SY28-0764-0
File No. S370-36

Systems

OS/VS2 System Logic Library Volume 4

VS2.03.805
VS2.03.807

IBM

Pages numbered as duplicates in this publication must be retained because each of these documents information specific to individual Selectable Units.

This minor revision incorporates the following Selectable Units:

Supervisor Performance #1	**VS2.03.805**
Supervisor Performance #2	**VS2.03.807**

The selectable unit to which the information applies, is noted in the upper corner of the page.

First Edition (July, 1976)

This is a reprint of SY28-0716-0 incorporating changes released in the following Selectable Unit Newsletters:

SN28-2687	**(dated May 28, 1976)**
SN28-2693	**(dated May 28, 1976)**

This edition applies to Release 3.7 of OS/VS2 and to all subsequent releases of OS/VS2 until otherwise indicated in new editions or Technical Newsletters. Changes are continually made to the information herein; before using this publication in connection with the operation of IBM systems, consult the latest *IBM System/370 Bibliography*, GC20-0001, for the editions that are applicable and current.

Requests for copies of IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form for readers' comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, Publications Development, Department D58, Building 706-2, PO Box 390, Poughkeepsie, N.Y. 12602. Comments become the property of IBM.

System Logic Library comprises seven volumes. Following is the content and order number for each volume.

OS/VS2 System Logic Library,

Volume 1 contents: SY28-0713
  MVS logic introduction
  Abbreviation list
  Index for all volumes

Volume 2 contents: SY28-0714
  Method of Operation diagrams for
  Communications Task
  Command Processing
  Region Control Task (RCT)
  Started Task Control (STC)
  LOGON Scheduling

Volume 3 contents: SY28-0715
  Method of Operation diagrams for
  System Resources Manager (SRM)
  System Activity Measurement Activity (MF/1)
  JOB Scheduling
    —Subsystem Interface
    —Master Subsystem
    —Initiator/Terminator
    —SWA Create Interface
    —Converter/Interpreter
    —SWA Manager
    —Allocation/Unallocation
    —System Management Facilities (SMF)
    —System Log
    —Checkpoint/Restart

Volume 4 contents: SY28-0716
  Method of Operation diagrams for
  Timer Supervision
  Supervisor Control
  Task Management
  Program Management
  Recovery/Termination Management (R/TM)

Volume 5 contents: SY28-0717
  Method of Operation diagrams for
  Real Storage Management (RSM)
  Virtual Storage Management (VSM)
  Auxiliary Storage Management (ASM)

Volume 6 contents: SY28-0718
  Program Organization

Volume 7 contents: SY28-0719
  Directory
  Data Areas
  Diagnostic Aids

Please note that if you use only one order number, you will only receive that volume. To receive all seven volumes, you must either use all seven form numbers or, simply the following number: SBOF-8210. If you use SBOF-8210, you will receive all seven volumes.

The publication is intended for persons who are debugging or modifying the system. For general information about the use of the MVS system, refer to the publication *Introduction to OS/VS Release 2,* GC28-0661.

How This Publication is Organized

This publication contains six chapters. Following, is a synopsis of the information in each section:

- *Introduction and Master Index* — an overview of each of the functions this publication documents, an abbreviation list of all acronyms used in the publication, and a complete index for all seven volumes.
- *Method of Operation* — a functional approach to each of the subcomponents, using both diagrams and text. Each subcomponent begins with an introduction; all the diagrams and text applying to that subcomponent follow.
- *Program Organization* — a description of module-to-module flow for each subcomponent; a description of each module's function, including entry and exit. The module-to-module flow is ordered by subcomponent. The module descriptions are in alphabetic order without regard to subcomponent.
- *Directory* — a cross-reference from names in the various subcomponents to their place in the source code and in the publication.
- *Data Areas* — a description of the major data areas used by the subcomponents (only those, however, that are not described in *OS/VS Data Areas,* SYB8-0606, which is on microfiche); a data area usage table, showing whether a module reads or updates a data area; a control block overview diagram for each subcomponent, showing the various pointer schemes for the control blocks applicable to each subcomponent; a table detailing data area acronyms, mapping macro instructions, common names, and symbol usage table.

- *Diagnostic Aids* — the messages issued, including the modules that issue, detect, and contain the message; register usage; return codes; wait state codes; and miscellaneous aids.

Corequisite Reading

The following publications are corequisites:

- *OS/VS2 JES2 Logic,* SY28-0622
- *OS/VS Data Areas,* SYB8-0606 (This document is on microfiche.)
- *OS/VS2 System Initialization Logic,* SY28-0623

Contents

Figures

Section 2: Method of Operation

This section uses diagrams and text to describe the functions performed by the scheduler, supervisor, MF/1, SRM, and ASM functions of the OS/VS2 operating system. The diagrams emphasize functions performed rather than the program logic and organization. Logic and organization is described in "Section 3: Program Organization."

