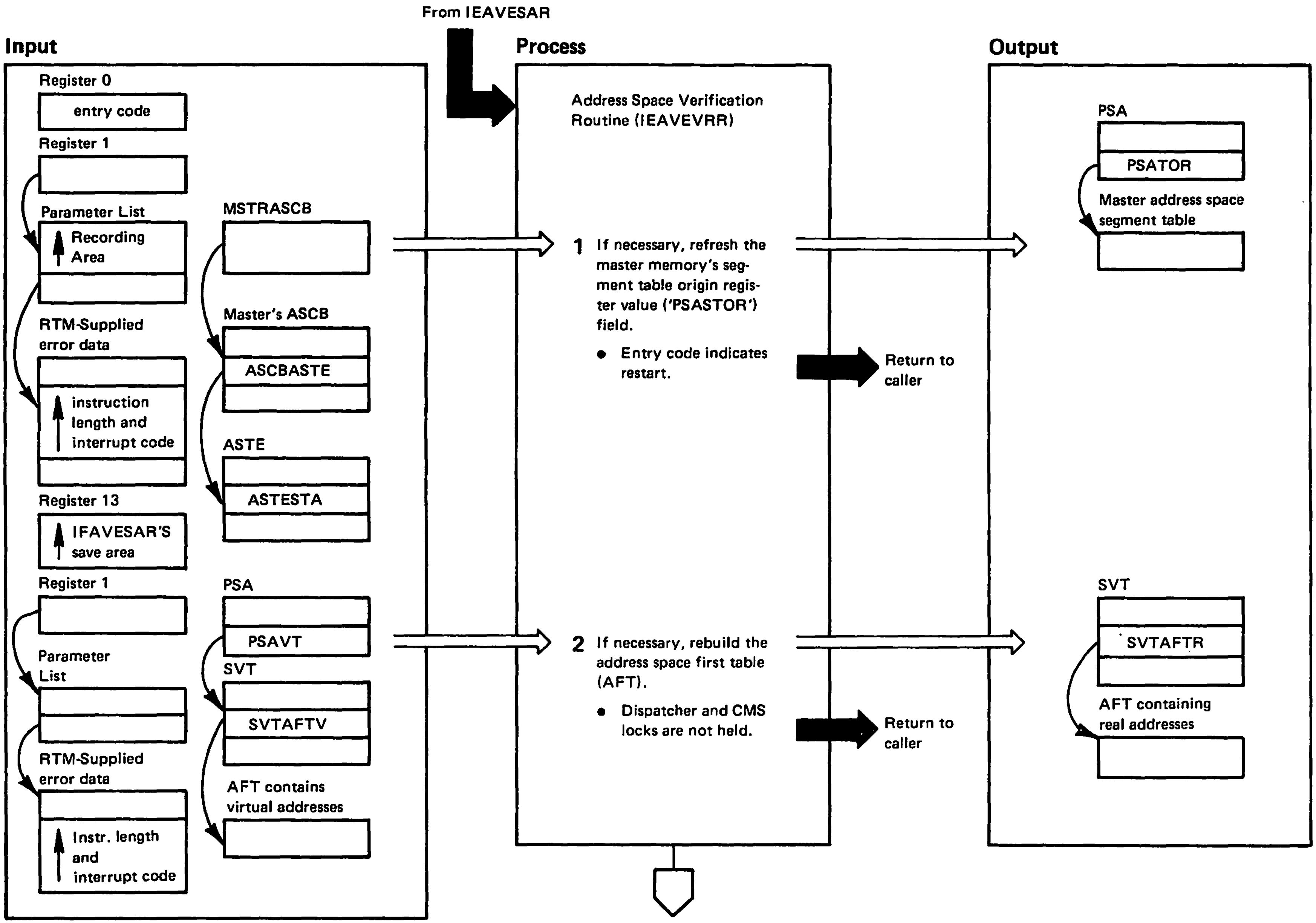
13 IEAVESAR performs necessary clean up prior to exiting. It resets the mainline recursion control, resets the IEAVESAR super bit, and deletes recovery. IEAVESAR then returns to the caller.

Recovery Processing

The IEAVESAR recovery routine (ESARFRR) performs the following functions:

- Resets the IEAVESAR super bit.
- Resets the recursion control (SCWSARP) to allow IEAVESAR to process the next call.
- Indicates in the SDWA, the module in control at the time of the error.
- Indicates in the VRA that an unexpected error occurred.
- Moves the 6-word FRR parameter area to the VRA.
- Indicates percolation to RTM via SETRP.
- Returns to RTM.

Diagram SUP-46. Address Space Verification Processing (IEAVEVRR) (Part 1 of 4)



Address Space Verification Routine (IEAVEVRR)

1 If necessary, refresh the master memory's segment table origin register value ('PSASTOR') field.

- Entry code indicates restart.

2 If necessary, rebuild the address space first table (AFT).

- Dispatcher and CMS locks are not held.

Diagram SUP-46. Address Space Verification Processing (IEAVEVRR) (Part 2 of 4)

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

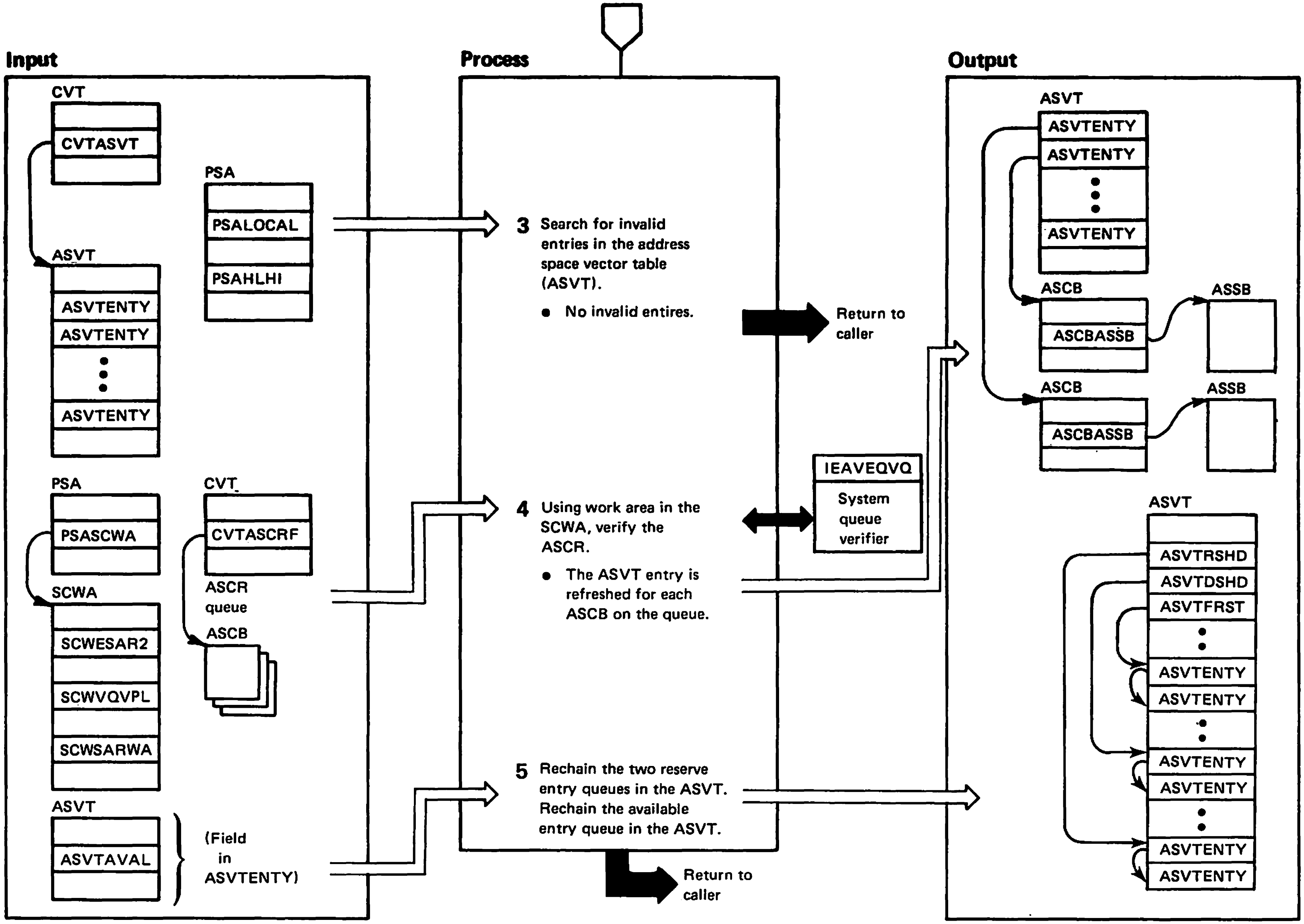
Address space verification processing routine.

1 If necessary, refreshes the master memory's segment table origin register value ('PSASTOR') field from the master address space second table entry (ASTE).

2 IEAVEVRR rebuilds the real AFT from the virtual AFT if the following is true:

- the entry is from RTM via IEAVESAR, and
- the error was a special operation exception, and
- an address first table (AFT) exception, or an address second table exception.

Diagram SUP-46. Address Space Verification Processing (IEAVEVRR) (Part 3 of 4)



3 Search for invalid entries in the address space vector table (ASVT).

- No invalid entires.

4 Using work area in the SCWA, verify the ASCR.

- The ASVT entry is refreshed for each ASCB on the queue.

5 Rechain the two reserve entry queues in the ASVT. Rechain the available entry queue in the ASVT.

Diagram SUP-46. Address Space Verification Processing (IEAVEVRR) (Part 4 of 4)

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

3 If entry is from RTM via IEAVESAR and the CMS and DISPATCHER locks are held, IEAVEVRR searches the address space vector table (ASVT) for invalid entires.

- Unassigned entries should contain an address within the table.
- Unassigned entries should contain the virtual address of a valid ASCB.

If no invalid entries are found, IEAVEVRR returns to its caller.

4 IEAVEVRR calls the system queue verifier IEAVEQV3) to chain through the address space created queue (ASCR). IEAVEVRR reconstructs all ASVT entries that are assigned to ASCBs.

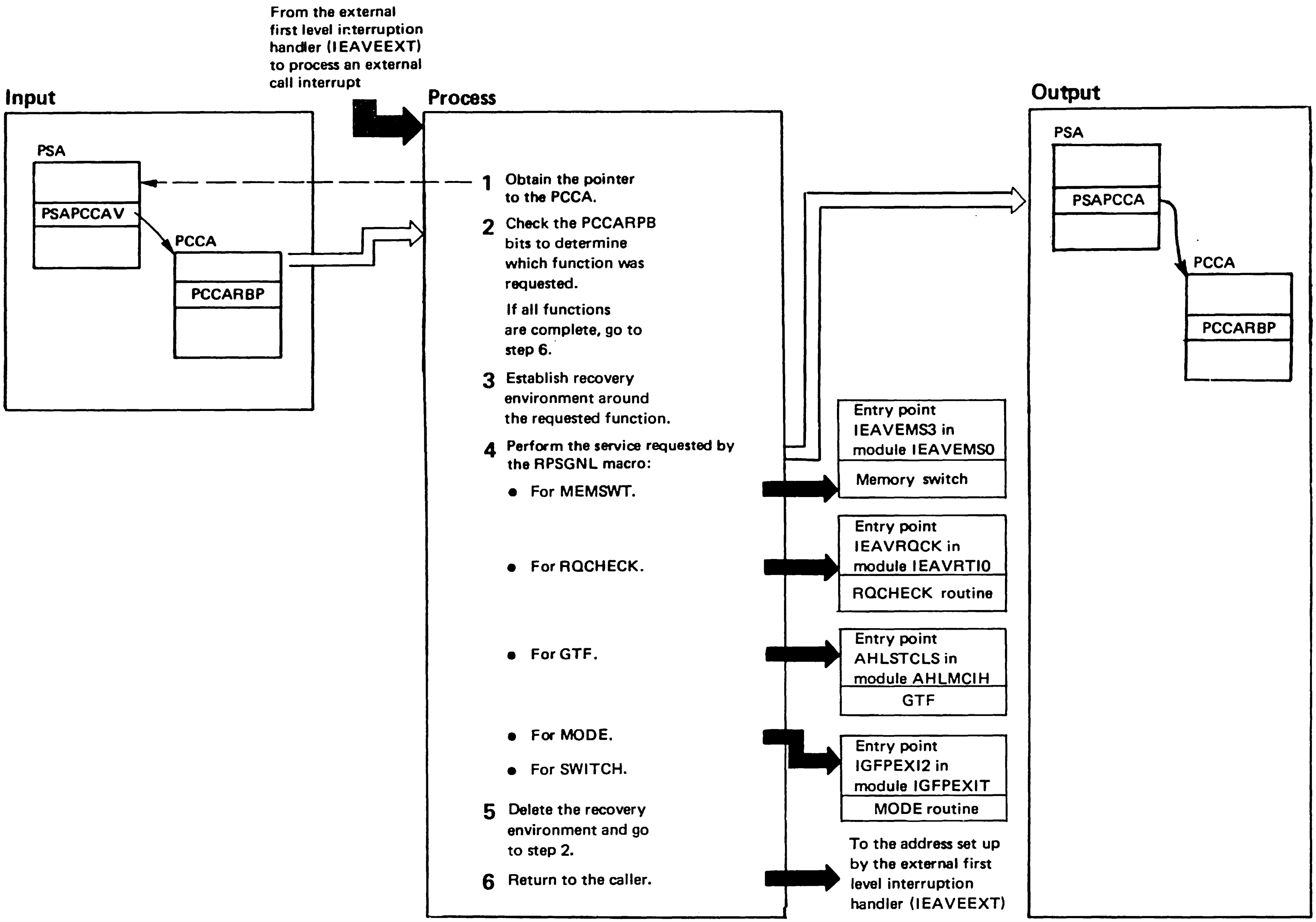
5 Working from the bottom of the ASVT to the top, available entries are chained into three queues:

- Available entries reserved for STARTed/SASI address spaces with the field ASVTRSHD as queue header.
- Available entries for max users, less the number of active address spaces, using the field ASVTFRST as queue header.
- All remaining entries used for replacements for non-reusable ASIDs with the field ASVTDSHD as queue header.

These queues are embedded within the ASVT. The queue header points to the first available ASVTENTY field, and so on. The last available entry contains zeros.

IEAVEVRR returns to the caller.

Diagram SUP-47. External Call Second Level Interruption Handler (IEAVEXS) (Part 1 of 4)


From the external first level interruption handler (IEAVEEXT) to process an external call interrupt
Input
PSA
PSAPCCAV
PCCA
PCCARBP
Process
Output
PSA
PSAPCCA
PCCA
PCCARBP
Entry point IEAVEMS3 in module IEAVEMS0
Memory switch
Entry point IEAVRQCK in module IEAVRTI0
RQCHECK routine
Entry point AHLSTCLS in module AHLMCIH
GTF
Entry point IGFPEXI2 in module IGFPEXIT
MODE routine
To the address set up by the external first level interruption handler (IEAVEEXT)

1 Obtain the pointer to the PCCA.

2 Check the PCCARPB bits to determine which function was requested.

If all functions are complete, go to step 6.

3 Establish recovery environment around the requested function.

4 Perform the service requested by the RPSGNL macro:

- For MEMSWT.
- For RQCHECK.
- For GTF.
- For MODE.
- For SWITCH.

5 Delete the recovery environment and go to step 2.

6 Return to the caller.

Diagram SUP-47. External Call Second Level Interruption Handler (IEAVEXS) (Part 2 of 4)

Extended Description	Module	Label
When an external call interruption occurs, the external FLIH (IEAVEEXT) gives control to the external call second level interrupt handler (SLIH). The SLIH routes control to any of four system routines. The system routine to receive control is determined by the issuer of the RPSGNL macro (remote pendable signal routine IEAVERP). The option specified on the macro corresponds to the following action:		
● MEMSWT - memory switch routine (IEAVEMS3) ● RQCHECK - TOD clock synchronization (IEAVRQCK) ● GTF - GTF function (AHLSTCLS) ● MODE - Mode function (IGFPEXI2) ● SWITCH - No system routine is given control; this option is used to preempt work currently on specified processor.		
Control returns to the external call SLIH; the SLIH returns control to the point established by the external FLIH.		
1 The external call SLIH locates the PCCA (physical control communications area) of the executing processor by referring to the PSA (prefixed storage area).	IEAVEXS	IEAVEXS

Extended Description	Module	Label
2 The PCCA contains an indicator in the external call SIGP buffer (PCCARPB) that specifies the service requested in the RPSGNL function. The external call SLIH checks the PCCARPB field for each possible condition to determine which services are to receive control. In each case, the external SLIH: ● Determines the actions requested in the RPSGNL function. ● Turns off the indicator in the PCCARPB.		
3 The external call SLIH establishes the recovery environment to handle errors in the receiving routine.		
4 The external call SLIH branches to the appropriate service routine except in the case of the SWITCH option. Since no further service routine processing is required for the SWITCH option the SLIH will return to the dispatcher if in task mode. The dispatcher will save the status of the interrupted task thus completing the function requested by the SWITCH option. If not in task mode, the SLIH will return to the interrupted routine.		
5 The SLIH deletes the recovery environment.		RETRYPT
6 Control returns to the address set up by the external FLIH. The return address is either the dispatcher or an entry point in the external FLIH, depending on the system mode at the time the interruption occurred.		

Diagram SUP-47. External Call Second Level Interruption Handler (IEAVEXS) (Part 3 of 4)

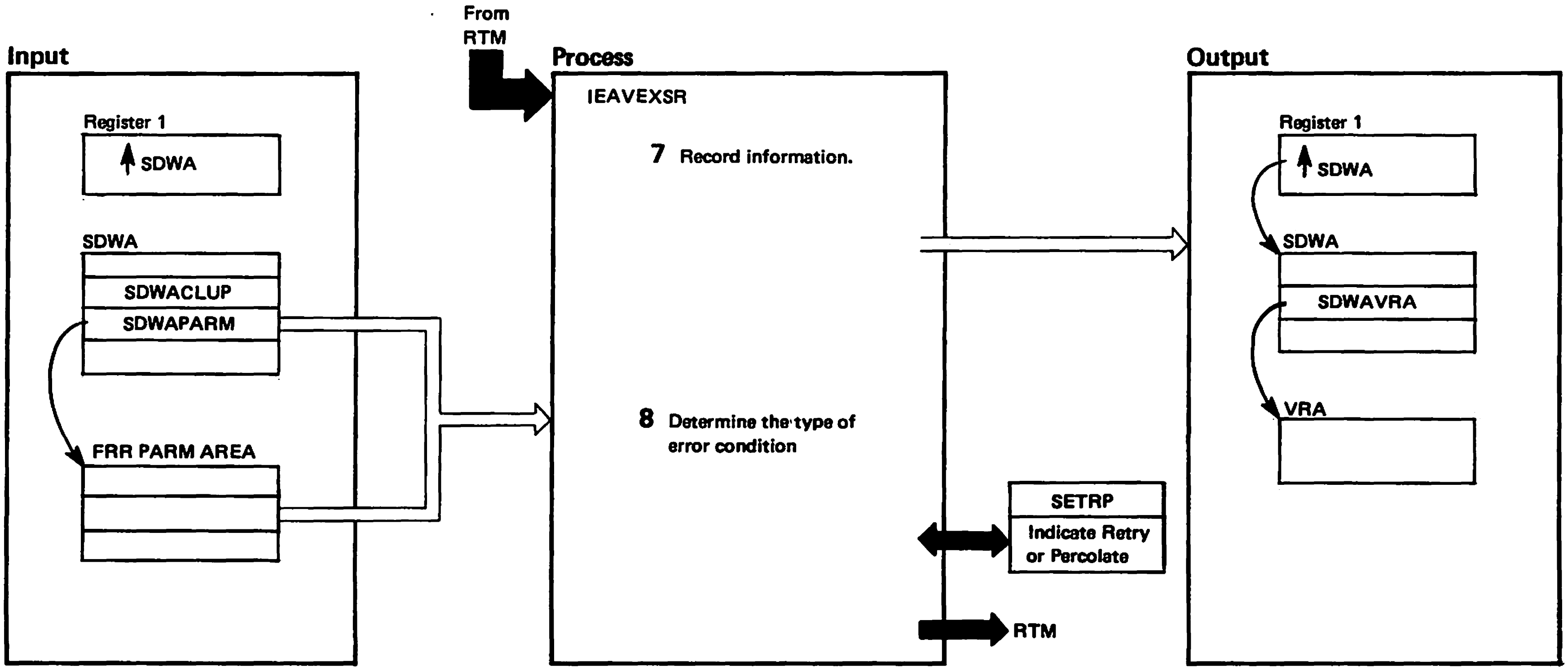
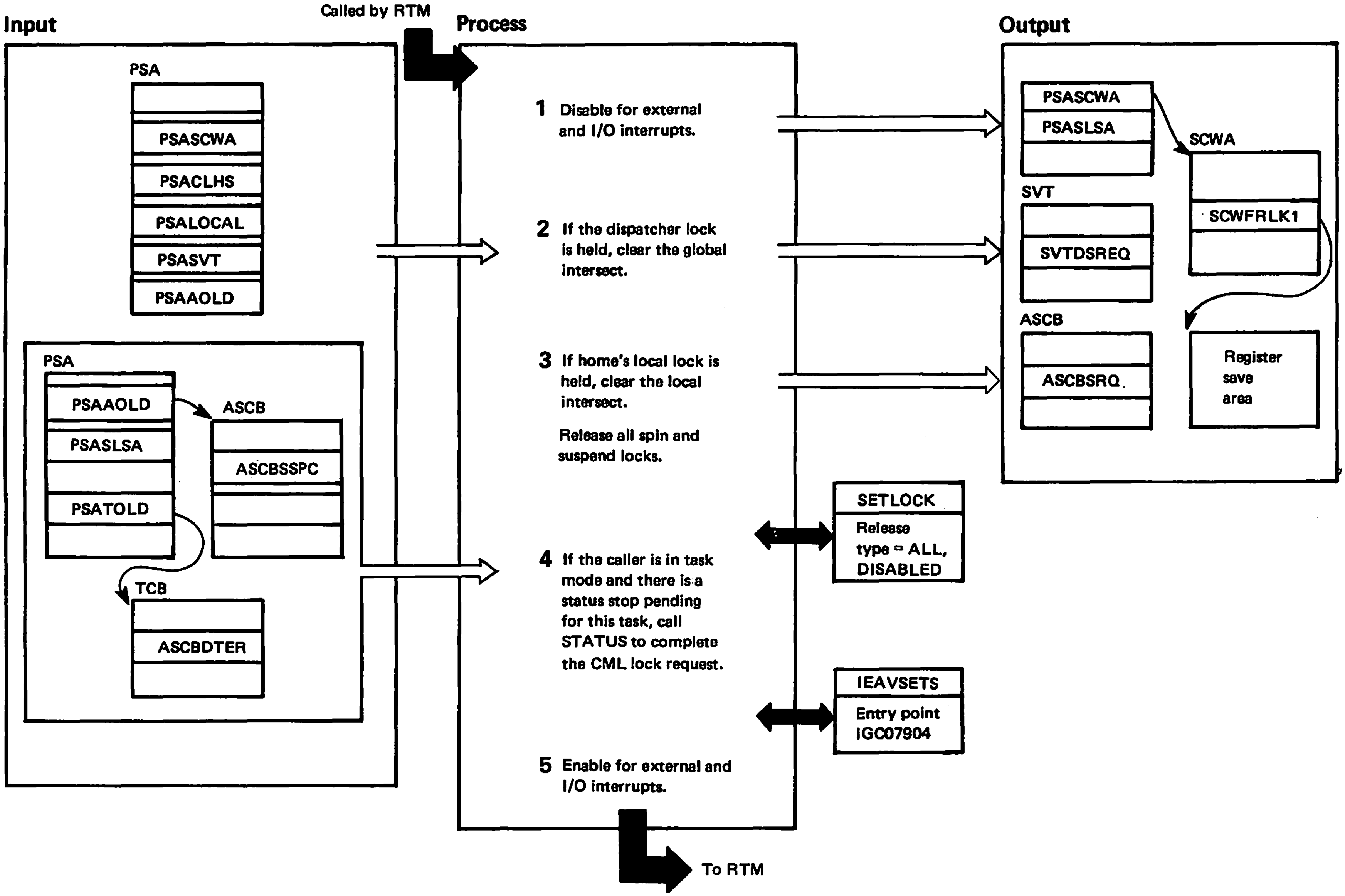


Diagram SUP-47. External Call Second Level Interruption Handler (IEAVEXS) (Part 4 of 4)

Extended Description	Module	Label
7 Indicates error information in the SDWA and VRA areas.		
8 Depending on the reason entered, uses the SETRP macro to indicate to RTM to either retry to the mainline or percolate.		