The method-of-operation diagrams are arranged by subcomponent as follows:

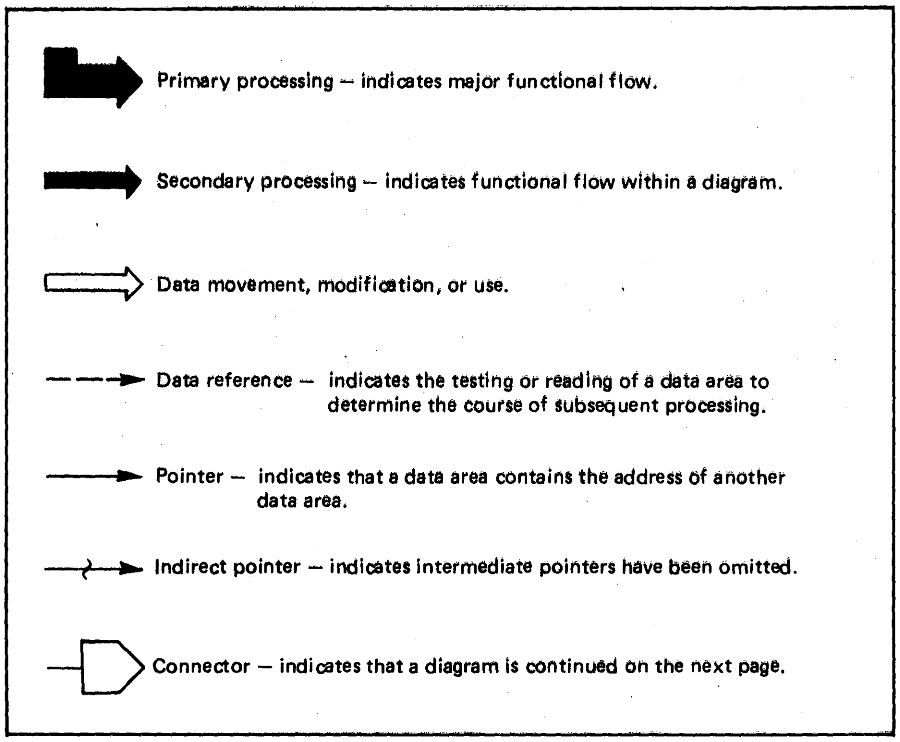
- Communications Task.
- Command Processing (includes Reconfiguration Commands).
- Region Control Task (RCT).
- Started Task Control (STC) (includes START/LOGON/MOUNT).
- LOGON Scheduling
- System Resources Manager
- System Activity Measurement Facility (MF/1)
- Job Scheduling:
  - Subsystem Interface.
  - Master Subsystem.
  - Initiator/Terminator.
  - SWA Create Interface.
  - Converter/Interpreter.
  - SWA Manager.
  - Allocation/Unallocation.
  - System Management Facilities (SMF).
  - System Log.
  - Checkpoint/Restart.
- Timer Supervision.
- Supervisor Control.
- Task Management.
- Program Management.
- Recovery/Termination Management (R/TM).
- Real Storage Management (RSM).
- Virtual Storage Management (VSM).
- Auxiliary Storage Management (ASM).

The diagrams for each subcomponent are preceded by an introduction that summarizes the subcomponent's function. Following each introduction is a visual table of contents that displays the organization and hierarchy of the diagrams for that subcomponent.

The diagrams cross-reference each other using diagram numbers and module names. As an aid in locating the diagrams that are cross-referenced, an alphabetic list of all diagram names and their corresponding page numbers follows this introduction.

Method-of-operation diagrams are arranged in an input-processing-output format: the left side of the diagram contains data that serves as input to the processing steps in the center of the diagram, and the right side contains the data that is output from the processing steps. Each processing step is numbered; the number corresponds to an amplified explanation of the step in the "Extended Description" area. The object module name and labels in the extended description point to the code that performs the function.

Note: The relative size and the order of fields within input and output data areas do not always represent the actual size and format of the data area.



Primary processing — indicates major functional 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 2-1. Key to Symbols Used in Method-of-Operation Diagrams

Timer Supervision

The timer supervision routines support the System/370 time-of-day clock, clock comparator, and CPU timer. The routines use these components to obtain the time of day and the date, schedule activity after a specified interval, and schedule activity after a specified time of day. Other timer routines set the time-of-day clock and synchronize the TOD clocks in a multiprocessing system.

For TIME macro instructions, the TIME routine returns the date and time of day to the requester.

For STIMER macro instructions, the STIMER routine sets a requested time interval that expires after the specified time has elapsed or at the specified time of day. When the requested time interval expires, a timer or clock comparator interruption occurs and the Timer Second Level Interruption Handler processes it. If the requester specifies the task timing option, the time interval is decreased only when the requester's task is active. If the requester specifies wait timing, his task is placed in a wait state until the time interval expires. If the requester specifies real timing, the time interval is decreased continuously.

For TTIMER requests, the TTIMER routine returns the amount of time remaining in an interval previously set by a STIMER macro instruction. The routine can also cancel the remaining time interval if so requested.

Timer supervision also provides a SETDIE routine that allows system programs (programs executing in supervisor state and with a protect key of 0) to specify a real time interval, after which a disabled interrupt exit (DIE) is to be given control. With SETDIE, the system program supplies timer supervision with a pre-built TQE. When the timer SLIH processes such a TQE (called a DIE TQE), it gives control directly to the specified exit routine (DIE).

Timer supervision maintains two queues of TQEs (timer queue elements): one for task timing requests, pointed to from the TCB of the reqeusting task and containing only one TQE at a time; and, one for real and wait timing requests, pointed to from the TPC (timer work area) and containing all real wait type TQEs in the system. TQEs (other than a DIE TQE) are constructed by the STIMER routine, and each element represents a request for a timed interval. Each new TQE is placed on the appropriate queue in the order in which the requested interval expires. When an interval expires, a timer interruption occurs. The Timer Second Level Interruption Handler removes the top TQE from the appropriate queue and determines what action to take.

Other timer routines provide for the initialization of the TOD clock at IPL (see *OS/VS2 System Initialization Logic,* SY28-0623) and when a CPU is being varied online, and the resetting or resynchronizing of a TOD clock that has suffered a machine check.

In either case, the Set Specific Clock routine searches for a TOD clock in the system to which the new or error TOD clock in the system to which the new or error TOD clock can be synchronized. If one is found, the synchronization is done and the results are validated. If no other valid TOD clock is found, a routine is entered that ensures that the TOD clock will be set with the correct value.

A set of service routines provide common services to the timer supervision functions. TQE Enqueue and TQE Dequeue provide for the movement of TQEs to or from the timing queues. TQE Purge purges all timer TQEs and SRBs during task termination. An FRR and two hardware recovery routines are also included.

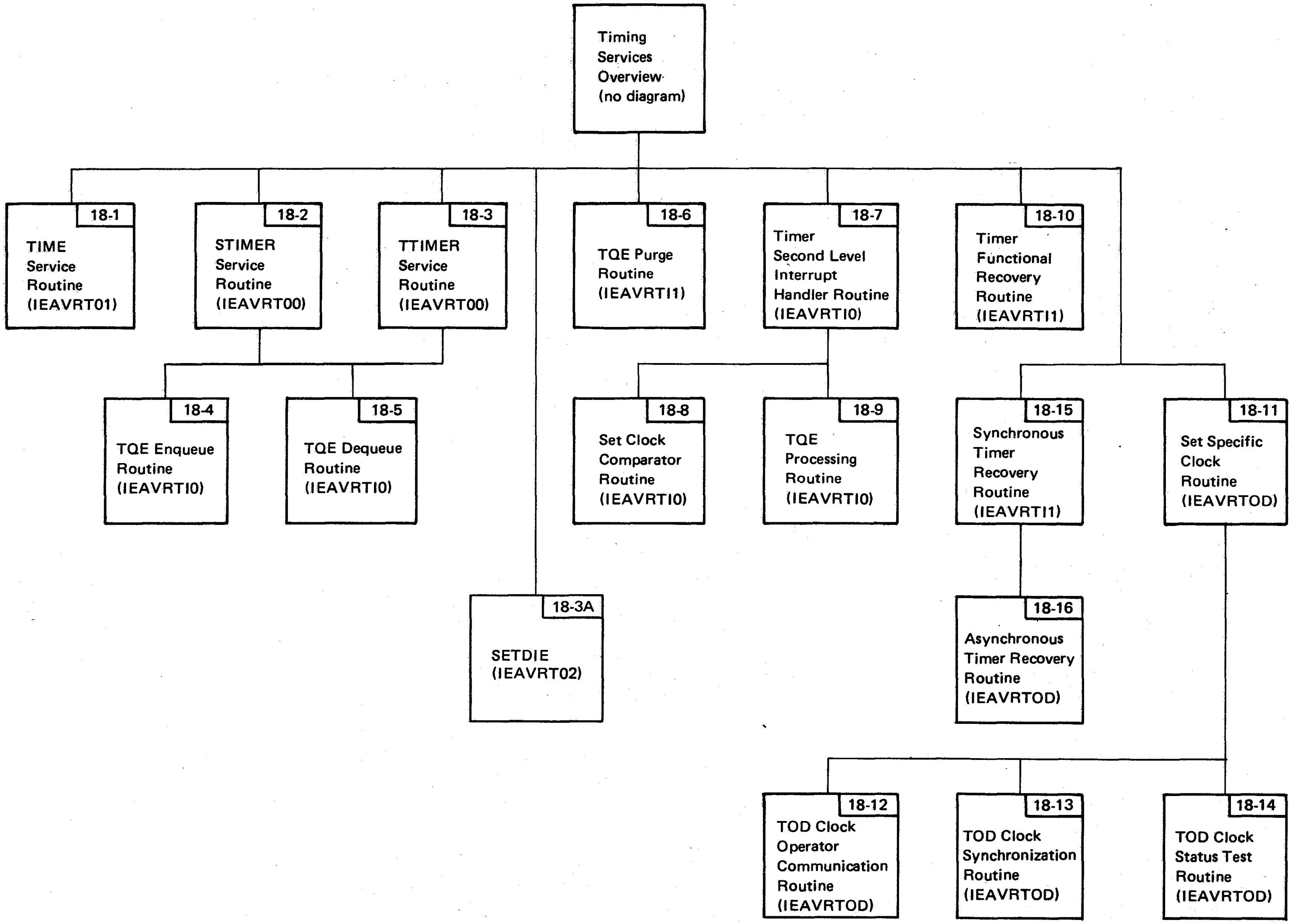

Timing Services Overview (no diagram)
18-1 TIME Service Routine (IEAVRT01)
18-2 STIMER Service Routine (IEAVRT00)
18-3 TTIMER Service Routine (IEAVRT00)
18-6 TQE Purge Routine (IEAVRTI1)
18-7 Timer Second Level Interrupt Handler Routine (IEAVRTI0)
18-10 Timer Functional Recovery Routine (IEAVRTI1)
18-4 TQE Enqueue Routine (IEAVRTI0)
18-5 TQE Dequeue Routine (IEAVRTI0)
18-8 Set Clock Comparator Routine (IEAVRTI0)
18-9 TQE Processing Routine (IEAVRTI0)
18-15 Synchronous Timer Recovery Routine (IEAVRTI1)
18-11 Set Specific Clock Routine (IEAVRTOD)
18-3A SETDIE (IEAVRT02)
18-16 Asynchronous Timer Recovery Routine (IEAVRTOD)
18-12 TOD Clock Operator Communication Routine (IEAVRTOD)
18-13 TOD Clock Synchronization Routine (IEAVRTOD)
18-14 TOD Clock Status Test Routine (IEAVRTOD)