Diagram SUP-48. Lock Freeing Routine (IEAVFRLK) (Part 1 of 2)



1 Disable for external and I/O interrupts.

2 If the dispatcher lock is held, clear the global intersect.

3 If home's local lock is held, clear the local intersect.

Release all spin and suspend locks.

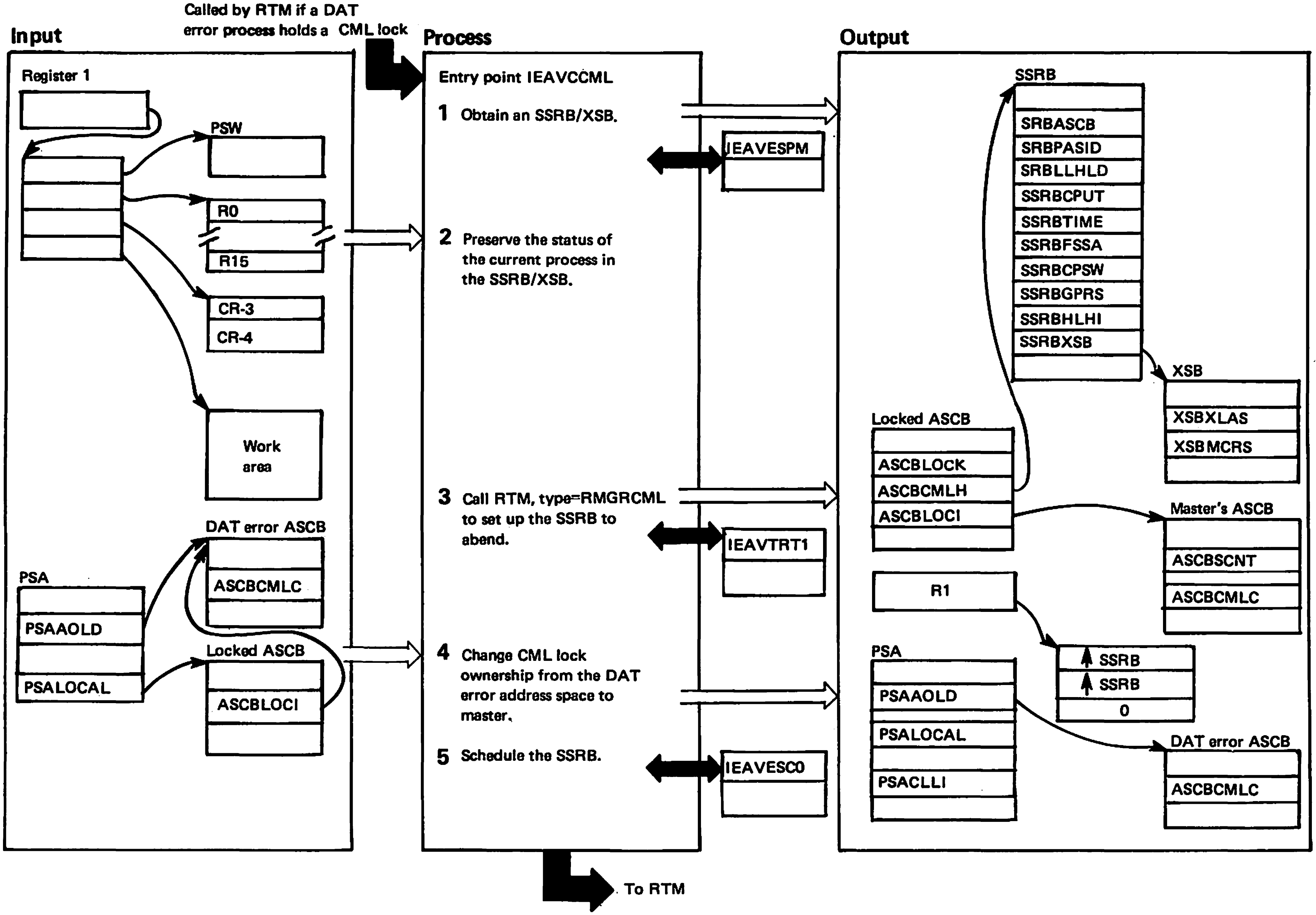
4 If the caller is in task mode and there is a status stop pending for this task, call STATUS to complete the CML lock request.

5 Enable for external and I/O interrupts.

Diagram SUP-48. Lock Freeing Routine (IEAVFRLK) (Part 2 of 2)

Extended Description	Module	Label
IEAVFRLK is called by RTM to clear the global and local intersects, and to free all locks held by the current process.		
1 IEAVFRLK disables for external and I/O interrupts. It then saves the caller's registers 1-14 in the SCWA.		
2 If the dispatcher lock is held, the global intersect is cleared by setting the SVTDSREQ word to all zeros.		
3 If the local lock is held, the local intersect is cleared by setting the ASCBSRQ word in home's ASCB to all zeros.		
4 If the caller is in task mode and the status stop pending flag (ASCBSSPC) is on and the DAT error flag (ASCBDTER) is off, then secondary addressability is established to the home address space. If the status stop pending flag (TCBSSPC) is on for the current task then the status routine is called to complete the pending status request. The caller's secondary addressability is restored.		
5 The caller's registers 1-14 are restored from the SCWA and external and I/O interrupts are enabled.		

Diagram SUP-49. CML Lock Cleanup for Current DAT ERROR Process (IEAVLKRM) (Part 1 of 2)



Entry point IEAVCCML

1 Obtain an SSRB/XSB.

2 Preserve the status of the current process in the SSRB/XSB.

3 Call RTM, type=RMGRCML to set up the SSRB to abend.

4 Change CML lock ownership from the DAT error address space to master.

5 Schedule the SSRB.

Diagram SUP-49. CML Lock Cleanup for Current DAT ERROR Process (IEAVLKRM) (Part 2 of 2)

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

If a process has suffered a DAT error while holding the CML lock of another address space, RTM calls IEAVLKRM at entry point IEAVCCML.

1 Obtain an SSRB/XSB.

2 IEAVLKRM initializes the SSRB/XSB to represent the DAT error process with the exception that it is to run in the master's address space. The following fields are initialized in the new SSRB:

- SRBPASID is set to one.
- SRBASCB is set to point to the master's ASCB.
- SRBLLHLD is set to indicate a local lock is held.
- SRBHLHI is set to indicate a local lock is held.
- SSRBCPUT and SSRBTIME fields are both set to 208-day time value.

The following DAT error process status is saved in the SSRB or XSB:

- PSW at the time of error
- GPRs 0 - 15
- Control registers 3 and 4
- The contents of the normal FRR stack
- XSBXLAS is set to point to the locked ASCB.

The PCLINK stack header (RSASTKE) is set to zero to prevent any accidental reference to the DAT address space private storage via the stack chain.

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

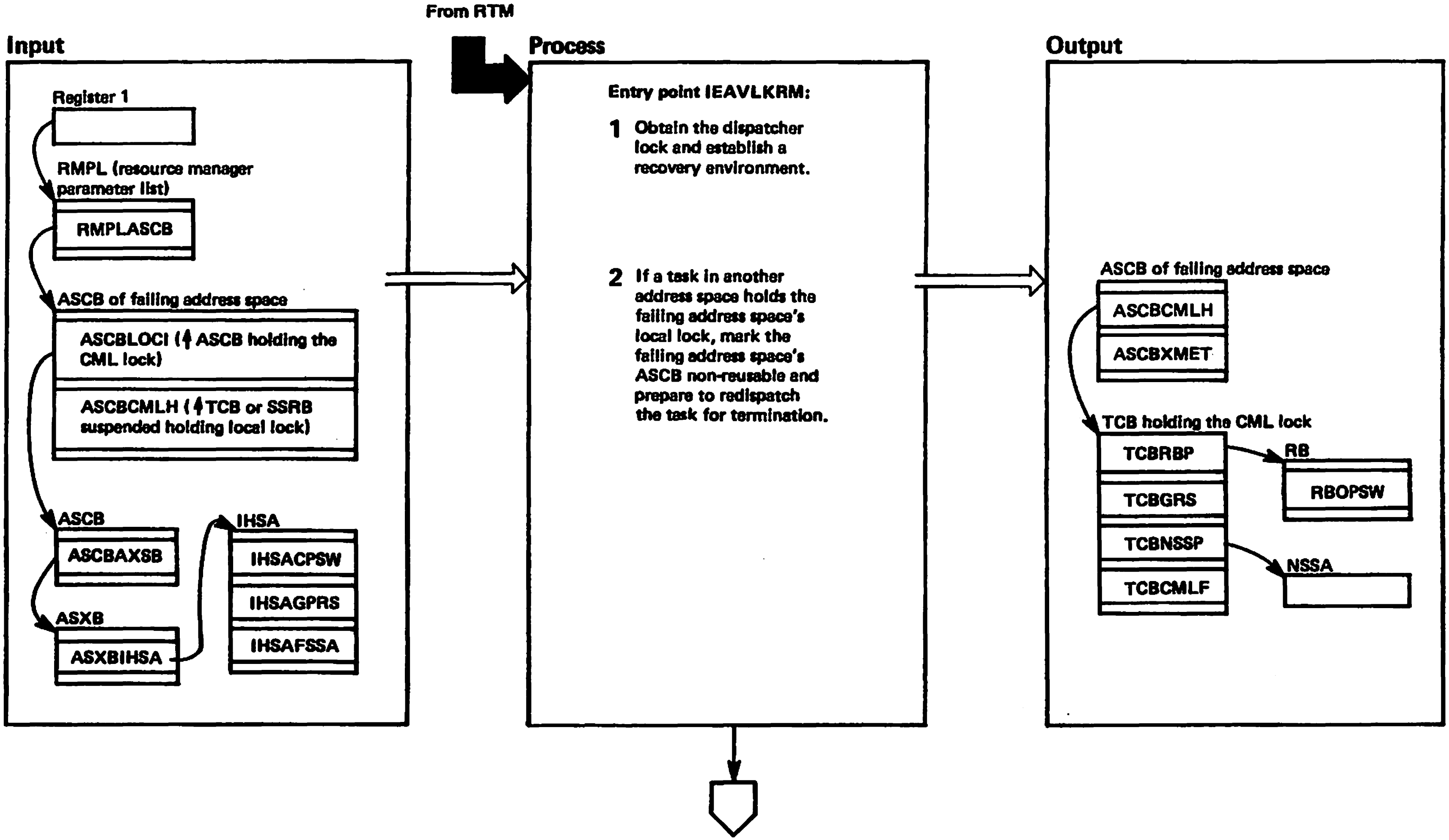
3 IEAVLKRM calls RTM, type=RMGRCML to set up the SSRB to abend.

4 The process of changing the CML lock ownership from the DAT error address space to appear to be owned by a suspended unit of work in the master's address space consists of the following steps:

- PSALOCAL is set to zero.
- PSALCLLI (local lock held flag in the CLHS) is turned off.
- In the locked address space:
  - ASCBLOCK is set to '4FFFFFFF' (ready-to-run ID)
  - ASCBLOCI is set to point to the master's ASCB
  - ASCBCMLH is set to point to the new SSRB
- In the DAT error address space:
  - The count of CML locks held (ASCBCMLC) is decreased by one.
- In the master's address space:
  - The suspended SRB count (ASCBSCNT) is increased by one.
  - The count of CML locks held (ASCBMLC) is increased by one.
- XSBXMCRS in the new XSB is set equal to the master's control register 3 and 4.

5 When this process is dispatched, it will abend with an X'064' completion code and RTM will attempt to give control to any FRRs on the normal stack at the time of the DAT error so that the FRRs can release resources, including the CML lock.

Diagram SUP-50. CML and LOCAL Lock Resource Managers (IEAVLKRM and IEAVELRM) (Part 1 of 6)


From RTM
Input
Register 1
RMPL (resource manager parameter list)
RMPLASCB
ASCB of failing address space
ASCBLOCI (↑ASCB holding the CML lock)
ASCBCMLH (↑TCB or SSRB suspended holding local lock)
ASCB
ASCBAXSB
ASXB
ASXBIHSA
IHSA
IHSACPSW
IHSAGPRS
IHSAFSSA
Process
Output
ASCB of failing address space
ASCBCMLH
ASCBXMET
TCB holding the CML lock
TCBRBP
TCBGRS
TCBNSSP
TCBCMLF
RB
RBOPSW
NSSA

Entry point IEAVLKRM:

1 Obtain the dispatcher lock and establish a recovery environment.

2 If a task in another address space holds the failing address space's local lock, mark the failing address space's ASCB non-reusable and prepare to redispatch the task for termination.

Diagram SUP-50. CML and LOCAL Lock Resource Managers (IEAVLKRM and IEAVELRM) (Part 2 of 6)

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

This diagram describes both the CML resource manager (IEAVLKRM) and the LOCAL lock resource manager (entry point IEAVELRM within IEAVLKRM). IEAVLKRM is described in steps 1-4, IEAVELRM in step 5.

CML Resource Manager (IEAVLKRM) Processing

RTM gives control to IEAVLKRM early in address space termination processing. IEAVLKRM ensures that any CML locks owned by the terminating address space are later released. Also, if the failing address space's CML lock is held, IEAVLKRM ensures that the unit of work holding the lock is abended. At entry, register 1 contains the address of a pointer to the resource manager parameter list (RMPL).

1 IEAVLKRM obtains the dispatcher lock to serialize the ASVT scan described in step 3. It issues a SETFRR macro to establish LKRMFRR (an entry point within IEAVLKRM) as its recovery routine. (See "Recovery Processing" for a description of LKRMFRR.)

2 If no address space owns the failing address space's local lock as a CML lock, IEAVLKRM continues at the next step. If another address space does own the failing address space's CML lock, IEAVLKRM sets the non-reusability flag (ASCBXMET) in the failing address space's ASCB. This prevents any suspended SRBs from being rescheduled to the wrong address space by preventing the reuse of this ASID.

If an SSRB holds the failing address space's CML lock, no other processing is required. When the SSRB is made dispatchable and the dispatcher finds that the CML-locked address space cannot be accessed, it terminates the unit of work represented by the SSRB.

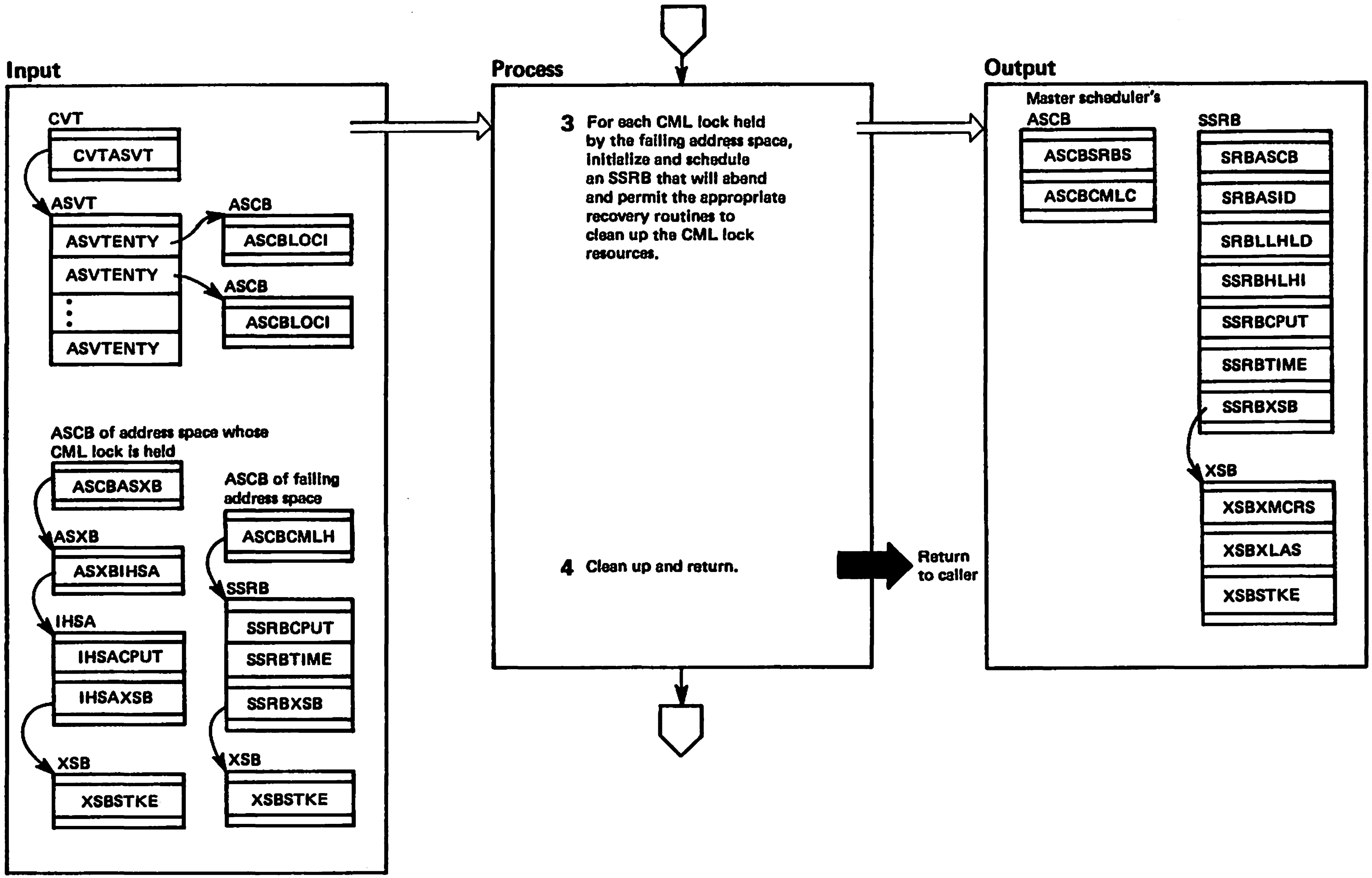
Extended Description **Module** **Label**

If a task holds the CML lock, IEAVLKRM prepares the task to be dispatched. It:

- Checks for a DAT error in the terminating address space. If there is an error, the task holding the CML lock is reset by calling the status subroutine.
- Copies the registers and PSW from the task's IHSA into the TCB/RB.
- Copies the FRR stack from the IHSA into the TCB's NSSA.
- Sets the TCBCMLF flag, which indicates to the dispatcher that the CML resource manager has put the appropriate status into the TCB/RB and NSSA. (The dispatcher does not redispatch the task until this flag is set.)
- If status pending is on for this task (TCBSSPC = ON), resets the task by calling the status subroutine at entry point IGC07904 in module IEAVSETS.

When the dispatcher attempts to dispatch the task and finds that the task cannot establish addressability to the CML-locked address space, it abends the task. RTM then gets control to process the FRR stack saved in the NSSA. This permits the recovery routines to clean up any resources serialized by the lock and to release the lock.