Figure 2-32. Timer Supervision Visual Contents (Part 1 of 2)

difference between local & GMT
in units of 1.048576 sec

Diagram 18-1. TIME Service Routine (IEAVRT01) (Part 1 of 2)

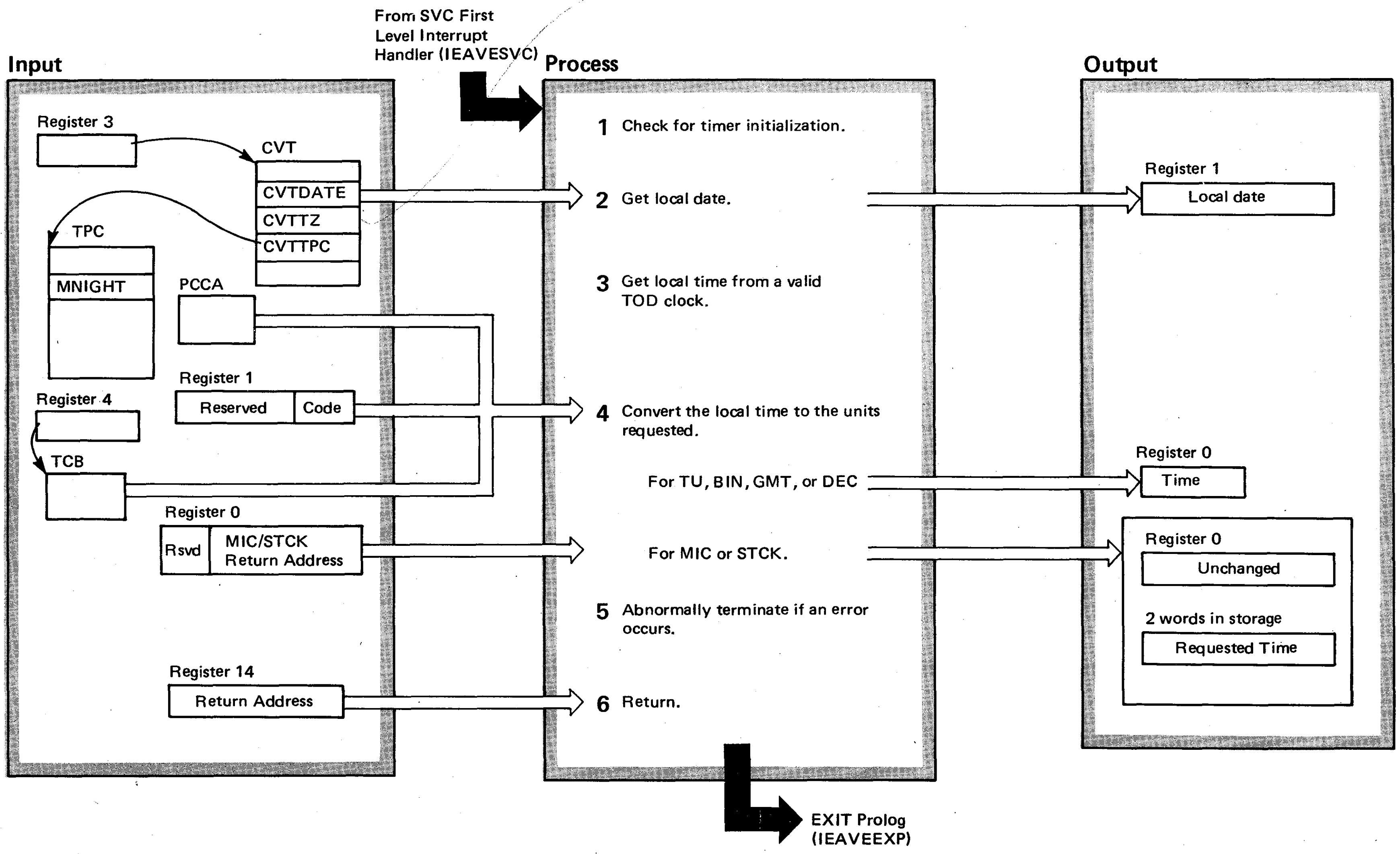


Diagram 18-1. TIME Service Routine (IEAVRT01) (Part 2 of 2)

Extended Description	Module	Label
The TIME routine (IEAVRT01) services TIME macro instructions. It obtains the local date, calculates the local time of day, and returns both to the caller as specified in the macro instruction.		
1 TIME checks the midnight value in the TPC. If it is zero, initialization has not taken place. TIME sets register 0 to zero and sets register 1 to 15.	IEAVRT01	IGC0001A
2 TIME gets the date and time zone constant from the CVT. It stores the date in register 1.		
3 TIME stores the TOD clock. If the operation fails, TIME gets the correct TOD clock setting from another clock in the system. If the time cannot be obtained, TIME puts an 8 in register 15.	IEAVRT01	OTHERCLK
4 TIME converts the local time into the units requested and stores it in either register 0 or, if STCK or MIC is specified, in the user-specified storage area. For STCK and MIC requests, if the caller is not in a system protect key (key 0–7), TIME checks the validity of the user-specified storage area key. (The key of the storage area passed by the caller should match the caller's TCB protection key.) If the check fails, TIME so indicates by placing '12' in reg 15.	IEAVRT01	IGC0001A
5 If the validity check in step 4 failed ('12' in reg 15), TIME abnormally terminates the caller's task with a code of X '10b'. If, however, the caller's protect key is valid but the TIME request failed for another reason, TIME checks for an error return address. If no error return address is found, it abnormally terminates the user with a code of X'20B'.		
6 If ERRET has been specified and no usable TOD clock was found (reg 15 = '08'), or the request was successful (reg 15 = 0), TIME gives control through reg 14 to the EXIT routine.		