Diagram SUP-50. CML and LOCAL Lock Resource Managers (IEAVELKRM and IEAVELRM) (Part 3 of 6)



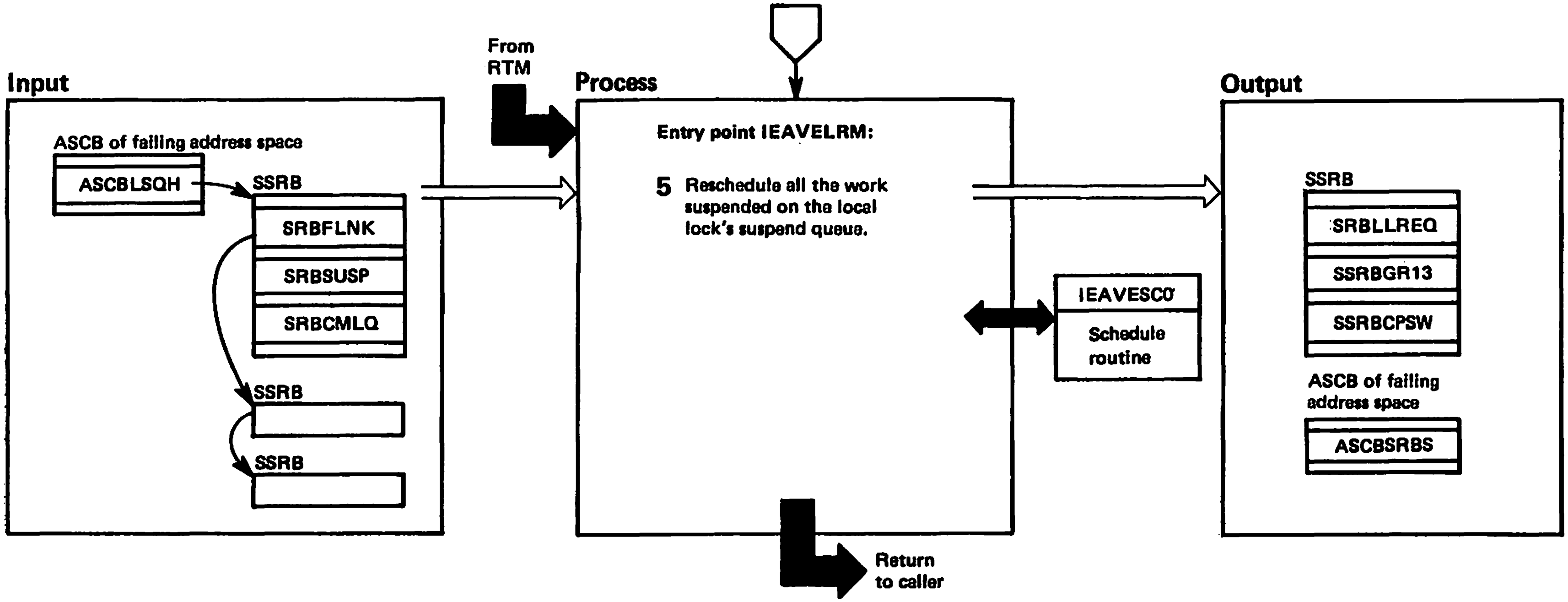
3 For each CML lock held by the failing address space, initialize and schedule an SSRB that will abend and permit the appropriate recovery routines to clean up the CML lock resources.

4 Clean up and return.

Diagram SUP-50. CML and LOCAL Lock Resource Managers (IEAVLKRM and IEAVELRM) (Part 4 of 6)

Extended Description	Module	Label
3 If the failing address space owns any CML locks (the ASCBCMLC field is greater than zero), IEAVLKRM scans the ASVT for the address spaces whose CML locks it holds (the address spaces whose ASCBLOCI values match the address of the failing address space's ASCB). For each such address space found, IEAVLKRM: • Issues a GETSSRB macro to obtain an SSRB. • Initializes the SSRB to run in the master scheduler's address space by setting the SRBPASID and SRBASCB fields to reflect the master scheduler's address space. • Copies the processor timer values and PCLINK stack header into the new SSRB/XSB. IEAVLKRM takes these values from either the old SSRB/XSB or the owning task's IHSA. • Issues a CALLRTM macro to have RTM copy the lock holder's recovery information into the SSRB, and to set the SSRB to abend when dispatched. This allows the appropriate FRRs to get control after the SSRB abends. • Puts the master scheduler's control register values into the new SSRB's control register fields. • Sets the SRBLLHLD and SRBHLHI fields to make it appear as though the SSRB holds the CML lock. • Puts the ASCB address of the master scheduler's ASCB into the ASCBLOCI field, and indicates that an SSRB holds the CML lock. • Adds one to the master scheduler's count of suspended SRBs (ASCBSRBS). • Adds one to the master scheduler's count of CML locks held (ASCBCMLC). • Schedules the SSRB to execute in the master scheduler's address space. When dispatched, the SSRB abends. RTM gets control to process the FRR stack associated with the CML lock. These FRR routines clean up any resources serialized by the lock and release the lock.	IEAVLKRM	LKRMCMLS
4 IEAVLKRM releases the dispatcher lock, deletes the FRR, and returns to RTM.	IEAVLKRM	LKRMXIT

Diagram SUP-50. CML and LOCAL Lock Resource Managers (IEAVLKRM and IEAVELRM) (Part 5 of 6)



Entry point IEAVELRM:

5 Reschedule all the work suspended on the local lock's suspend queue.

Diagram SUP-50. CML and LOCAL Lock Resource Managers (IEAVLKRM and IEAVELRM) (Part 6 of 6)

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

LOCAL Lock Resource Manager (IEAVELRM) Processing

5 RTM gives control to IEAVELRM near the end of address space termination processing. IEAVELRM cleans up the terminating address space's local lock suspend queue. It puts the appropriate information in the queued SRB/SSRB's to make them appear valid and reschedules them. This causes the dispatcher to put the SRB/SSRBs back on the local or global service priority list (SPL). However, because the failing address space's ASCBFAIL flag is on, the dispatcher cannot allow the SRB/SSRBs to run. They remain on the dispatching queue until PURGEDQ finds them and calls their respective resource termination managers.

IEAVELRM then returns to RTM.

Recovery Processing

When an error occurs while IEAVLKRM is executing, RTM gives control to LKRMFRR. LKRMFRR requests that RTM free the dispatcher lock, records diagnostic information in the SDWA and, if necessary, frees the SSRB that IEAVLKRM obtained. LKRMFRR then returns to RTM.

Diagram SUP-51. DATOFF Macro Service Routine (IEAVMVC0) (Part 1 of 4)

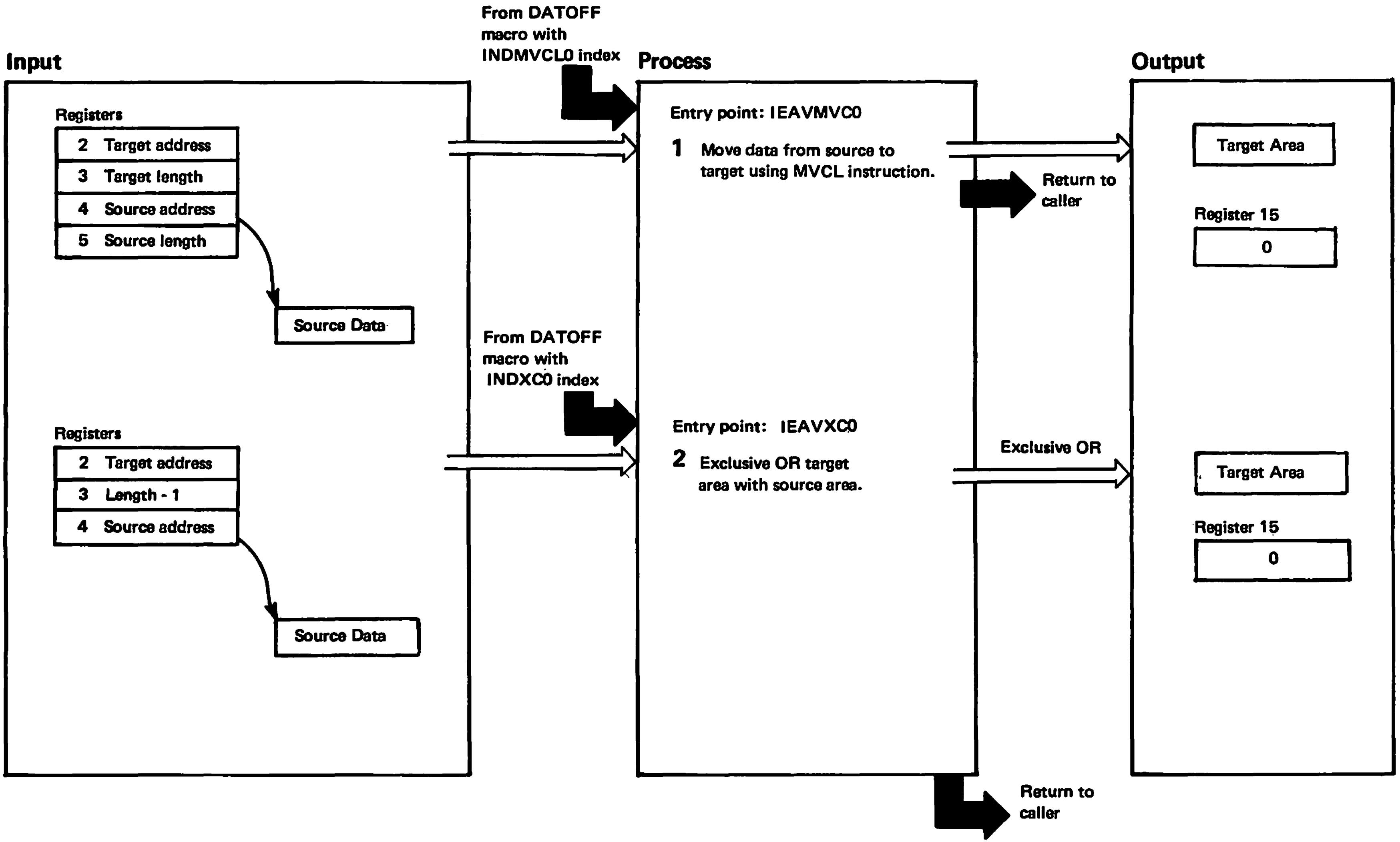
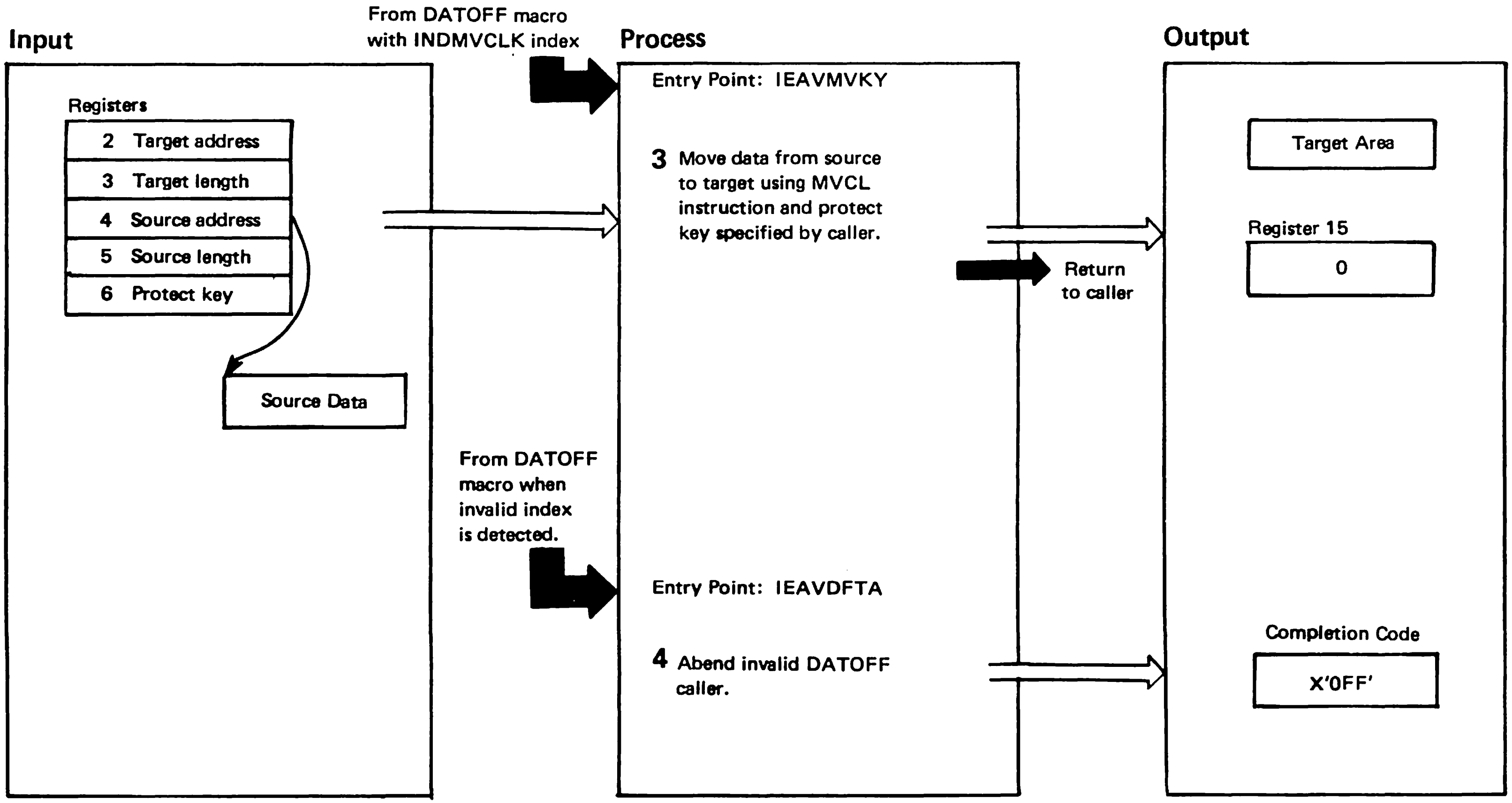


Diagram SUP-51. DATOFF Macro Service Routine (IEAVMVC0) (Part 2 of 4)

Extended Description	Module	Label
IEAVMVC0 contains several generalized IBM-supplied DATOFF routines. Access to the routines is through the DATOFF macro. DATOFF index INDMVCL0:		
1 Moves the data from the source location to the target location specified by the caller using a MVCL instruction. Sets a zero return code.	IEAVMVC0	IEAVMVC0
DATOFF index INDXC0:		
2 Exclusive ORs the data in the target location with the source area data specified by the caller using an XC instruction. Sets a zero return code.	IEAVMVC0	IEAVXC0

Diagram SUP-51. DATOFF Macro Service Routine (IEAVMVC0) (Part 3 of 4)



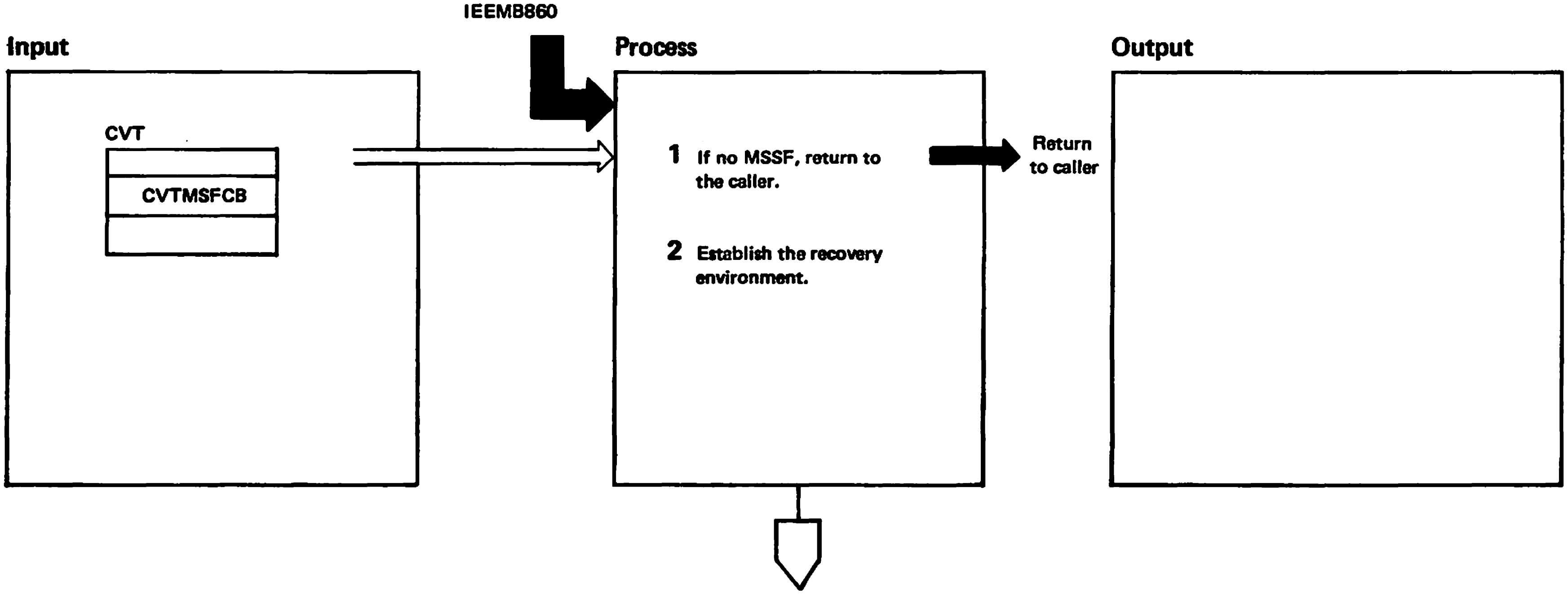
3 Move data from source to target using MVCL instruction and protect key specified by caller.

4 Abend invalid DATOFF caller.

Diagram SUP-51. DATOFF Macro Service Routine (IEAVMVCO) (Part 4 of 4)

Extended Description	Module	Label
DATOFF Index INDMVCLK		
3 Switches to the PSW protect key specified by the caller. Moves the data from the source location to the target location specified by the caller using a MVCL instruction. Returns to protect key zero and sets a zero return code.	IEAVMVCO	IEAVMVKY
DATOFF Index is Invalid		
4 Abends the caller with a X'0FF' completion code. DATOFF detected an invalid index passed to the DATOFF linkage routine.	IEAVMVCO	IEAVDFTA

Diagram SUP-52. Processor Controller Damage Monitor Routine (IEAVSPDM) (Part 1 of 6)



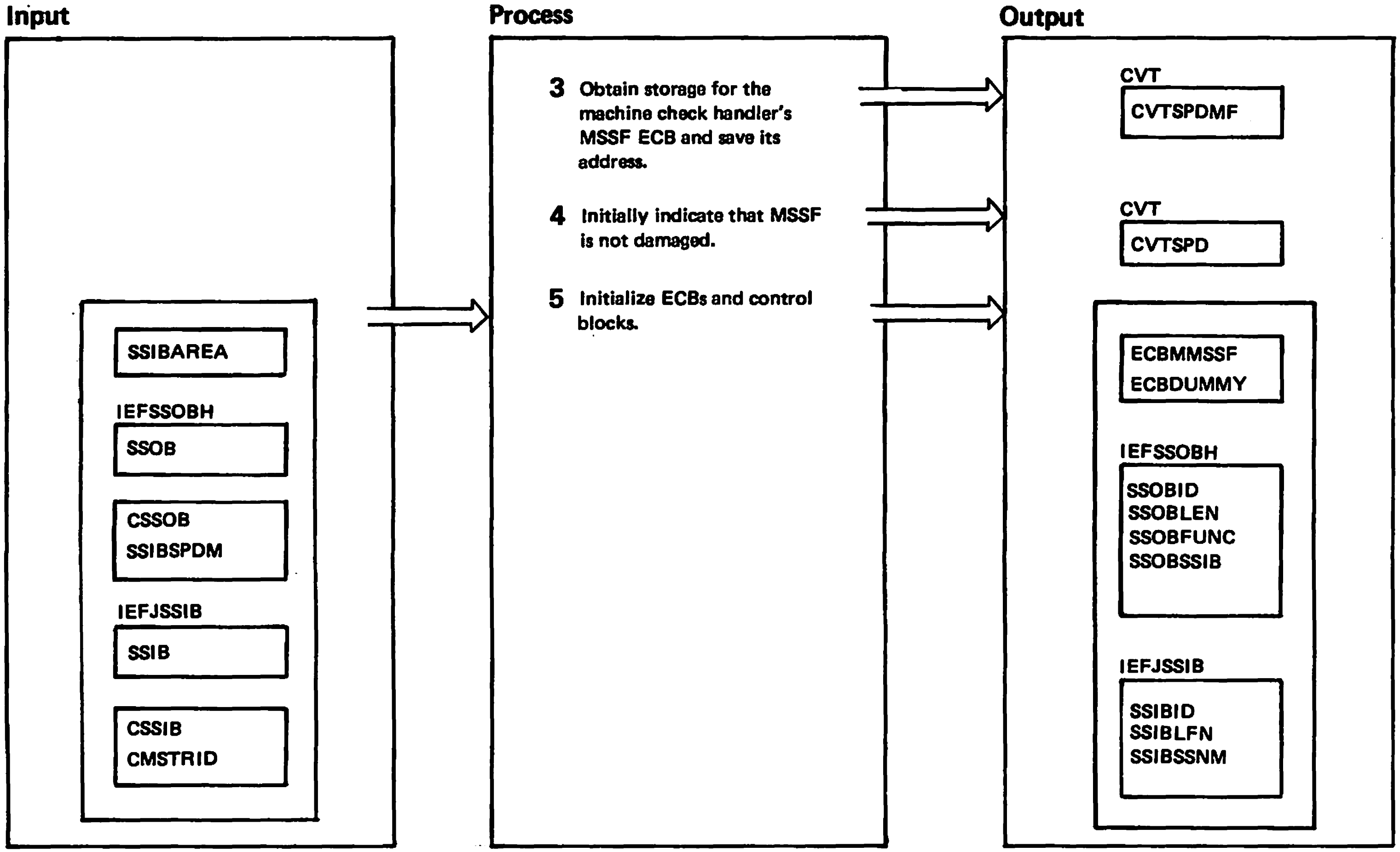
1 If no MSSF, return to the caller.

2 Establish the recovery environment.

Diagram SUP-52. Processor Controller Damage Monitor Routine (IEAVSPDM) (Part 2 of 6)

Extended Description	Module	Label
IEAVSPDM is attached by the master scheduler (IEEMB860) during system initialization and waits on an ECB that will be posted when a processor controller machine check occurs. If the ECB is posted, IEAVSPDM notifies all listening subsystems that the processor controller is damaged via the subsystem interface. IEAVSPDM also issues a message to notify the operator that the processor controller is damaged.		
1 If there is no usable maintenance service and support facility (MSSF) hardware present or if the code is executing on a VM configuration (field CVTMSFCB is zero), return to the master scheduler. Before accessing the CVT, force the reset of cache lines. This prevents this module from bringing down the system if there were cache interference and MSSF was not working at this point.	IEAVSPDM	IEAVSPDM
2 Establish SPDMRECV as the error recovery routine by issuing the ESTAE macro. If ESTAE is not successful, abend the caller with abend code X'050'.		

Diagram SUP-52. Processor Controller Damage Monitor Routine (IEAVSPDM) (Part 3 of 6)



3 Obtain storage for the machine check handler's MSSF ECB and save its address.

4 Initially indicate that MSSF is not damaged.

5 Initialize ECBs and control blocks.

Diagram SUP-52. Processor Controller Damage Monitor Routine (IEAVSPDM) (Part 4 of 6)

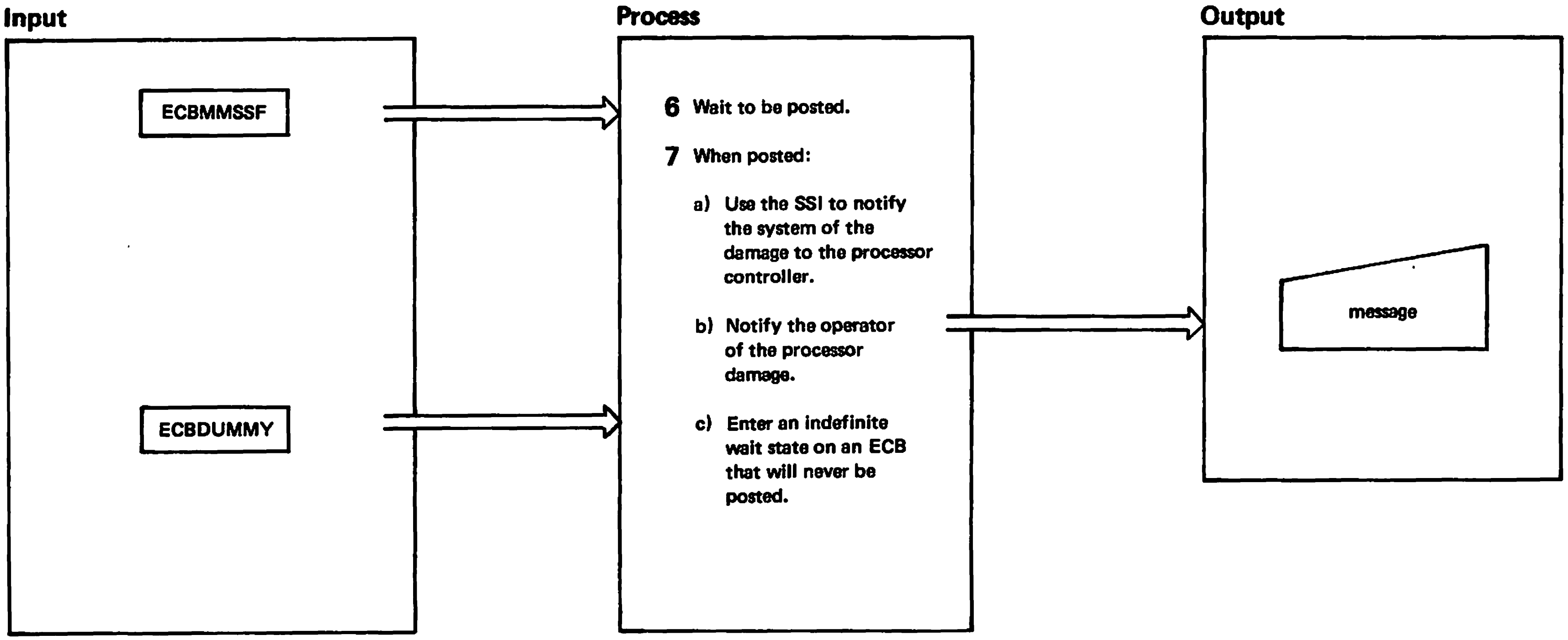
Extended Descriptior

3 Obtain fixed LSQA storage (subpool 255) for the machine check handler's MSSF ECB and place its address in the CVT. Before accessing the CVT, force the reset of cache lines. This prevents this module from bringing down the system if there were cache interference and MSSF was not working at this point.

4 Set a flag to indicate that MSSF is not damaged.

5 Clear the machine check handler's MSSF ECB and clear the dummy ECB to be used for an indefinite wait after the processor controller has been damaged. Initialize the subsystem output block (SSOB) and subsystem initialization block (SSIB).

Diagram SUP-52. Processor Controller Damage Monitor Routine (IEAVSPDM) (Part 5 of 6)



6 Wait to be posted.

7 When posted:

a) Use the SSI to notify the system of the damage to the processor controller.

b) Notify the operator of the processor damage.

c) Enter an indefinite wait state on an ECB that will never be posted.

Diagram SUP-52. Processor Controller Damage Monitor Routine (IEAVSPDM) (Part 6 of 6)

Extended Description	Module	Label
6 Issue WAIT macro to wait until the ECB is posted, either by the machine check handler or alternate CPU recovery.		
7 When the ECB is posted, do the following:		
a) Notify listening subsystems of the damage to the processor controller via the subsystem interface.		MSSFDEAD
b) Issue message IEA470W to notify the operator of the damage to the processor controller.		
c) Enter an indefinite wait state on the dummy ECB that will never be posted. An indefinite wait is entered rather than returning to the caller to preserve the MSSF ECB in case any attempt is made to access it again.		

SUP-53. CHECKPOINT/RESTART EXIT ROUTINE ROUTER (IEAVCKRS)

IEAVCKRS - MODULE DESCRIPTION

DESCRIPTIVE NAME: Checkpoint/Restart Exit Routine Router

FUNCTION:

Checkpoint/Restart passes control to this Supervisor Control module (IEAVCKRS) (via the entry point address in CVTCSPIE) which is responsible for calling the appropriate exit routines to perform Checkpoint/Restart processing.

IEAVCKRS is called by Checkpoint/Restart three times (twice for the Checkpoint and once for the Restart) to invoke the appropriate exit routines. Each time, register one will point to a queue of at least one SSCR (Subsystem Checkpoint Record). The SSCRCKRS and SSCRBKC bits within the first SSCR will be set as follows:

- First call -	SSCRCKRS = 1	SSCRBKC = 1
- Second call -	SSCRCKRS = 1	SSCRBKC = 0
- Third call -	SSCRCKRS = 0	SSCRBKC is unused

The first call is a request to return the number of SSCRs needed to checkpoint the task. The Exit Routines are responsible for returning the number of SSCRs they require. On the second call, Checkpoint/Restart passes a queue of the requested SSCRs to IEAVCKRS. The Exit Routines are responsible for storing the appropriate task related status in the SSCRs. On the third call, Checkpoint/Restart passes the same queue of SSCRs to IEAVCKRS. The exit routines are responsible for restoring the task's status from the SSCRs.

ENTRY POINT: IEAVCKRS

PURPOSE: Refer to the module function topic.

LINKAGE: Branch entered via CVTCSPIE

CALLERS: Checkpoint/Restart

INPUT:

- A queue of one or more SSCRs pointed to by register one upon entry

- The SSCRCKRS and SSCRBKC bits in the first SSCR on the queue set to indicate the function to be performed

OUTPUT:

- All exit routines called to perform the appropriate Checkpoint/Restart processing based on the values of the SSCRCKRS and SSCRBKC bits

- SSCRBLKC set to the total number of SSCRs requested by the exit routines plus one for the SYSTEM SSCR (SSCRBLKC is set only when the SSCRCKRS and SSCRBKC bits are one)

EXIT NORMAL: To Checkpoint/Restart with return code in register 15

EXIT ERROR: To Checkpoint/Restart with return code in register 15

EXTERNAL REFERENCES:

ROUTINES:
IEAVCRVF - (Vector Checkpoint/Restart Exit routine)
IEAVSPI - (ESPIE Checkpoint/Restart Exit routine)

DATA AREAS:
The first SSCR on the queue during Checkpoint processing and Common IEAVCKRS processing will contain SSCR count information related to each of the called exit routines. This SSCR is refered to as the System SSCR.

IEAVCKRS - MODULE DESCRIPTION (Continued)

CONTROL BLOCKS:

SSCR (Subsystem Checkpoint Record)	- (r,w)

r = Read
w = Write

TABLES: EXITTABLE - The internal exit routine name and ID table

SERIALIZATION: No special serialization is required.

SUP-53. Checkpoint/Restart Exit Routine Router (IEAVCKRS)

IEAVCKRS - MODULE OPERATION

IEAVCKRS will LOAD each exit routine module and save the address for later use. The names of the exit routines will be obtained from a static table resident in IEAVCKRS. Associated with each exit routine will be a one byte unique exit routine ID. This ID will also reside in the table. Any future exit routines will be added to the end of the table.

The SSCRCKRS bit within the first input SSCR will be examined. If it is one, the SSCRBKC bit will be examined. If it is one, Checkpoint/Restart has requested a count of SSCRs to be returned. "Checkpoint Count SSCR Processing" will be done. If the SSCRBKC bit is zero, "Checkpoint Processing" will be done. If the SSCRCXRS bit is zero, a restart operation is taking place. "Common IEAVCKRS processing" will be performed.

All exit routines previously LOADed will then be DELETEd. Control will pass back to the caller with the appropriate return code. The return code will be in the following format.

'0000xxyy' Where xx is a one byte exit routine
ID of the exit routine which passed
a non-zero return code
and
yy is the return code passed back from
the exit routine whose ID is xx

If all exit routines completed successfully, the return code will be zero.

"CHECKPOINT COUNT SSCR PROCESSING"

IEAVCKRS will keep a count of the total number of SSCR blocks requested by the exit routines. One additional SSCR will be needed to contain IEAVCKRS control information. This SSCR will be referred to as the system SSCR.

The count of total SSCRs will be initialized to one (for the system SSCR). Then, for every exit routine in the table, the following will be done. The SSCRBLKC field in the input SSCR will be zeroed. Then the exit routine will be called. The address of the input SSCR will be passed in register one. Upon return, if the SSCRBLKC is not less than zero, it will be added to the count of total SSCRs. If the SSCRBLKC field is less than zero, IEAVCKRS will return to Checkpoint/Restart with the appropriate return code.

After all exit routines have been called, the count of SSCRs will be compared to one. If equal, (none of the exits require SSCRs) the SSCRBLKC field will be set to zero. Checkpoint will not invoke IEAVCKRS again if no SSCRs are requested since none of the exit routines have data to checkpoint. If the count of SSCRs is not equal to one, SSCRBLKC will be set to the count of SSCRs.

"CHECKPOINT PROCESSING"

Because Checkpoint/Restart provides IEAVCKRS no usable data area across the first and second calls to IEAVCKRS (between "Checkpoint Count SSCR Processing" and "Checkpoint

Processing"), the count of SSCRs needed by each exit routine is not saved from the first call. Register one points to the queue of SSCRs whose first element is the system SSCR. "Checkpoint Count SSCR Processing" will be repeated with one exception. Instead of totaling the number of SSCR blocks, the number of SSCRs requested by each exit routine will be saved in the system SSCR. This will be accomplished in the following way.

The SSCRCKRS and SSCRBKC bits of a 20 byte dummy SSCR will be set to one. Then for each exit routine in the table, SSCRBLKC in the dummy SSCR will be zeroed. The exit routine will then be called. Register one will contain the address of the dummy SSCR. Upon return, the dummy SSCRBLKC field will be saved in the system SSCR in a location associated with the exit routine invoked.

The number of SSCRs on the queue will then be compared to the total number of SSCRs recorded in the system SSCR. If the counts are not equal, at least one of the called exit routines has requested more or less SSCRs than it requested on the first call. In this case, IEAVCKRS will return to Checkpoint/Restart with the indicative return code.

After all exit SSCR counts have been recorded in the system SSCR, "Common IEAVCKRS Processing" will be performed.

"COMMON IEAVCKRS PROCESSING"

For the purpose of discussion, let each SSCR in the queue of SSCRs be subscripted from 1 to N. Furtner let SSCR(1) be the system SSCR.

An SSCR-INDEX will be set to 2 (pointing to the first non-system SSCR). Then for each exit routine in the table that requested more than 0 SSCRs, the following will be done.

The forward pointer in the last SSCR belonging to the current exit routine (SSCRFCHN) will be saved. Then the SSCRFCHN field will be zeroed. This will prevent the called exit routine from accessing SSCR blocks it does not own. The SSCRFLG1 field from the system SSCR (SSCR(1)) will be copied to the SSCRFLG1 field in SSCR(SSCR-INDEX). The SSCRFLG1 field contains the SSCRCKRS and SSCRBKC bits which communicate to the exit routine whether the checkpoint or restart action is to be taken. The exit will then be called. Register one will contain the address of SSCR(SSCR-INDEX). Upon return, the saved SSCRFCHN field will be restored to the forward pointer of the last SSCR belonging to the exit routine just processed. The SSCR-INDEX will be advanced to the first SSCR belonging to the next exit routine to be called.

RECOVERY OPERATION:

IEAVCKRS runs under the caller's recovery.

Note - Each exit routine is responsible for its own recovery. Checkpoint/Restart has recovery set up before the call to IEAVCKRS. Any undetected error in an exit routine will percolate to Checkpoint/Restart.

SUP-53. Checkpoint/Restart Exit Routine Router (IEAVCKRS)

IEAVCKRS - DIAGNOSTIC AIDS

ENTRY POINT NAME: IEAVCKRS

MESSAGES: None

ABEND CODES: None

WAIT STATE CODES: None

RETURN CODES:

EXIT NORMAL:

Return codes are presented to the caller in register 15.

Return code	Explanation
X'00000000'	All exit routines returned a zero return code.

EXIT ERROR:

Return codes are presented to the caller in register 15.

Return code	Explanation
X'0000xxyy'	The exit routine whose ID is xx passed back the non-zero yy return code.

Specific return codes for error conditions detected in the IEAVCKRS module are as follows:

Return code	Explanation
X'00000108'	IEAVCKRS has detected that the number of SSCRs requested by the exit routines from the first Checkpoint/Restart invocation is not equal to the number of of SSCRs requested by the exit routines on the second Checkpoint/Restart invocation
X'0000xxFF'	IEAVCKRS has detected that the the exit routine whose ID is xx has requested a negative number of SSCRs

REGISTER CONTENTS ON ENTRY:

0 - Irrelevant
1 - Address of SSCR Queue
2-12 - Irrelevant
13 - Address of 18 Word Save Area
14 - Return Address
15 - Entry Point Address

REGISTER CONTENTS ON EXIT:

IEAVCKRS - DIAGNOSTIC AIDS (Continued)

EXIT NORMAL:

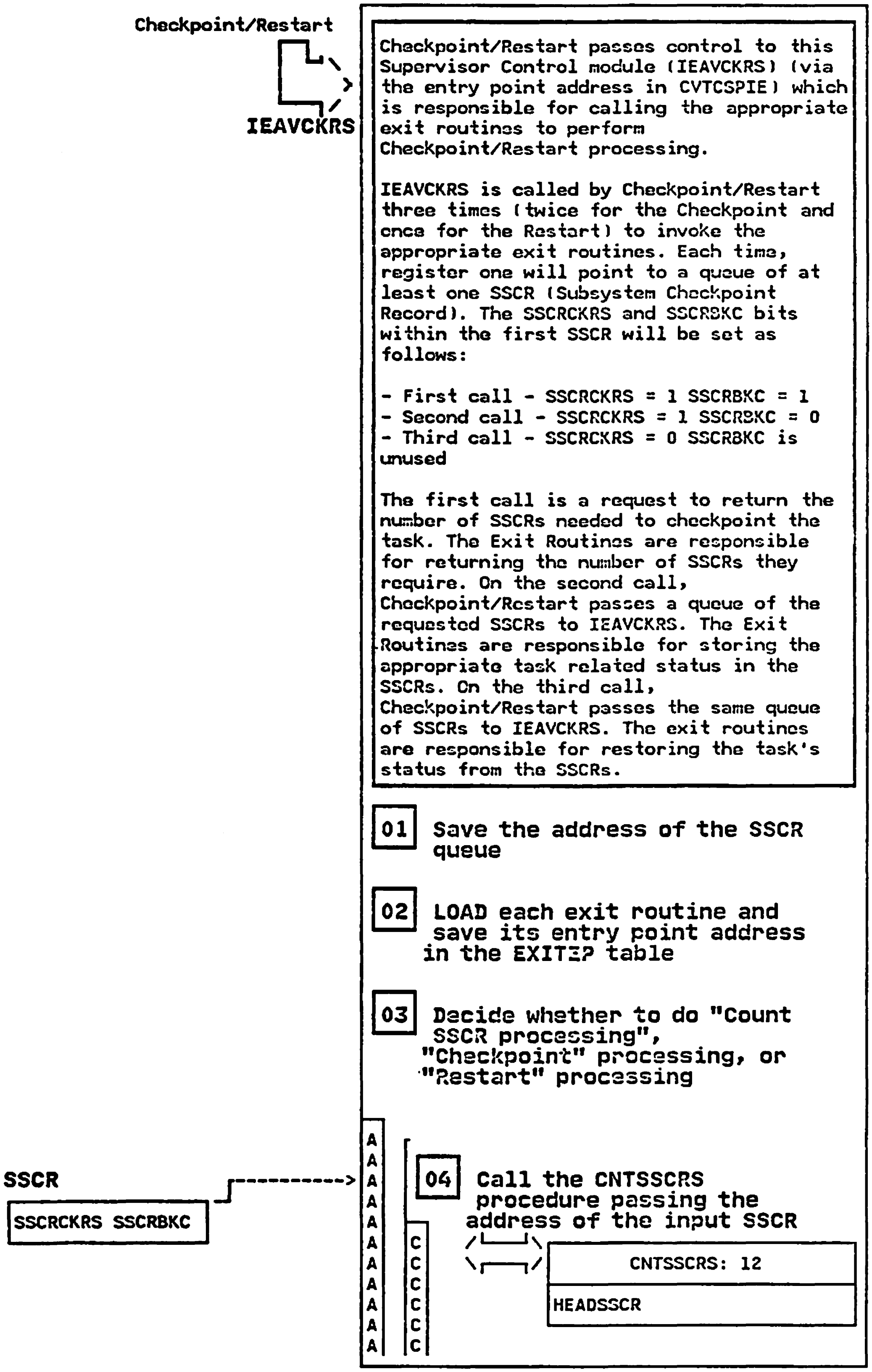
0-13	Unchanged
14	Return address
15	Return code

EXIT ERROR:

0-13	Unchanged
14	Return address
15	Return code

SUP-53. Checkpoint/Restart Exit Routine Router (IEAVCKRS)

IEAVCKRS - Checkpoint/Restart Exit Routine Router STEP 01



Checkpoint/Restart passes control to this Supervisor Control module (IEAVCKRS) (via the entry point address in CVTCSPIE) which is responsible for calling the appropriate exit routines to perform Checkpoint/Restart processing.

IEAVCKRS is called by Checkpoint/Restart three times (twice for the Checkpoint and once for the Restart) to invoke the appropriate exit routines. Each time, register one will point to a queue of at least one SSCR (Subsystem Checkpoint Record). The SSCRCKRS and SSCRBKC bits within the first SSCR will be set as follows:

- First call - SSCRCKRS = 1 SSCRBKC = 1
- Second call - SSCRCKRS = 1 SSCRBKC = 0
- Third call - SSCRCKRS = 0 SSCRBKC is unused

The first call is a request to return the number of SSCRs needed to checkpoint the task. The Exit Routines are responsible for returning the number of SSCRs they require. On the second call, Checkpoint/Restart passes a queue of the requested SSCRs to IEAVCKRS. The Exit Routines are responsible for storing the appropriate task related status in the SSCRs. On the third call, Checkpoint/Restart passes the same queue of SSCRs to IEAVCKRS. The exit routines are responsible for restoring the task's status from the SSCRs.