Diagram 18-2. STIMER Service Routine (IEAVRT00) (Part 1 of 2)

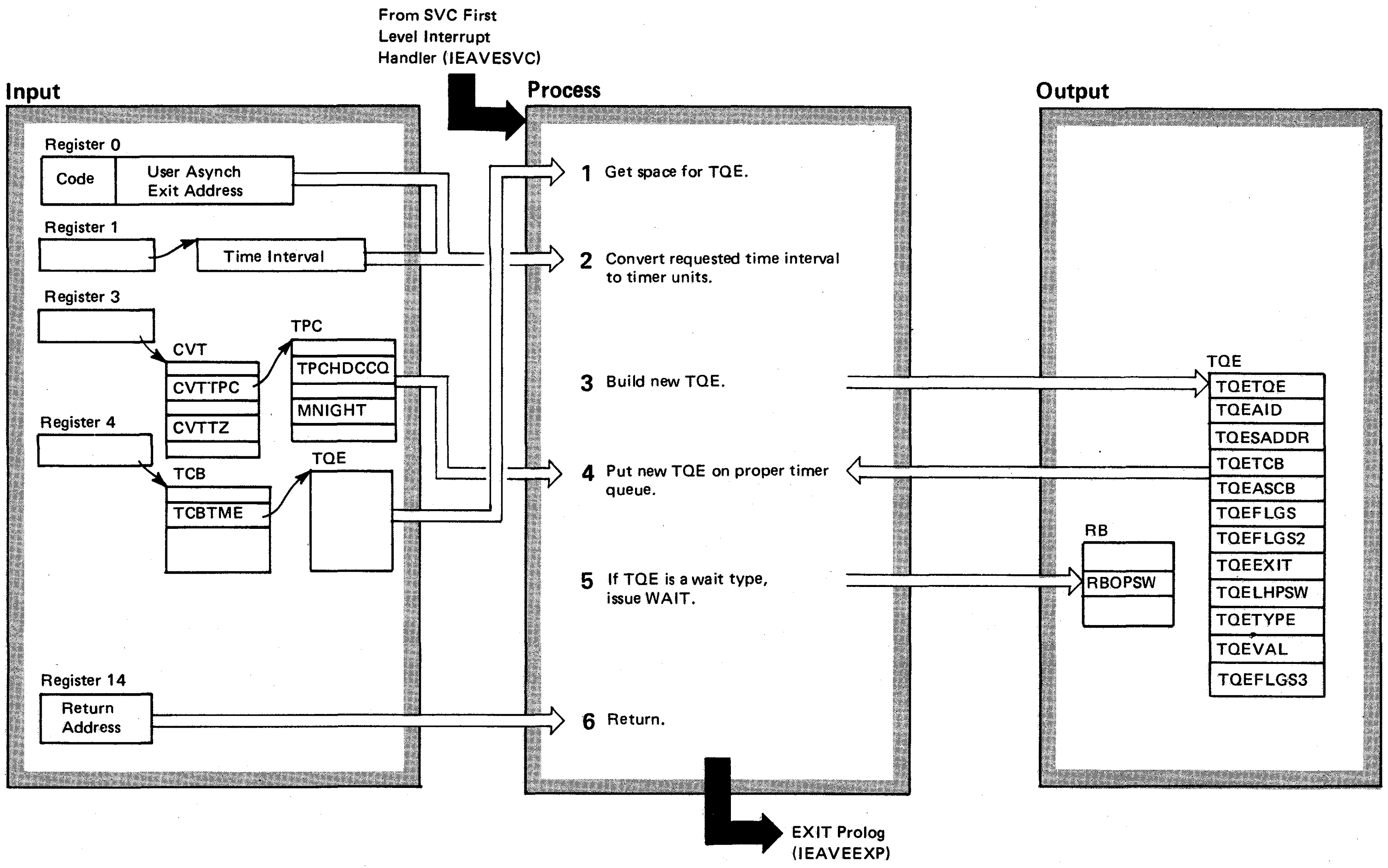
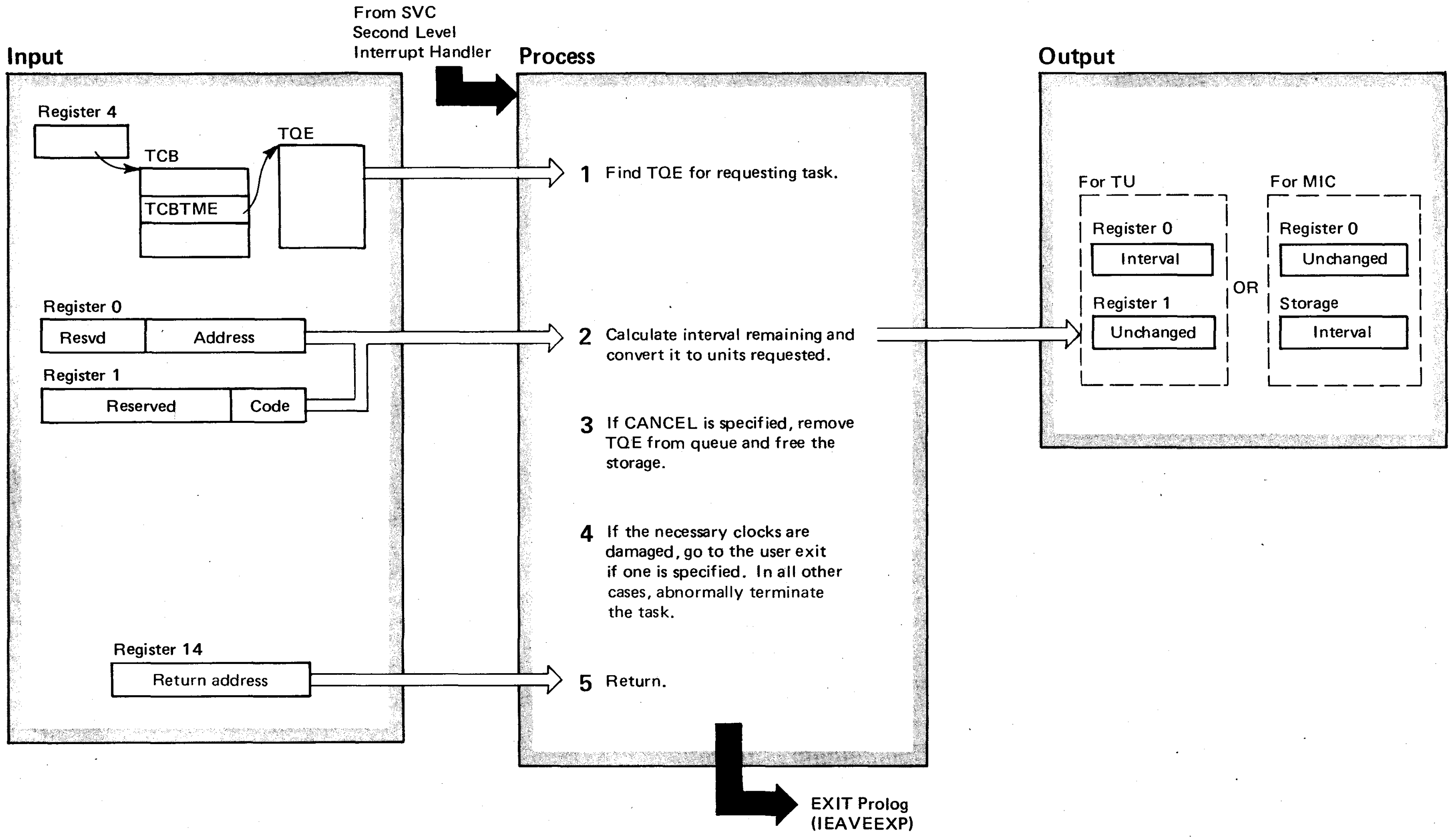