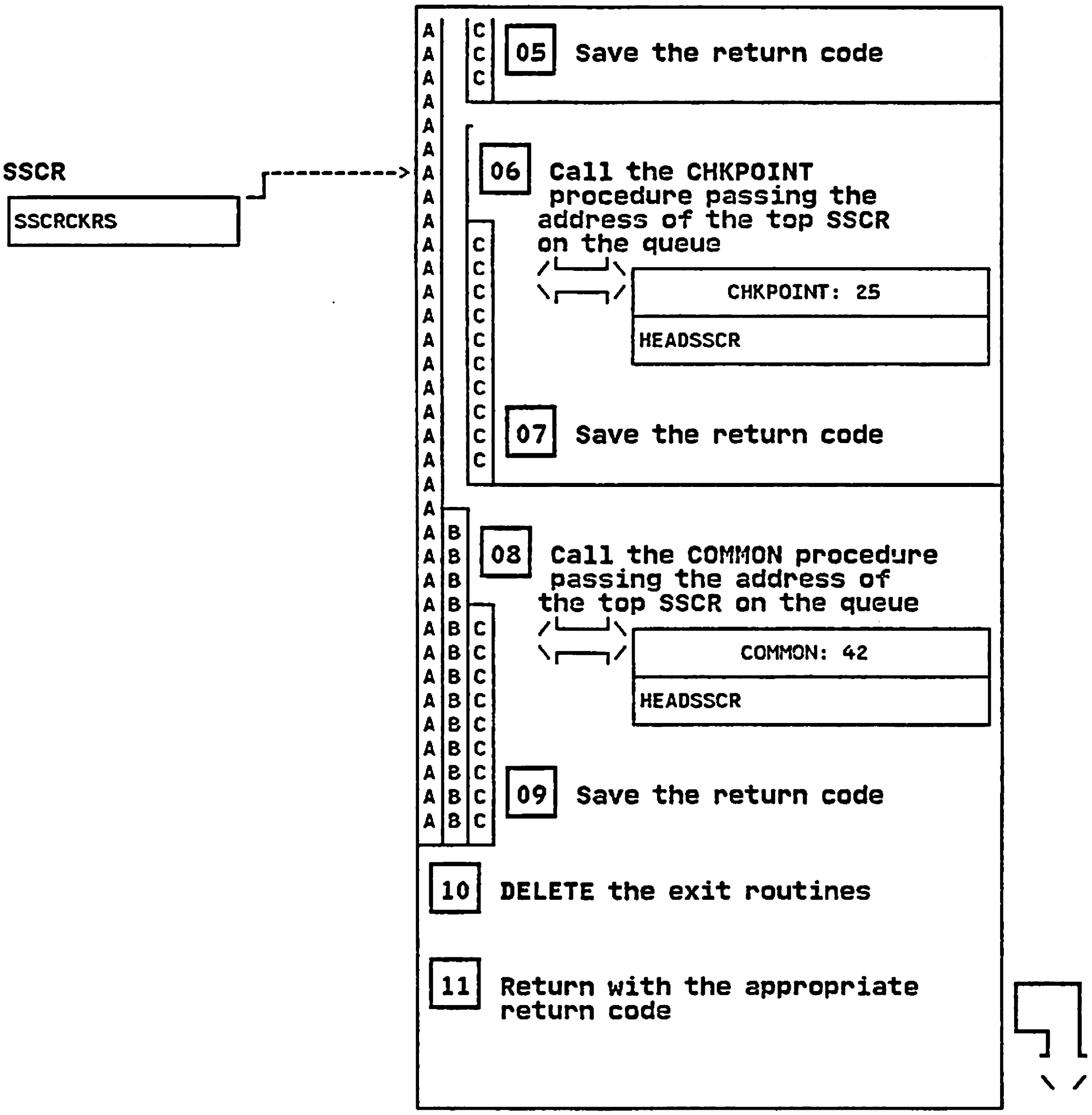
01 Save the address of the SSCR queue

02 LOAD each exit routine and save its entry point address in the EXITEP table

03 Decide whether to do "Count SSCR processing", "Checkpoint" processing, or "Restart" processing

04 Call the CNTSSCRS procedure passing the address of the input SSCR

IEAVCKRS - Checkpoint/Restart Exit Routine Router STEP 05



05 Save the return code

06 Call the CHKPOINT procedure passing the address of the top SSCR on the queue

07 Save the return code

08 Call the COMMON procedure passing the address of the top SSCR on the queue

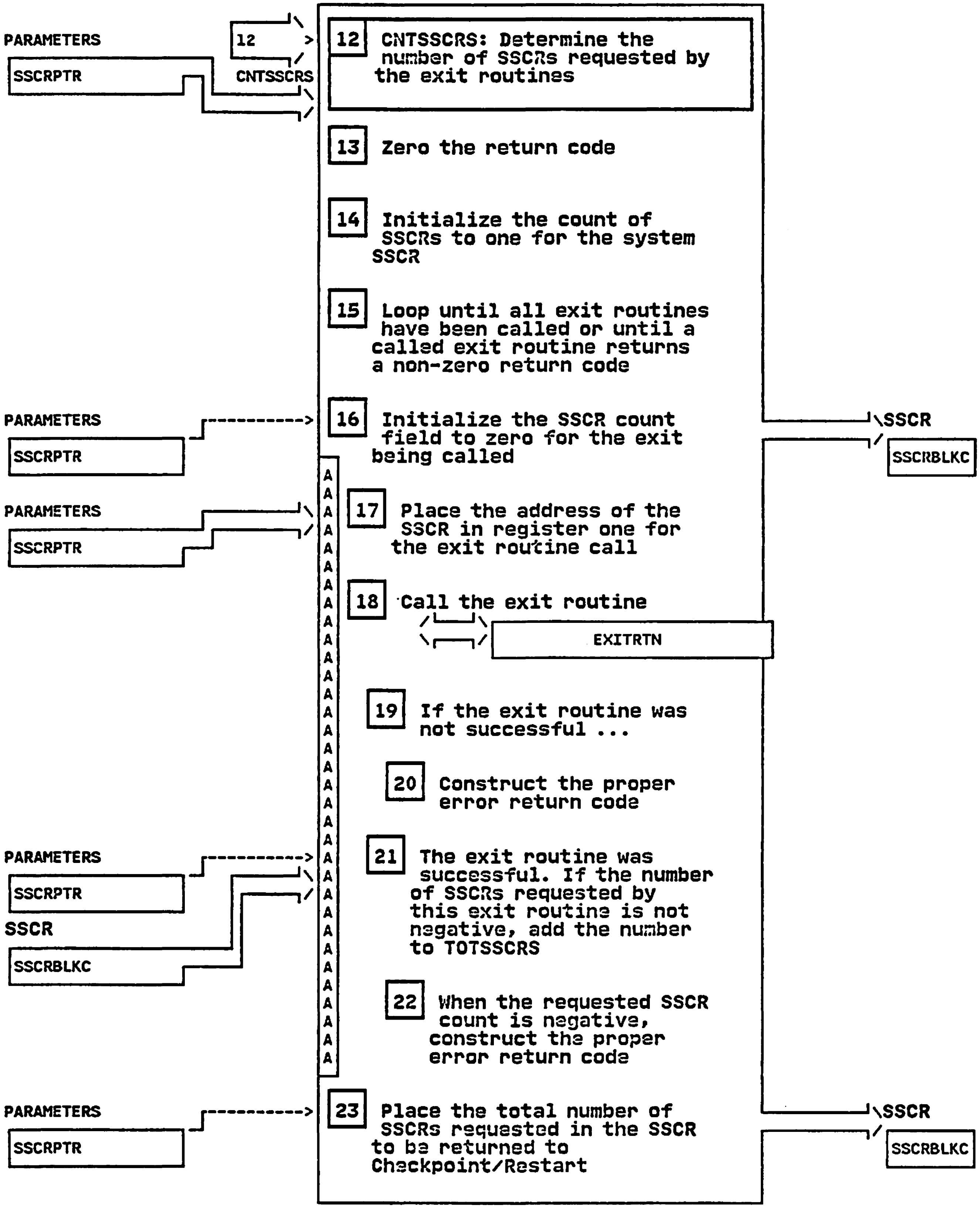
09 Save the return code

10 DELETE the exit routines

11 Return with the appropriate return code

SUP-53. Checkpoint/Restart Exit Routine Router (IEAVCKRS)

IEAVCKRS - Checkpoint/Restart Exit Routine Router STEP 12



12 CNTSSCRS: Determine the number of SSCRs requested by the exit routines

13 Zero the return code

14 Initialize the count of SSCRs to one for the system SSCR

15 Loop until all exit routines have been called or until a called exit routine returns a non-zero return code

16 Initialize the SSCR count field to zero for the exit being called

17 Place the address of the SSCR in register one for the exit routine call

18 Call the exit routine

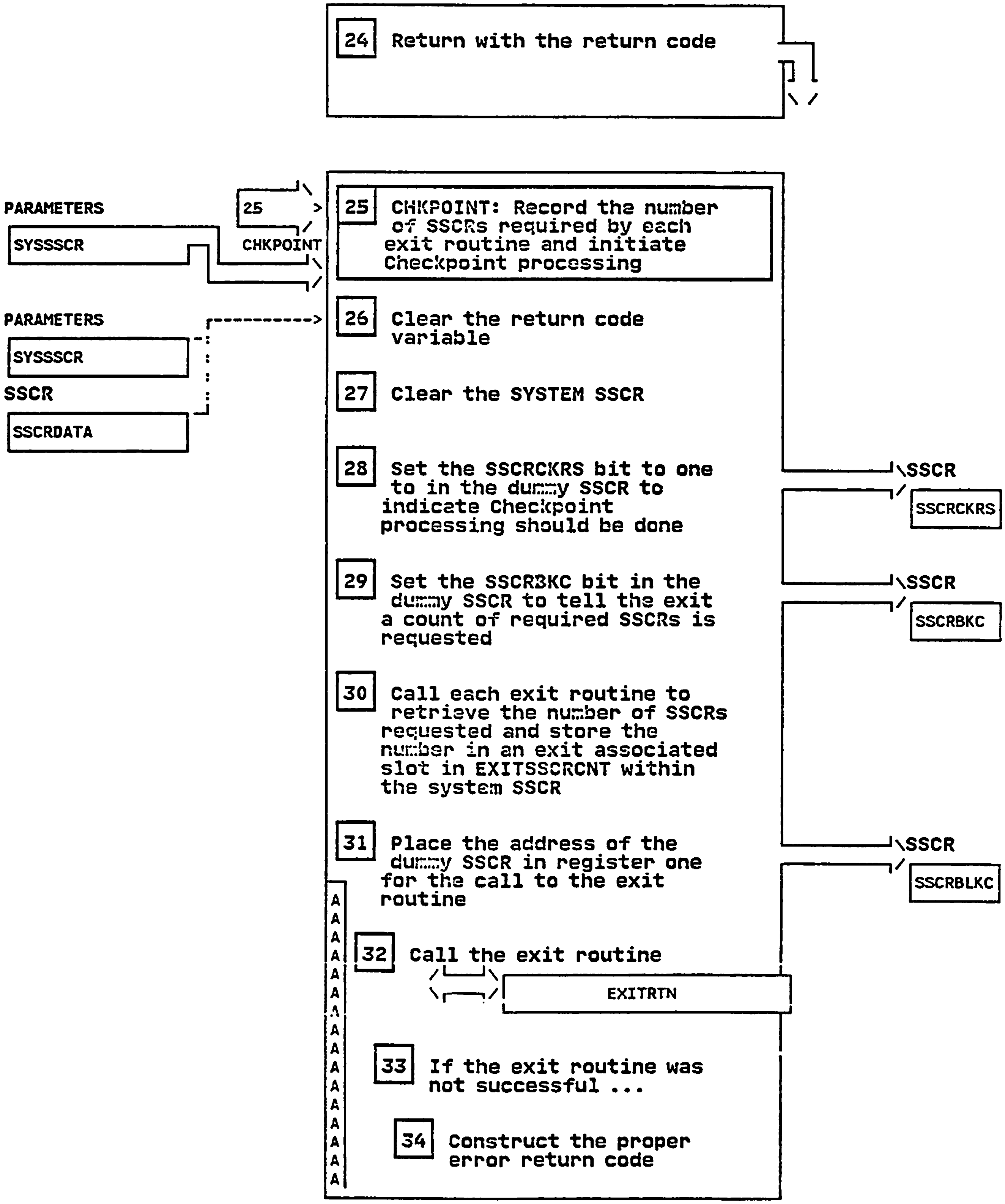
19 If the exit routine was not successful ...

20 Construct the proper error return code

21 The exit routine was successful. If the number of SSCRs requested by this exit routine is not negative, add the number to TOTSSCRS

22 When the requested SSCR count is negative, construct the proper error return code

23 Place the total number of SSCRs requested in the SSCR to be returned to Checkpoint/Restart



24 Return with the return code

25 CHKPOINT: Record the number of SSCRs required by each exit routine and initiate Checkpoint processing

26 Clear the return code variable

27 Clear the SYSTEM SSCR

28 Set the SSCRCKRS bit to one to in the dummy SSCR to indicate Checkpoint processing should be done

29 Set the SSCRBKC bit in the dummy SSCR to tell the exit a count of required SSCRs is requested

30 Call each exit routine to retrieve the number of SSCRs requested and store the number in an exit associated slot in EXITSSCRCNT within the system SSCR

31 Place the address of the dummy SSCR in register one for the call to the exit routine

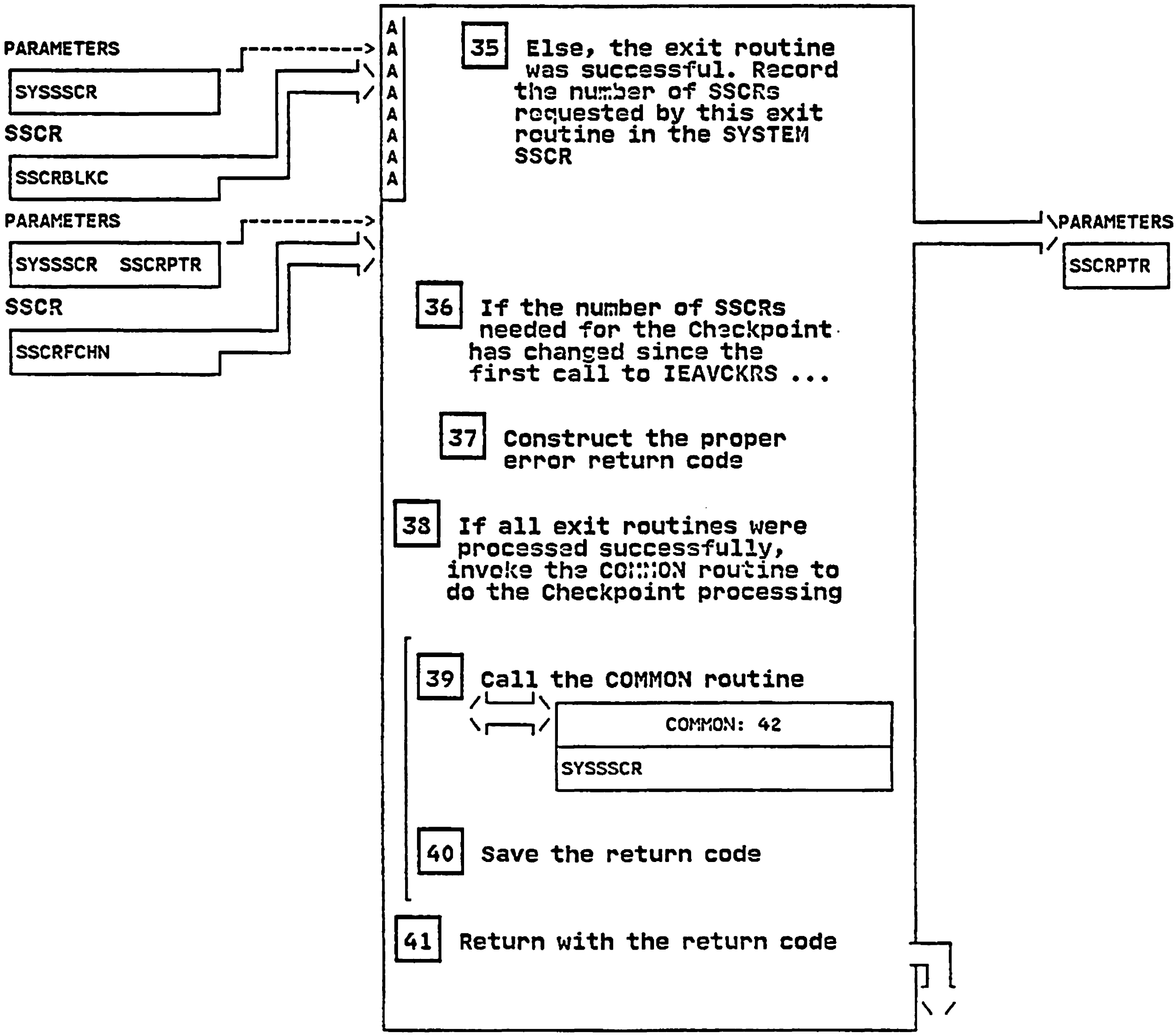
32 Call the exit routine

33 If the exit routine was not successful ...

34 Construct the proper error return code

SUP-53. Checkpoint/Restart Exit Routine Router (IEAVCKRS)

IEAVCKRS - Checkpoint/Restart Exit Routine Router STEP 35



35 Else, the exit routine was successful. Record the number of SSCRs requested by this exit routine in the SYSTEM SSCR

36 If the number of SSCRs needed for the Checkpoint has changed since the first call to IEAVCKRS ...

37 Construct the proper error return code

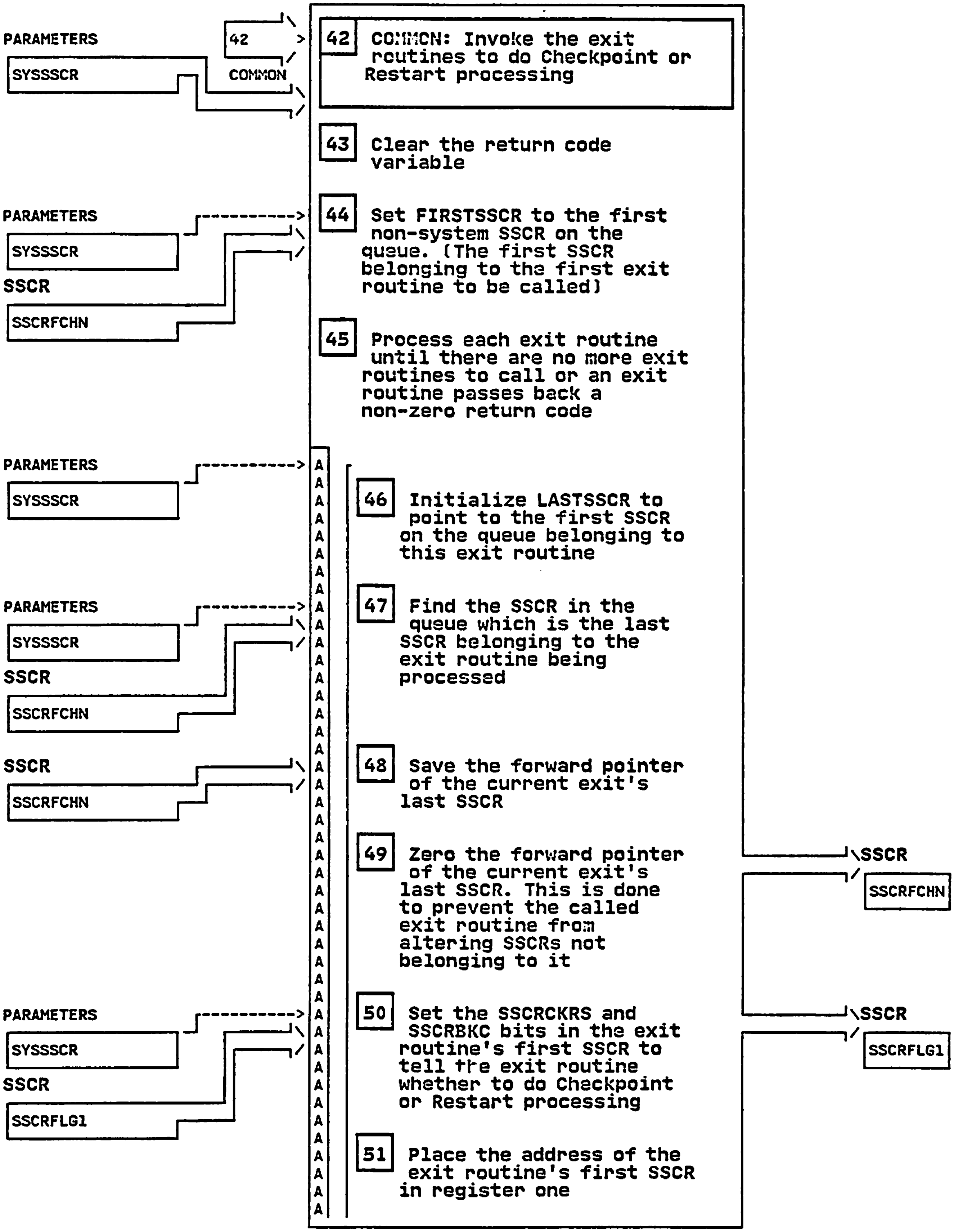
38 If all exit routines were processed successfully, invoke the COMMON routine to do the Checkpoint processing

39 Call the COMMON routine

40 Save the return code

41 Return with the return code

IEAVCKRS - Checkpoint/Restart Exit Routine Router   STEP 42



42 COMMON: Invoke the exit routines to do Checkpoint or Restart processing

43 Clear the return code variable

44 Set FIRSTSSCR to the first non-system SSCR on the queue. (The first SSCR belonging to the first exit routine to be called)

45 Process each exit routine until there are no more exit routines to call or an exit routine passes back a non-zero return code

46 Initialize LASTSSCR to point to the first SSCR on the queue belonging to this exit routine

47 Find the SSCR in the queue which is the last SSCR belonging to the exit routine being processed

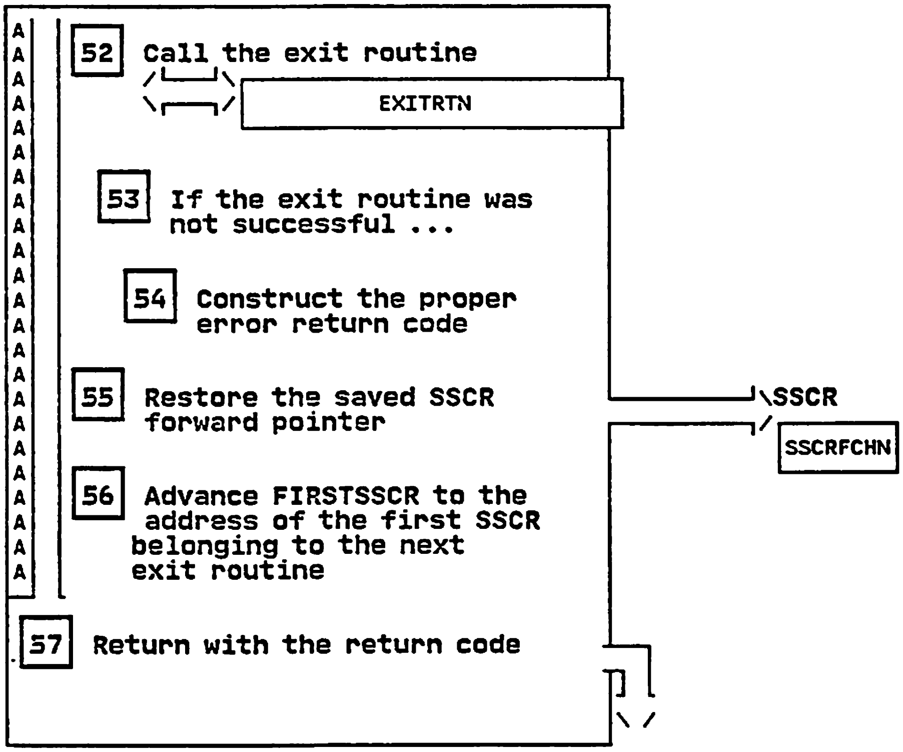
48 Save the forward pointer of the current exit's last SSCR

49 Zero the forward pointer of the current exit's last SSCR. This is done to prevent the called exit routine from altering SSCRs not belonging to it

50 Set the SSCRCKRS and SSCRBKC bits in the exit routine's first SSCR to tell the exit routine whether to do Checkpoint or Restart processing

51 Place the address of the exit routine's first SSCR in register one

IEAVCKRS - Checkpoint/Restart Exit Routine Router STEP 52



52 Call the exit routine

53 If the exit routine was not successful ...

54 Construct the proper error return code

55 Restore the saved SSCR forward pointer

56 Advance FIRSTSSCR to the address of the first SSCR belonging to the next exit routine

57 Return with the return code

IEAVCRVF - MODULE DESCRIPTION

DESCRIPTIVE NAME: Vector Checkpoint/Restart Exit Routine

FUNCTION:

IEAVCRVF is responsible for saving or restoring the subject task's vector related status during a Checkpoint or Restart operation.

IEAVCRVF is called by the Supervisor Checkpoint/Restart Router module (IEAVCKRS) three times (twice for Checkpoint processing and once for Restart processing) to perform the Checkpoint and Restart processing. Each time, register one contains the address of a queue of at least one SSCR (Subsystem Checkpoint Record). The SSCRCKRS and SSCRBKC bits within the first SSCR will be set as follows.

First call -	SSCRCKRS = 1	SSCRBKC = 1
Second call -	SSCRCKRS = 1	SSCRBKC = 0
Third call -	SSCRCKRS = 0	SSCRBKC is unused

The first call is a request to return the number of SSCRs needed to checkpoint the task. On the second call, IEAVCKRS passes a queue of the requested SSCRs to IEAVCRVF. The exit routine is responsible for storing the appropriate vector related status in the SSCRs. On the third call, IEAVCKRS passes the same queue of SSCRs to IEAVCRVF. The exit routine is responsible for restoring the task's vector status from the SSCRs.

SUP-54. VECTOR CHECKPOINT/RESTART EXIT ROUTINE (IEAVCRVF)

ENTRY POINT: IEAVCRVF

PURPOSE: Refer to the module function topic

LINKAGE: Branch entered

CALLERS: IEAVCKRS (Supervisor Checkpoint/Restart Router)

INPUT:

- The subject task's TCB referenced via PSATOLD
- A queue of one or more SSCRs pointed to by register one upon entry
- The SSCRCKRS and SSCRBKC bits in the first SSCR on the queue
- The Vector Status Save Area of a task using the vector facility
- The STCBVSSA pointer from the subject task
- The CVTVSS and CBLVSSA fields

OUTPUT:

- SSCRBLKC set to the total of SSCRs needed to checkpoint the task's vector status (When SSCRCKRS and SSCRBKC are set to one)
- The task's vector status saved in the queue of SSCRs (When SSCRCKRS is set to one and SSCRBKC is zero)
- The task's VSSA (obtained by the Vector SLIH) restored from the data in the queue of SSCRs (When SSCRCKRS is zero)
- Return code in register 15

EXIT NORMAL: To IEAVCKRS with return code in register 15

ENTRY POINT: CRVFRRTN

PURPOSE: Provide recovery for the IEAVCRVF mainline.

LINKAGE: Branch entered

CALLERS: RTM

INPUT:

- Values placed in the FRR parameter area by the mainline for registers 11 through 13
- The address of the mainline return code field placed in the FRR parameter area by the mainline
- The address of the top SSCR on the queue for recording purposes only (placed in the FRR parameter area by the mainline)
- The SDWACMPC and SDWAHRC fields in the SDWA
- The STCBVSSA field of the subject task

SUP-54. Vector Checkpoint/Restart Exit Routine (IEAVCRVF)

IEAVCRVF - MODULE DESCRIPTION (Continued)

OUTPUT:

- Standard error diagnostic information recorded in the SDWA
- The appropriate SDWA fields for RETRY communication to RTM
- The SSCRHDR area from the head of the input SSCR queue recorded in the VRA
- The FRR parameter area recorded in the VRA
- The mainline return code field set to the appropriate value
- The STCBVAFN and TCBAFFN fields reset to reflect "stolen" vector affinity
- The vector control bit in Control Register Zero set to zero by the Dispatcher via the CALLDISP macro

EXIT NORMAL: To RTM

EXTERNAL REFERENCES:

ROUTINES: None

CONTROL BLOCKS:

CBLS	(Control Block Lengths Table)	- (r)
CVT	(Communication Vector Table)	- (r)
FRRS	(Functional Recovery Routine Stack)	- (r,w)
PSA	(Prefixed Save Area)	- (r)
SDWA	(System Diagnostic Work Area)	- (r,w)
SSCR	(Subsystem Checkpoint Record)	- (r,w)
STCB	(Secondary Task Control Block)	- (r,w)
SVT	(Supervisor Vector Table)	- (r)
TCB	(Task Control Block)	- (r,w)
VSSA	(Vector Status Save Area)	- (r,w)

r = Read
w = Write

SERIALIZATION:

STCBVAFN is set to zero to ensure that the Dispatcher and Stop/Reset do not attempt to save vector status while IEAVCRVF is altering the task's VSSA during Restart processing.

SUP-54. Vector Checkpoint/Restart Exit Routine (IEAVCRVF)

IEAVCRVF - MODULE OPERATION

IEAVCRVF will set an EUT type FRR which will pass control to an internal recovery routine (CRVFRRTN) should an error occur. SVCs cannot be issued while the EUT FRR is in place. The values from registers 11, 12, and 13 will be saved in the FRR parameter area to allow use in the recovery routine. Registers 11 through 13 contain the address of the data area, the module base address, and the address of the general register save area.

The SSCRCKRS bit in the first SSCR of the input SSCR queue will then be examined. If the SSCRCKRS bit is one, the SSCRBKC will be examined. If it is one, "CNTSSCRS" processing will be performed. If the SSCRBKC bit is zero, "CHKPOINT" processing will be performed. If the SSCRCKRS bit is zero, "RESTART" processing will be done. The return code from the called subroutine will be saved.

The EUT FRR will then be deleted and control will pass to the caller with the saved return code.

"CNTSSCRS" Processing

PSATOLD will be used to gain addressability to the subject TCB. STCBVSSA will then be compared to zero. If it is zero, the SSCRBLKC field within the first input SSCR will be set to zero to indicate no vector checkpoint will be done.

If STCBVSSA is non-zero, IEAVCRVF will calculate the number of SSCR blocks needed to checkpoint the task's Vector Status Save Area (VSSA). The size of the VSSA will be obtained by multiplying the vector section size (the value in CVTVSS) by 64 and adding the size of the static portion of the VSSA (CBLVSSA) to the result. The number of SSCRs needed to store the VSSA will be placed in the SSCRBLKC field of the first SSCR on the input SSCR queue. When calculating the number of SSCR blocks, remember VSSA data will be stored in the SSCRDATA fields only.

"CHKPOINT" Processing

The VSSA will be validity checked via the LRA instruction and "VSSA" acronym checks. If the STCBVSSA does not point to a valid VSSA, control will be passed to the caller with a return code of X'04'. Otherwise, a "CALLDISP BRANCH=YES, FIXED=NO,FRRSTK=SAVE" will be issued to cause the dispatcher to save the task's vector status in the VSSA. When IEAVCRVF is re-dispatched, the STCBVMCK bit will be checked to determine whether vector status for the task was lost due to a machine check. If so, control will be passed to the caller with a return code of X'20'.

If vector status for the task was saved successfully, the task's VSSA will then be copied into the SSCRDATA fields of successive elements of the input SSCR queue. The current level number of the VSSA (CBLVSSAL) will be stored in the in the first SSCR at the VSSARPTR field location. The actual VSSARPTR value is not needed for the Restart operation. The saved CBLVSSAL value will be compared to the level of the VSSA at the time of the Restart to ensure VSSA compatibility.

"RESTART" Processing

This processing will restore the task's vector status on a restart operation.

A simple vector instruction will be issued (VXVC, Extract Vector Count). This will cause the vector SLIH to build and initialize a vector environment for the task. After the vector instruction is issued, STCBVAFN will be zeroed. This will stop the Dispatcher and Stop/Reset from saving/restoring vector status for the task while IEAVCRVF is restoring the VSSA. TCBAFFN will then be set to the value in STCBAFNS. A "CALLDISP BRANCH=YES,FIXED=NO,FRRSTK=SAVE" will then be issued to cause the dispatcher to turn off the vector control bit in control register 0. To turn off the vector control bit in IEAVCRVF would require disablement to serialize the PSACROSV field.

When IEAVCRVF is re-dispatched, the VSSA will be validity checked via the LRA instruction and "VSSA" acronym checks. If the STCBVSSA does not point to a valid VSSA, control will be passed to the caller with a return code of X'04'.

The Vector section size recorded in the SSCR data area at Checkpoint time will be compared against the current section size. If the section sizes are not equal, control will be passed to the caller with a return code of X'10'.

The VSSA level number saved in the SSCR data area at Checkpoint time will be compared against the current VSSA level number in the CBLS. If they are not equal, control will be passed to the caller with a return code of X'14'.

Appropriate VSSA fields will then be copied from the queue of SSCR blocks to the task's VSSA. (The VSSA was created as a result of the vector instruction issued.)

RECOVERY OPERATION:

The FRR Routine (CRVFRRTN) will be responsible for recording standard diagnostic information in the SDWA. Also, FRR parmeter area data and other message text will be recorded in the VRA.

The FRR will then determine whether the error was caused because none of the online vector processors meets the task's affinity requirements, because there is no vector processing capability in the complex, because the VSSA could not be obtained by the Vector SLIH, or because of an unexpected program check. If the Vector SLIH initiated a X'09F' reason code X'08' ABEND, the return code field from the mainline will be set to X'08'. The address of the return code field was passed to the recovery routine in the FRR parameter area. If the Vector SLIH initiated a X'09F' reason code X'0C' ABEND, the return code field from the mainline will be set to X'0C'. If the Vector SLIH initiated a X'09F' reason code X'24' ABEND or if the VXVC instruction issued by the Restart

IEAVCRVF - MODULE OPERATION (Continued)

routine in this module resulted in a X'0C1' ABEND, the return code field will be set to X'18'. In all other cases, the return code field will be set to X'1C' to indicate an unexpected error occurred.

If the task had obtained a VSSA, vector affinity will be stolen in the following way. STCBVAFN will be set to zero and TCBAFFN will be set to STCBAFNS. A "CALLDISP BRANCH=YES, FIXED=NO, FRRSTK=SAVE" will then be issued to cause the dispatcher to turn off the vector control bit in control register 0.

The FRR will retry to label CRVFEXIT in IEAVCRVF where the FRR will be deleted and where control will be passed to the caller with the appropriate mainline return code.

SUP-54. Vector Checkpoint/Restart Exit Routine (IEAVCRVF)

IEAVCRVF - DIAGNOSTIC AIDS

ENTRY POINT NAMES: IEAVCRVF
CRVFRRTN

MESSAGES: None

ABEND CODES: None

WAIT STATE CODES: None

RETURN CODES:

ENTRY POINT IEAVCRVF:

EXIT NORMAL:

Return codes are presented to the caller in register 15.

Return code	Explanation
X'00000000'	IEAVCRVF processing was successful.

EXIT ERROR:

Return codes are presented to the caller in register 15.

Return code	Explanation
X'00000004'	IEAVCRVF has detected an invalid pointer to the subject task's VSSA.
X'00000008'	The Vector SLIH has determined that none of the online vector processors meets the task's affinity requirements.
X'0000000C'	The Vector SLIH could not obtain a VSSA for the task.
X'00000010'	The vector section size recorded for the Checkpoint is not equal to the vector section size for the Restart.
X'00000014'	The mapping of the Vector Status Save Area (VSSA) has changed from the time the Checkpoint was taken to the time the Restart was attempted.
X'00000018'	There exists no vector processor in the complex.
X'0000001C'	An unexpected error has occurred.
X'00000020'	Vector status has been lost due to a machine check.

Exit routines invoked by IEAVCKRS are assigned a one byte unique exit routine ID. IEAVCRVF's ID is X'03'. When one of the above error return codes is passed back to IEAVCKRS, it is concatenated with the ID

IEAVCRVF - DIAGNOSTIC AIDS (Continued)

and returned to Checkpoint/Restart.

ENTRY POINT CRVFRRTN: None

REGISTER CONTENTS ON ENTRY:

ENTRY POINT IEAVCRVF:

```
   0 - Irrelevant
   1 - Address of SSCR Queue
2-12 - Irrelevant
  13 - Address of 18 Word Save Area
  14 - Return Address
  15 - Entry Point Address
```

ENTRY POINT CRVFRRTN:

```
   0 - Address of a 200 byte FRR work area (Unused)
   1 - Address of the SDWA
2-13 - Irrelevant
  14 - RTM return address
  15 - Entry point address
```

REGISTER CONTENTS ON EXIT:

ENTRY POINT IEAVCRVF:

EXIT NORMAL:

```
0-13 Unchanged
14   Return address
15   Return code
```

EXIT ERROR:

```
0-13 Unchanged
14   Return address
15   Return code
```

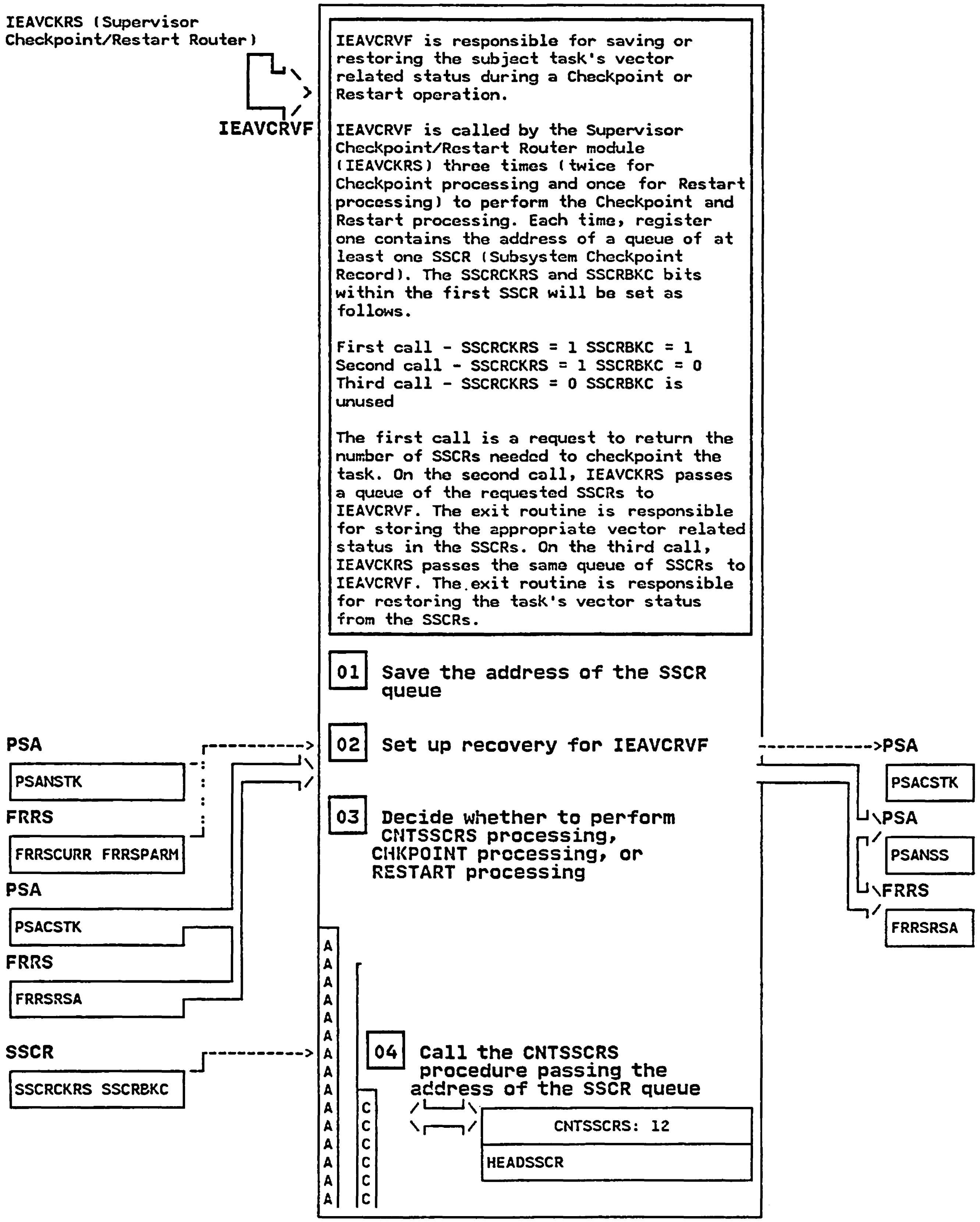
ENTRY POINT CRVFRRTN:

EXIT NORMAL:

```
0-13 Irrelevant
14   Return address
15   Irrelevant
```

SUP-54. Vector Checkpoint/Restart Exit Routine (IEAVCRVF)

IEAVCRVF - Vector Checkpoint/Restart Exit Routine STEP 01



IEAVCKRS (Supervisor Checkpoint/Restart Router)

IEAVCRVF

IEAVCRVF is responsible for saving or restoring the subject task's vector related status during a Checkpoint or Restart operation.

IEAVCRVF is called by the Supervisor Checkpoint/Restart Router module (IEAVCKRS) three times (twice for Checkpoint processing and once for Restart processing) to perform the Checkpoint and Restart processing. Each time, register one contains the address of a queue of at least one SSCR (Subsystem Checkpoint Record). The SSCRCKRS and SSCRBKC bits within the first SSCR will be set as follows.

First call - SSCRCKRS = 1 SSCRBKC = 1
Second call - SSCRCKRS = 1 SSCRBKC = 0
Third call - SSCRCKRS = 0 SSCRBKC is unused

The first call is a request to return the number of SSCRs needed to checkpoint the task. On the second call, IEAVCKRS passes a queue of the requested SSCRs to IEAVCRVF. The exit routine is responsible for storing the appropriate vector related status in the SSCRs. On the third call, IEAVCKRS passes the same queue of SSCRs to IEAVCRVF. The.exit routine is responsible for restoring the task's vector status from the SSCRs.

01 Save the address of the SSCR queue

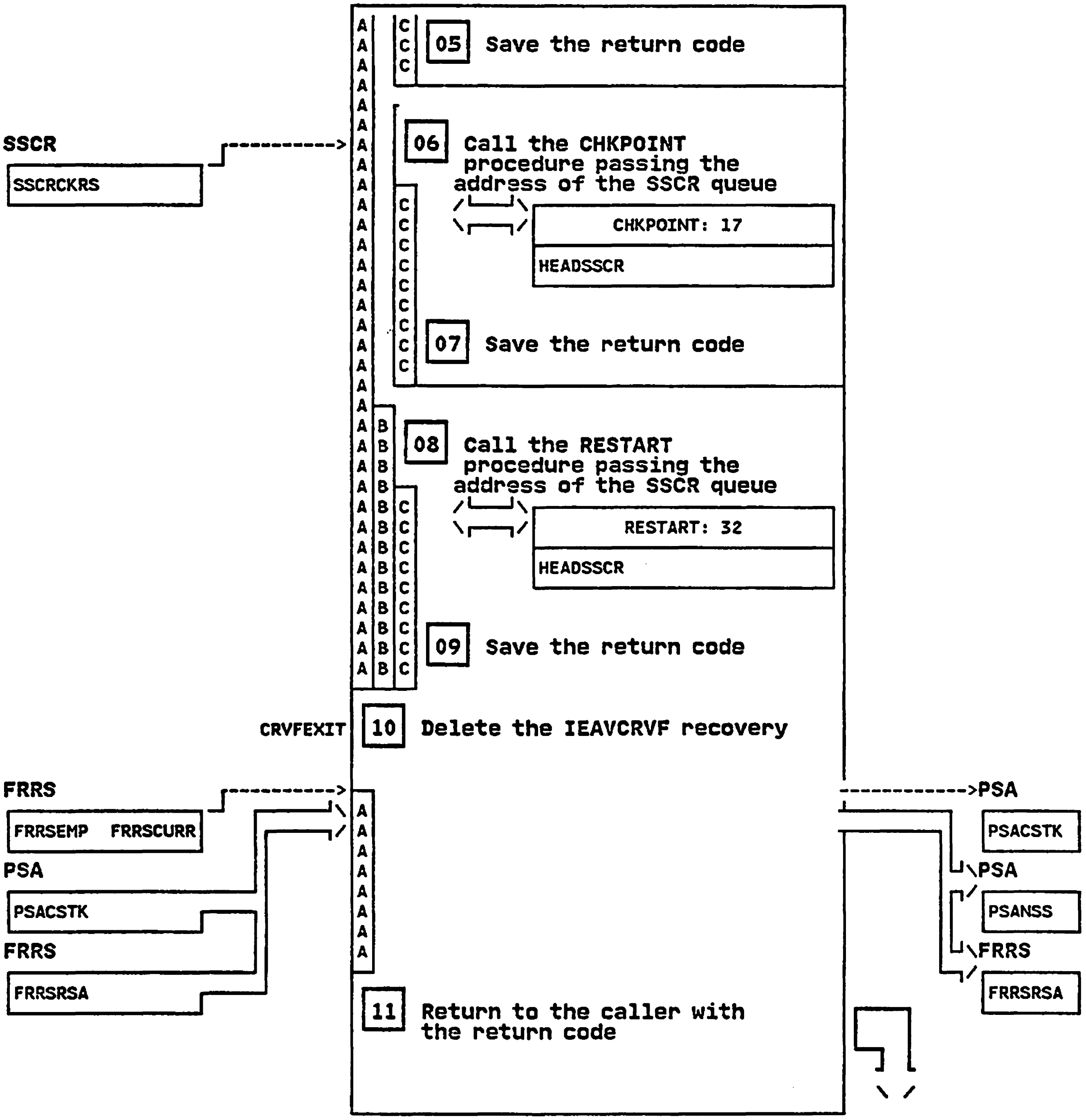
02 Set up recovery for IEAVCRVF

03 Decide whether to perform CNTSSCRS processing, CHKPOINT processing, or RESTART processing

04 Call the CNTSSCRS procedure passing the address of the SSCR queue

SUP-54. Vector Checkpoint/Restart Exit Routine (IEAVCRVF)