Diagram 18-2. STIMER Service Routine (IEAVRT00) (Part 2 of 2)

Extended Description	Module	Label
The STIMER routine (IEAVRT00) processes STIMER macro instructions. The routine sets a time interval which expires after a specified time has elapsed or at a specified time of day.		
1 STIMER checks for existing TQEs and dequeues them. If the type of the old TQE is the same as the type requested, STIMER indicates that old TQE space may be used. Otherwise, old TQE space is freed. Then STIMER obtains space in LSQA for a task TQE or from SQA for a real type TQE.	IEAVRT00	IGC0004G
2 STIMER checks the type of request. If it is a task type request with TOD or GMT specified, a 4 is set in register 15. If it is a real or wait request, STIMER stores the TOD clock.		
If that fails, it attempts to set the TOD clock value with the TIME routine. If it fails, STIMER sets register 15 to 8. If the time is obtained, STIMER converts it to the units requested. If GMT or TOD requests specify more than 2400, a 4 is placed in register 15.	IEAVRT01	IGC0001A

Extended Description	Module	Label
3 STIMER initializes the TQE identifier field, the ASID, the TCB address, and the ASCB address fields in the newly obtained storage area. Then it builds the common portion of the TQE.		
4 STIMER determines whether the timing components necessary to service the request are usable. If they are not, it puts an 8 in register 15. If the necessary components are usable, STIMER enqueues the new TQE on either the real or the task timing queue.	IEAVRT00	IEAVQTE00
5 If this is a wait request, STIMER passes control to the WAIT routine.		
6 If a 4 is set in register 15, the user is abnormally terminated with a code of X'12F'. If an 8 is set in register 15 and no error exit address was specified, STIMER abnormally terminates the user with a code of X'22F'. Otherwise, STIMER passes control to the EXIT routine.		
Error Processing When an STIMER error is passed from R/TM, the ESTAE routine records error information in the SDWA, if one exists. Then, it passes control to R/TM to record the error and to continue with termination.	IEAVRT00	TTSTSTAE

Diagram 18-3. TTIMER Service Routine (IEAVRT00) (Part 1 of 2)



1 Find TQE for requesting task.

2 Calculate interval remaining and convert it to units requested.

3 If CANCEL is specified, remove TQE from queue and free the storage.

4 If the necessary clocks are damaged, go to the user exit if one is specified. In all other cases, abnormally terminate the task.