IEAVCRVF - Vector Checkpoint/Restart Exit Routine STEP 05



05 Save the return code

06 Call the CHKPOINT procedure passing the address of the SSCR queue

07 Save the return code

08 Call the RESTART procedure passing the address of the SSCR queue

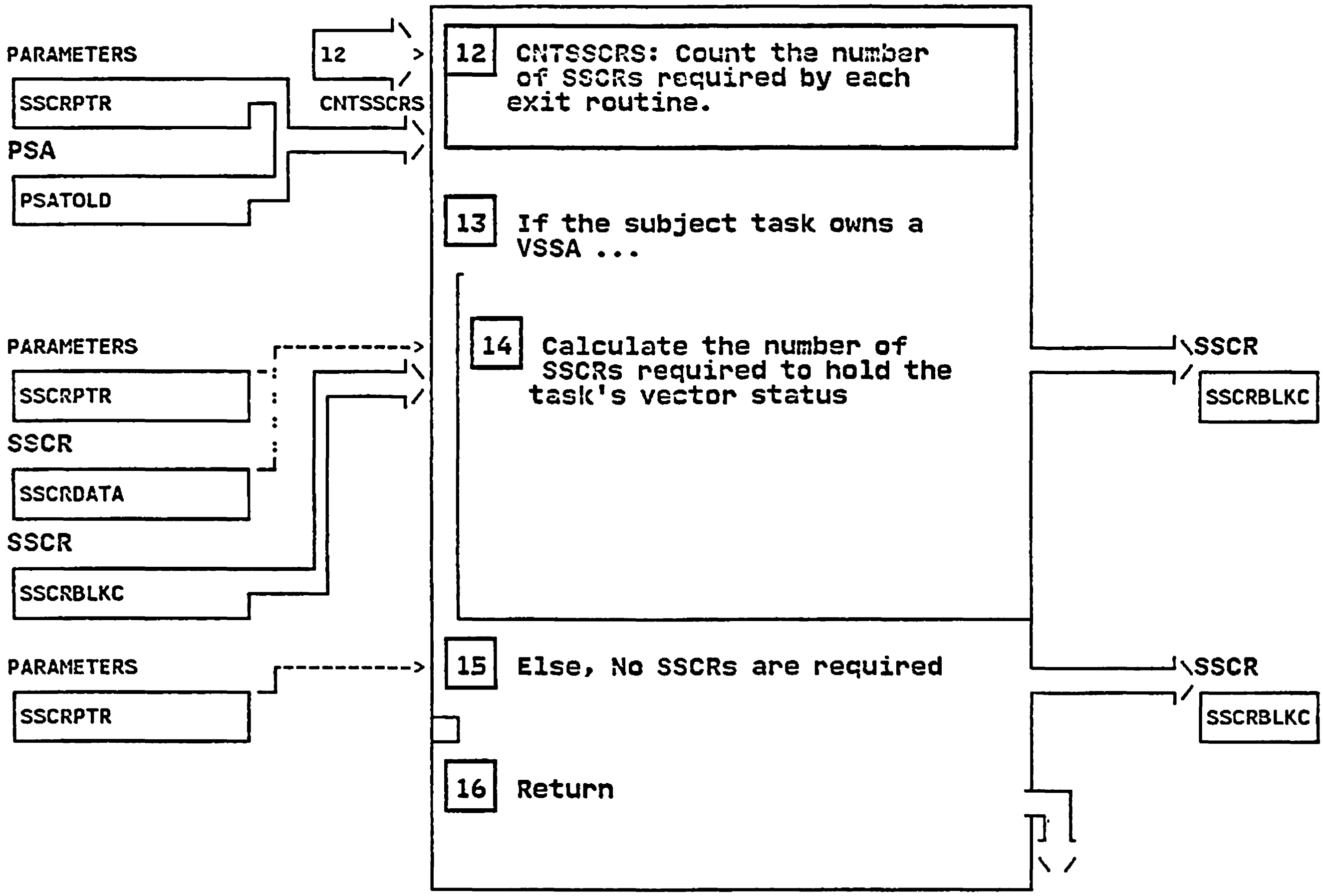
09 Save the return code

CRVFEXIT 10 Delete the IEAVCRVF recovery

11 Return to the caller with the return code

SUP-54. Vector Checkpoint/Restart Exit Routine (IEAVCRVF)

IEAVCRVF - Vector Checkpoint/Restart Exit Routine   STEP 12



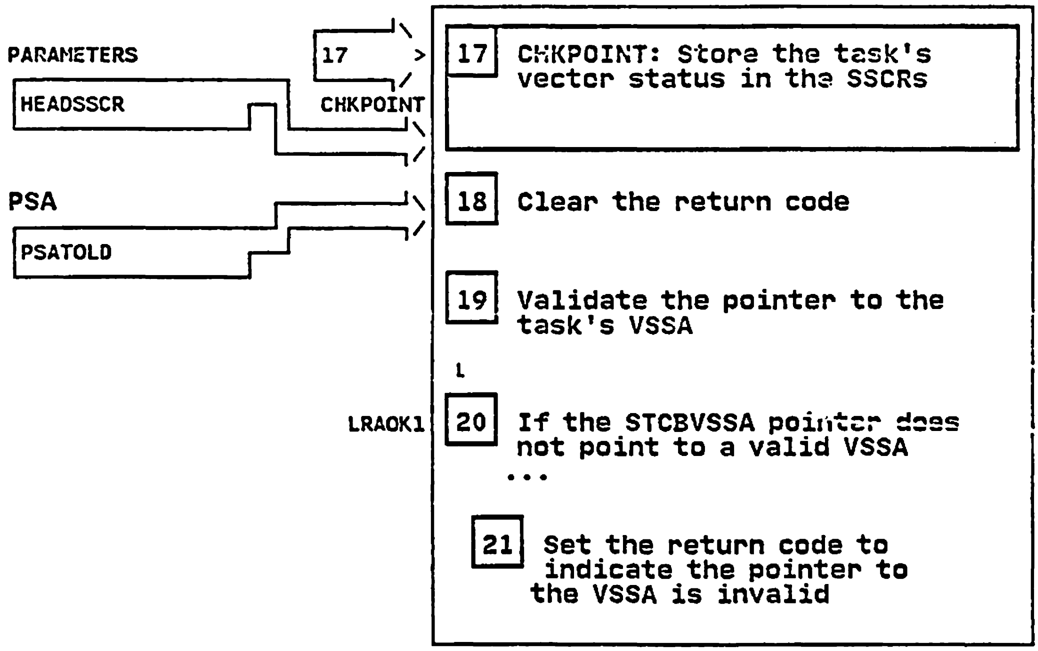
12 CNTSSCRS: Count the number of SSCRs required by each exit routine.

13 If the subject task owns a VSSA ...

14 Calculate the number of SSCRs required to hold the task's vector status

15 Else, No SSCRs are required

16 Return



17 CHKPOINT: Store the task's vector status in the SSCRs

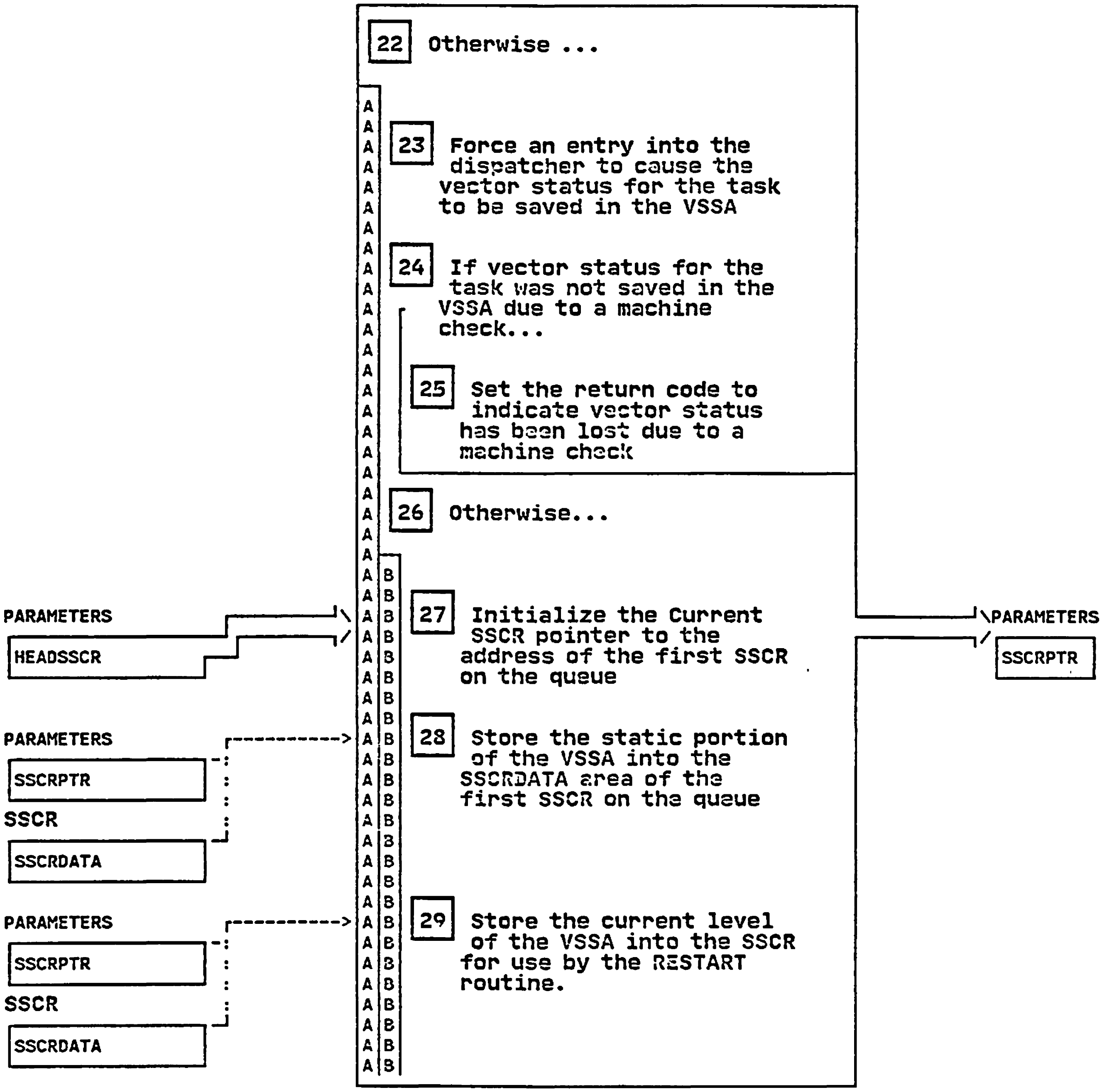
18 Clear the return code

19 Validate the pointer to the task's VSSA

20 If the STCBVSSA pointer does not point to a valid VSSA ...

21 Set the return code to indicate the pointer to the VSSA is invalid

IEAVCRVF - Vector Checkpoint/Restart Exit Routine STEP 22



22 Otherwise ...

23 Force an entry into the dispatcher to cause the vector status for the task to be saved in the VSSA

24 If vector status for the task was not saved in the VSSA due to a machine check...

25 Set the return code to indicate vector status has been lost due to a machine check

26 Otherwise...

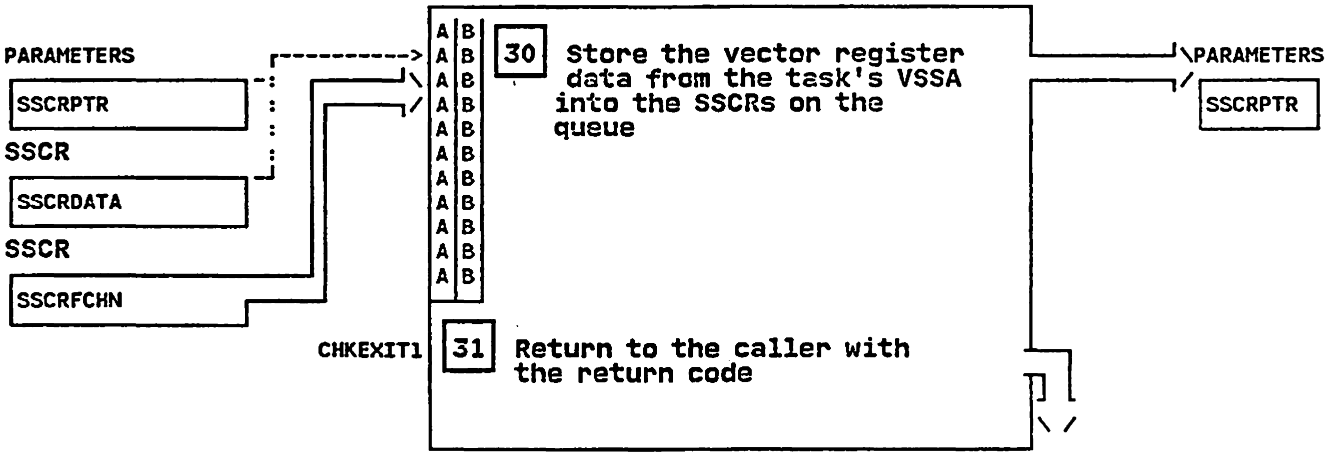
27 Initialize the Current SSCR pointer to the address of the first SSCR on the queue

28 Store the static portion of the VSSA into the SSCRDATA area of the first SSCR on the queue

29 Store the current level of the VSSA into the SSCR for use by the RESTART routine.

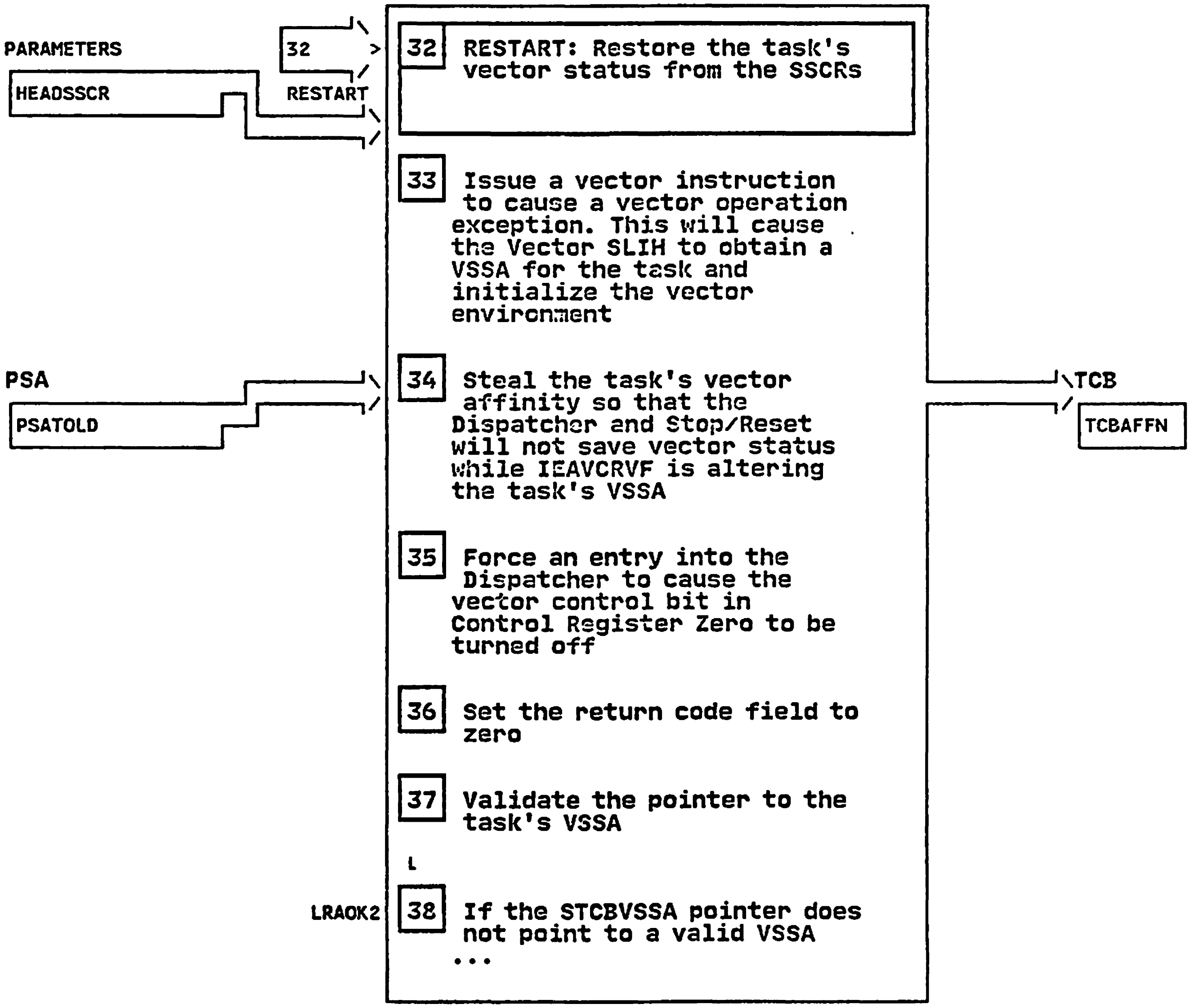
SUP-54. Vector Checkpoint/Restart Exit Routine (IEAVCRVF)

IEAVCRVF - Vector Checkpoint/Restart Exit Routine STEP 30



30 Store the vector register data from the task's VSSA into the SSCRs on the queue

31 Return to the caller with the return code



32 RESTART: Restore the task's vector status from the SSCRs

33 Issue a vector instruction to cause a vector operation exception. This will cause the Vector SLIH to obtain a VSSA for the task and initialize the vector environment

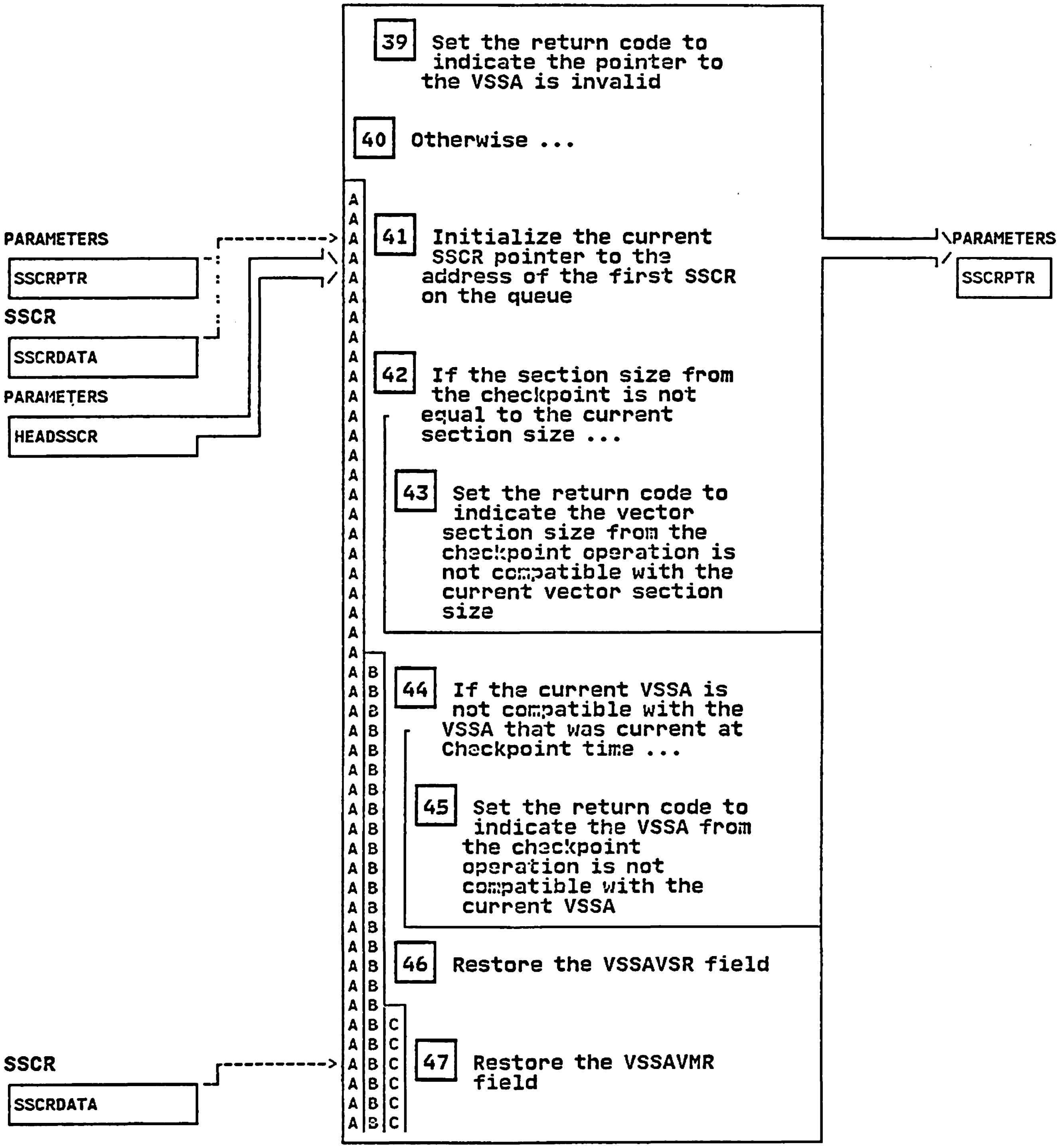
34 Steal the task's vector affinity so that the Dispatcher and Stop/Reset will not save vector status while IEAVCRVF is altering the task's VSSA

35 Force an entry into the Dispatcher to cause the vector control bit in Control Register Zero to be turned off

36 Set the return code field to zero

37 Validate the pointer to the task's VSSA

38 If the STCBVSSA pointer does not point to a valid VSSA
...



39 Set the return code to indicate the pointer to the VSSA is invalid

40 Otherwise ...

41 Initialize the current SSCR pointer to the address of the first SSCR on the queue

42 If the section size from the checkpoint is not equal to the current section size ...

43 Set the return code to indicate the vector section size from the checkpoint operation is not compatible with the current vector section size

44 If the current VSSA is not compatible with the VSSA that was current at Checkpoint time ...

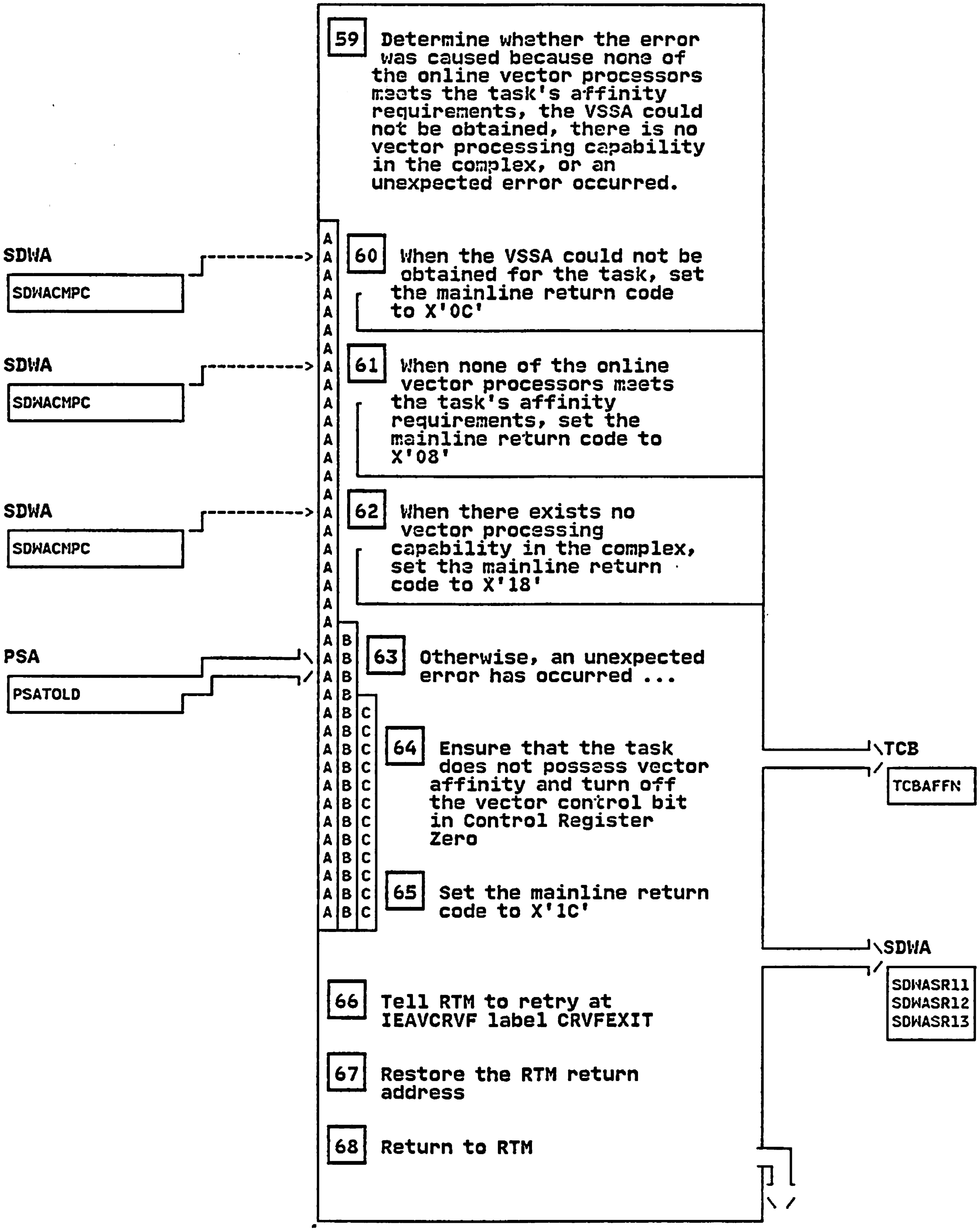
45 Set the return code to indicate the VSSA from the checkpoint operation is not compatible with the current VSSA

46 Restore the VSSAVSR field

47 Restore the VSSAVMR field

SUP-54. Vector Checkpoint/Restart Exit Routine (IEAVCRVF)

IEAVCRVF - Vector Checkpoint/Restart Exit Routine STEP 59



59 Determine whether the error was caused because none of the online vector processors meets the task's affinity requirements, the VSSA could not be obtained, there is no vector processing capability in the complex, or an unexpected error occurred.

60 When the VSSA could not be obtained for the task, set the mainline return code to X'0C'

61 When none of the online vector processors meets the task's affinity requirements, set the mainline return code to X'08'

62 When there exists no vector processing capability in the complex, set the mainline return code to X'18'

63 Otherwise, an unexpected error has occurred ...

64 Ensure that the task does not possess vector affinity and turn off the vector control bit in Control Register Zero

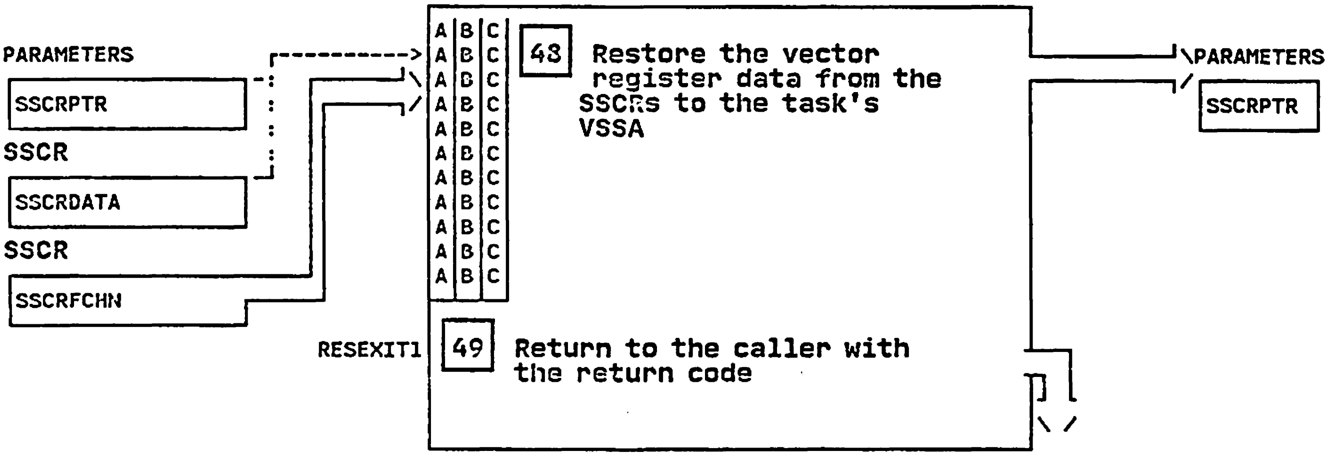
65 Set the mainline return code to X'1C'

66 Tell RTM to retry at IEAVCRVF label CRVFEXIT

67 Restore the RTM return address

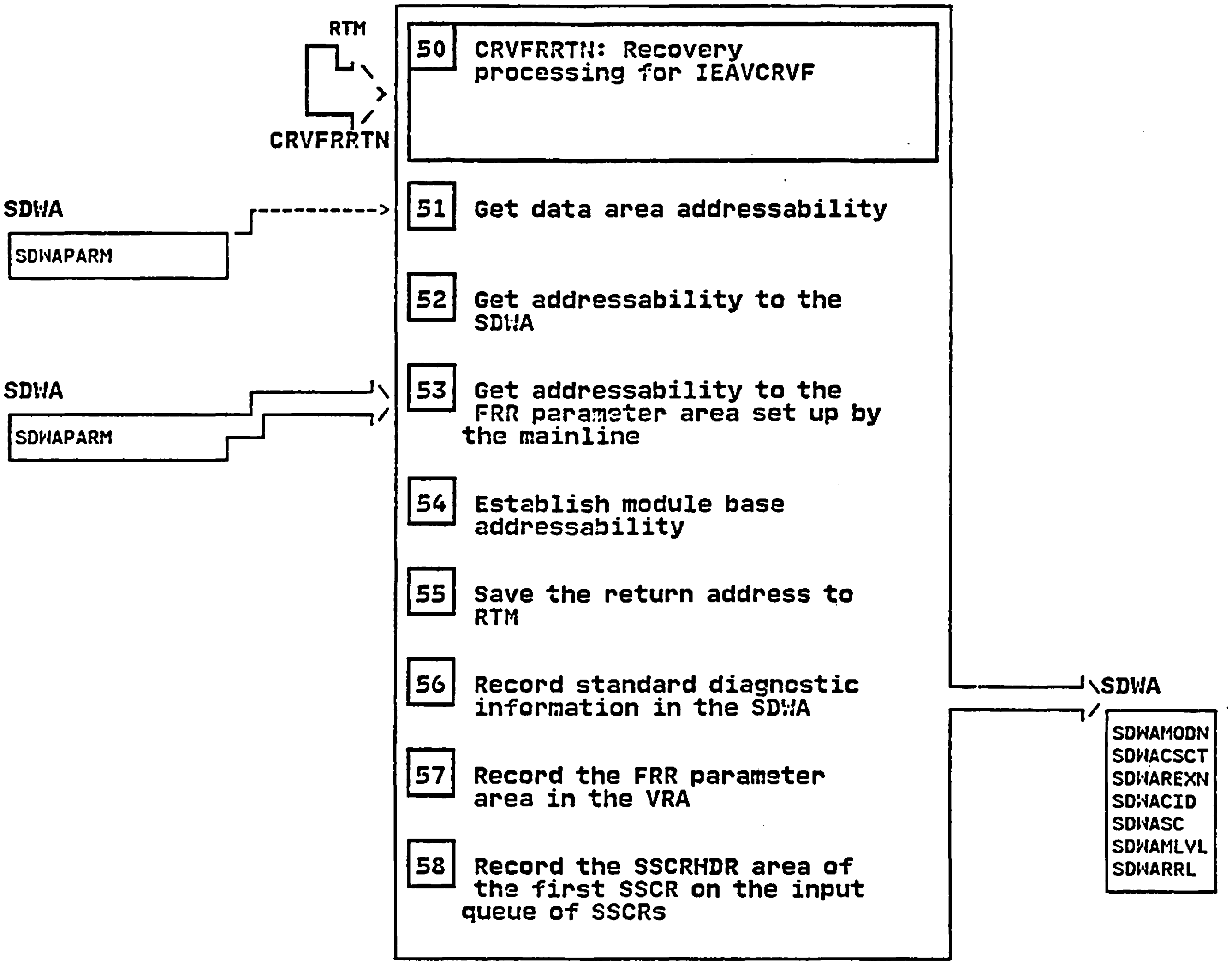
68 Return to RTM

IEAVCRVF - Vector Checkpoint/Restart Exit Routine STEP 48



48 Restore the vector register data from the SSCRs to the task's VSSA

49 Return to the caller with the return code



50 CRVFRRTN: Recovery processing for IEAVCRVF

51 Get data area addressability

52 Get addressability to the SDWA

53 Get addressability to the FRR parameter area set up by the mainline

54 Establish module base addressability

55 Save the return address to RTM

56 Record standard diagnostic information in the SDWA

57 Record the FRR parameter area in the VRA

58 Record the SSCRHDR area of the first SSCR on the input queue of SSCRs

SUP-55. VECTOR SECOND LEVEL INTERRUPT HANDLER (IEAVEVS)

IEAVEVS - MODULE DESCRIPTION

DESCRIPTIVE NAME: Vector Second Level Interrupt Handler

FUNCTION:
The vector SLIH receives control from the program FLIH upon detection of a program interrupt code x'19'. The SLIH is responsible for the creation and restoration of the vector environment of a task issuing vector instructions.

ENTRY POINT: IEAVEVS

PURPOSE:
This entry point is responsible for the creation and restoration of the vector environment of a task issuing vector instructions. For detailed description of the operation in this entry point, refer to the OPERATION section.

LINKAGE:
BASR R14,R15
Address of 18 word save area in R13

CALLERS: IEAVEPC (DAT-ON Program Check FLIH)

INPUT:
- ASCB referenced by PSAAOLD.
- ASSB referenced by ASCBASSB.
- TCB referenced by PSATOLD.
- STCB vector related status indicators.
- Online processor masks resident in the PCCA and CSD.
- Control Register Zero.
- Old Program check PSW resident in the LCCA.
- Current lock held string indicators in the PSA.
- SRB and task mode indicators in the PSA.

OUTPUT:
- The Vector Status Save Area (VSSA) initialized and chained from the STCB (If first time vector user).
- STCB vector related status indicators.
- Control Register Zero (bit 14) is set to one.
- Vector Activity Count saved in VSSAEVST.
- Return code in register 15.

EXIT NORMAL: To the Program Check FLIH with return code in R15.

EXIT ERROR:
To the Program Check FLIH with return code in R15 and associated reason code in R0.

ENTRY POINT: IEAVVRTM

PURPOSE:
IEAVVRTM clears the STCBVFRB and STCBPIQ fields to tell the SRB's (scheduled by the vector SLIH), RTM is in the process of abnormally terminating the task. When the vector SLIH schedules an SRB, the address of the current RB (the one STOPped by the vector SLIH), is placed in STCBVFRB. This field is compared against the STOPped RB in the SRB routine to ensure RTM processing has not made the task active. The SRB routine will perform the corresponding RESET only when the STOPped RB (passed in the SRBPARM area) matches the RB in STCBVFRB. RTM calls this entry point to zero the STCBVFRB field prior to making the task active. This mechanism allows the SRB routine to detect asynchronous ABTERM processing against the task. The STCBVFRB field is serialized between the vector SLIH and the SRB routine with the dispatcher lock. The field is serialized between the vector SLIH and the IEAVVRTM routine with TCBACTIV and disablement, (RTM STATUS STOPs the task for ABTERM

processing). The STCBVFRB is serialized between the SRB routine and IEAVVRTM with the local lock (RTM obtains home's local lock to initiate ABTERM processing).

LINKAGE:
BALR R14,R15
General registers 0-15 area saved in the PSASLSA
after entry to IEAVVRTM.

CALLERS: IEAVTRTM (RTM termination processing)

INPUT:
- TCB passed in register 3.
- STCB fields STCBVFRB and STCBPIQ.

OUTPUT: - STCBVFRB and STCBPIQ are cleared to zero.

EXIT NORMAL: To the caller with return code in R15.

EXIT ERROR: To the caller with return code in R15.

ENTRY POINT: IEAVVFGM

PURPOSE:
IEAVVFGM obtains and initializes the vector status save area (VSSA) for the first time vector user. For detailed description of the operation in this entry point, refer to the OPERATION section.

LINKAGE:
BASR R14,R15
Address of 18 word register save area in R13

CALLERS:
IEAVEVS (Vector Second Level Interrupt Handler)
IEAVVSRB (Vector Obtain VSSA SRB routine)

INPUT:
- Key fields in the Vector Work Area (VECWA).
- ASCB referenced by PSAAOLD.
- TCB passed in VFGMTCB@ within Vector Work Area.
- Vector section size contained in CVTVSS.

OUTPUT:
- The Vector Status Save Area (VSSA) initialized and chained from the STCB.
- Return code in register 15.

EXIT NORMAL: To the caller with return code in R15.

EXIT ERROR: To the caller with return code in R15.

EXTERNAL REFERENCES:

ROUTINES:

IEAVECMS	- (CMSET Routine)
IEAVELK	- (Spin Lock Manager)
IEAVESCO	- (SCHEDULE Service Routine)
IEAVESLK	- (Suspend Lock Manager)
IEAVESPM	- (SRB Pool Manager)
IEAVESRT	- (STOP/RESET)
IEAVGM00	- (GETMAIN - FREEMAIN Routine)
IEAVVSRB	- (Vector Environment Create SRB Routine)

DATA AREAS: IHAVECWA maps the SCWVSLIH area in the SCWA.

IEAVEVS - MODULE DESCRIPTION (Continued)

CONTROL BLOCKS:

ASCB	(Address Space Control Block)	- (r)
ASSB	(Address Space Secondary Block)	- (r,w)
CBLS	(Control Block Lengths Table)	- (r)
CSD	(Common System Data Area)	- (r)
CVT	(Communication Vector Table)	- (r)
FRRS	(Functional Recovery Routine Stack)	- (r,w)
LCCA	(Logical Communication Configuration Area)	- (r)
PCCA	(Physical Communication Configuration Area)	- (r)
PSA	(Prefixed Save Area)	- (r,w)
RB	(Request Block)	- (r,w)
SCWA	(Supervisor Control Work Area)	- (r,w)
SRB	(Service Request Block)	- (o,r,w)
STCB	(Secondary Task Control Block)	- (r,w)
SVT	(Supervisor Vector Table)	- (r)
TCB	(Task Control Block)	- (r,w)
VECWA	(Vector Work Area) maps the SCWVSLIH area	- (r,w)
VSSA	(Vector Status Save Area)	- (c,r,w)
XSB	(Extended Status Block)	- (r,w)

r = Read
w = Write
c = Create (GETMAIN is done in this module)
o = Obtain (Control block is obtained via service
routine call)

SERIALIZATION:

(For IEAVEVS entry point...)

- Obtains and releases Home's Local Lock for the VSSA GETMAIN processing.
- Obtains and releases the Dispatcher Lock to serialize fields ASSBVAFN, CSDCPUVF, and STCBVFRB.
- Sets STCBPIQ under TCBACTIV to prohibit IRB queuing by the STAGE 3 exit effector.
- Legally Disabled for I/O, EXTERNAL interrupts upon entry.

(For IEAVVRTM entry point...)

- Home's Local Lock held upon entry.
- Disabled for I/O, EXTERNAL interrupts.

(For IEAVVFGM entry point...)

- Legally Disabled for I/O, EXTERNAL interrupts upon entry.
- Home's Local Lock held upon entry.

IEAVEVS - MODULE OPERATION

(OPERATION FOR ENTRY POINT IEAVEVS) ...

After standard entry linkage is performed, the SLIH ensures that the processor is in task mode. If not, control is returned to the program FLIH with the appropriate return code. Addressability is then established to the home address space via CMSET to PSAAOLD. A CMSET RESET is performed prior to exit. The SLIH then validates the TCBSTCB pointer and decides whether to create a task's VF environment, restore the task's VF environment from the VSSA, or simply reset the vector control bit in control register zero.

CREATION OF THE VF ENVIRONMENT

When the first vector instruction is issued by a task, the SLIH knows to create the VF environment because the pointer to the VSSA (STCBVSSA) is zero. In this situation, the following restrictions apply:

- The task must be running enabled for I/O and external interrupts,
- The task must be unlocked.
- When VFs are online, there must exist at least one vector engine to which the task has affinity.
- VF capability must exist in the complex. That is, the VF hardware must be installed and a VF must have been online at some point during this ipl (although VFs may currently all be offline).

If any of these requirements are violated, control is returned to the program FLIH with the appropriate return code (reference the RETURN CODE section below). This path through the vector SLIH holds the Dispatcher lock to serialize the manipulation of the ASSBVAFN field. The lock insures the integrity of ASSBVAFN and prohibits Reconfiguration from taking a candidate vector engine off-line before the task has a chance to claim affinity to it. The SLIH then issues a conditional obtain for the local lock. If the lock is not obtained, the task is suspended via a call to STOP/RESET and a local SRB is scheduled (with the local lock and FRR required) to run on a CP/VF to create the task's vector environment. Module IEAVVSRB serves as the SRB routine. Also, STCBVFRB is set to the address of the STOPped RB to tell the SRB routine the task has been suspended. This communication is necessary since RTM may initiate ABTERM processing against the task which results in the task becoming active. Supervisor Control has provided RTM an entry point (IEAVVRTM) in this module which is called to zero the STCBVFRB field before the task becomes active. The SRB routine performs its function only when the contents of STCBVFRB equals the address of the STOPped RB. After the SRB is scheduled, control is passed back to the program FLIH with the appropriate return code to indicate task preemption. If the local lock is obtained, the SLIH performs the necessary calculations for the size of the vector status save area and issues a conditional branch entry GETMAIN for the storage. If the GETMAIN fails, control passes to the program FLIH with the appropriate return code. The VSSA is then chained to the STCB (STCBVSSA) and initialized. The task's affinity is then saved in field STCBAFNS and TCBAFFN is replaced with a more restrictive affinity. TCBAFFN must be set to set to a more restrictive affinity (a value that is nonzero and not all ones) so that the vector affinity checking path through the Dispatcher will be executed. The task's affinity will be recalculated in the Dispatcher using the value saved in STCBAFNS. Also, ASSBVAFN is incremented by one to indicate

there exists at least one task in the address space which has affinity to vector processors.
The SLIH then determines if the current processor meets the task's restrictive affinity requirements. If not, the resume PSW, registers, and cross memory status is saved. Control is returned to the program FLIH with a return code which indicates preemption for the task. If the task has affinity to the current processor, bit 14 in control register zero is set, vector status is loaded from the VSSA and control is routed back to the program FLIH with a return code to indicate the task can continue.

RESTORING THE VF ENVIRONMENT

This processing takes place when the task executes a vector instruction after its vector affinity was taken away because of vector facility non-use (STCBVSSA is not zero and STCBVAFN is zero). In this situation, the following restrictions apply:

- The task must be running enabled for I/O and external interrupts,
- The task must be unlocked.
- When VFs are online, there must exist at least one vector engine to which the task has affinity.
- VF capability must exist in the complex. That is, the VF hardware must be installed and a VF must have been online at some point during this ipl (although VFs may currently all be offline).

If any of these requirements are violated, control is returned to the program FLIH with the appropriate return code. The SLIH validates the pointer to the vector status save area with an LRA instruction and appropriate acronym checks. If the pointer is not valid, control is returned to the program FLIH with the appropriate return code. TCBAFFN is then replaced with a more restrictive affinity. The SLIH then determines if the current processor meets the task's affinity requirements. If not, the resume PSW, registers, and cross memory status is saved. Control is returned to the program FLIH with a return code which indicates preemption for the task. If the task has affinity to the current processor, bit 14 in control register zero is set, vector status is loaded from the VSSA and control is routed back to the program FLIH with a return code to indicate the task can continue.

RESETTING THE VECTOR CONTROL REGISTER BIT

In certain circumstances, the DAT-OFF program FLIH may refresh control register zero bit 14 with a default value of zero. Because the DAT-OFF program FLIH cannot establish the TCB addressability necessary for vector task recognition, the FLIH does not maintain the integrity of the vector control bit. This condition is detected because of a non-zero pointer to the VSSA (indicating a non-first time vector operation) and a non-zero STCBVAFN field (indicating the task has affinity to a