5 Return.

Diagram 18-3. TTIMER Service Routine (IEAVRT00) (Part 2 of 2)

Extended Description	Module	Label
The TTIMER routine (IEAVRT00) processes TTIMER macro instructions. The routine calculates the time remaining in a timer interval previously set by STIMER; optionally, it cancels the interval. The time remaining in the interval is returned to the caller as specified in the macro instruction.		
1 TTIMER checks a TCB field to find a TQE. If none is found, TTIMER sets the time interval to zero.	IEAVRT00	IGC0004F
2 If a TQE is found, TTIMER converts the interval to the units specified for the type of request. If the time cannot be obtained, TTIMER puts an 8 in register 15.		
3 If CANCEL has been specified, TTIMER dequeues the TQE and frees the TQE space by using FREEMAIN.	IEAVRTI0	IEAQTD00
4 TTIMER checks register 15 for an error condition. If one is found, it checks for a user error exit address. If the user exit address is specified, control is passed to that address. If no error exit is specified, TTIMER abnormally terminates the user with a code of X'22E'.		
5 If no error has occurred, TTIMER returns control through register 14 to the EXIT routine.		
Error Processing When an error is passed from R/TM, the ESTAE routine initializes the SDWA, if one is present. Then it returns control to R/TM for error recording and further termination processing. If the error is due to storing into a user area, the user is abnormally terminated with a code of X'12E'.	IEAVRT00	TTSTSTAE

Diagram 18-3A. SETDIE Routine (IEAVRT02) (Part 1 of 2)

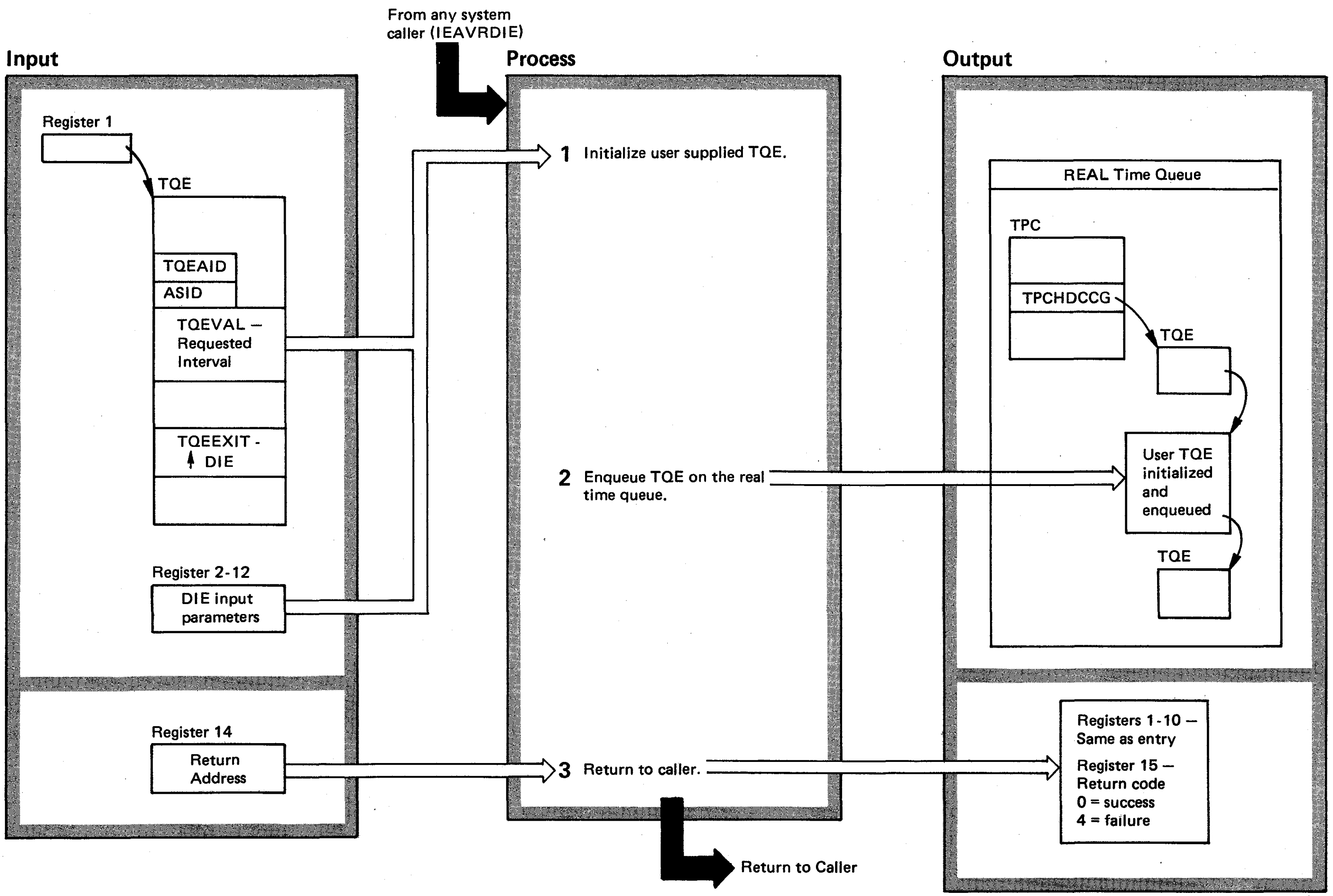
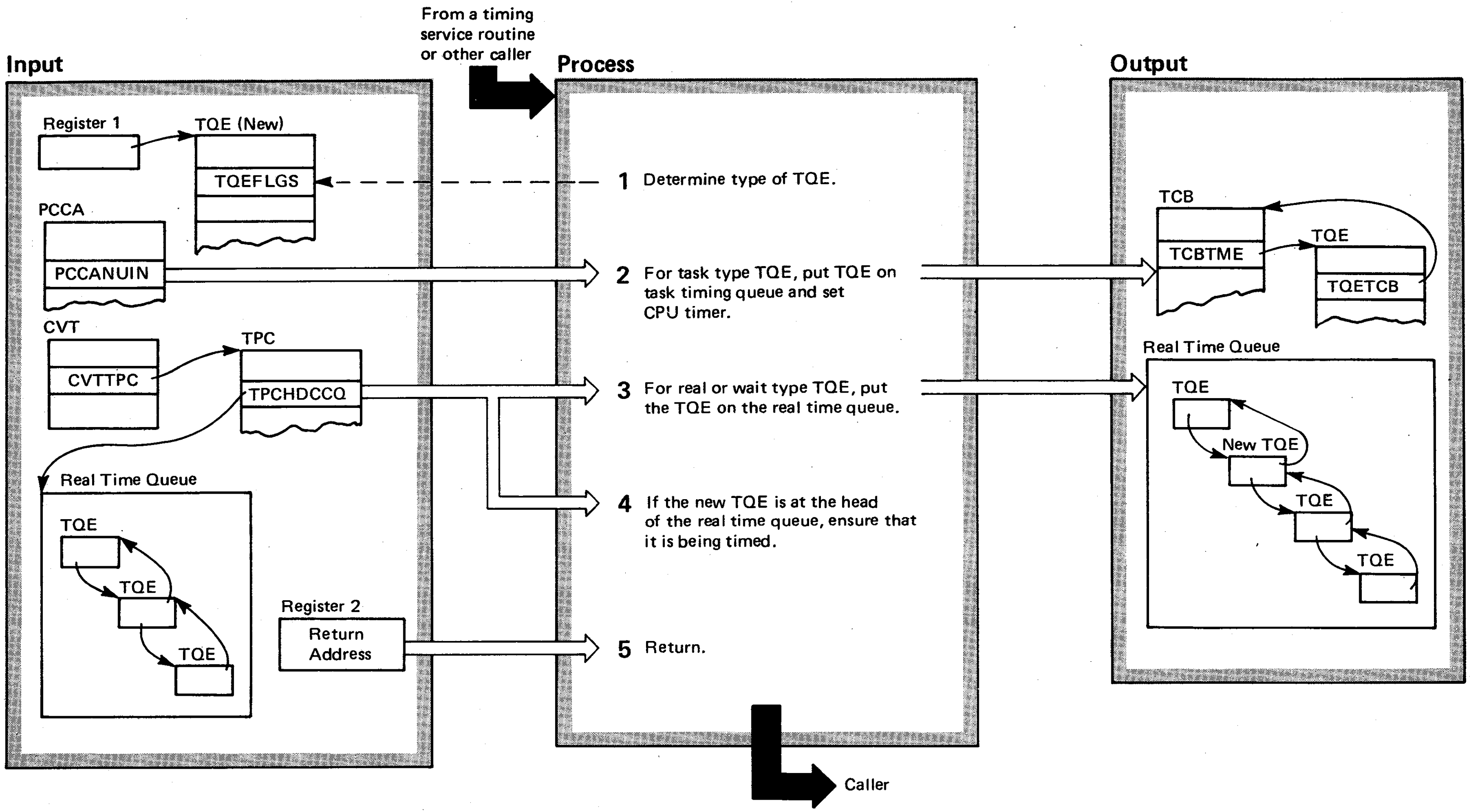

From any system caller (IEAVRDIE)
Input
Register 1
TQE
TQEAID
ASID
TQEVAL — Requested Interval
TQEEXIT - DIE
Register 2-12
DIE input parameters
Register 14
Return Address
Process
1 Initialize user supplied TQE.
2 Enqueue TQE on the real time queue.
3 Return to caller.
Return to Caller
Output
REAL Time Queue
TPC
TPCHDCCG
TQE
User TQE initialized and enqueued
TQE
Registers 1-10 — Same as entry
Register 15 — Return code
0 = success
4 = failure

Diagram 18-3A. SETDIE Routine (IEAVRT02) (Part 2 of 2)

Extended Description	Module	Label
The SETDIE routine (IEAVRT02) enqueues a user supplied TQE on the system's real time queue.		
1 The following fields in the user supplied TQE are initialized: TQETQE (TQE ID word) TQEFLGS (X'87') TQEFLGS2 (X'40') TQEFLGS3 (X'80') TQEVAL (clock comparator value) TQEREGS (DIE parameter registers)	IEAVRT02	IEAVRDIE
2 The completed TQE is then enqueued on the system's real time queue using the timer supervision enqueue routine.	IEAVRTI0	IEAQTE00
3 SETDIE returns to the caller via register 14.		

Diagram 18-4. TQE Enqueue Routine (IEAVRTI0) (Part 1 of 2)



1 Determine type of TQE.

2 For task type TQE, put TQE on task timing queue and set CPU timer.

3 For real or wait type TQE, put the TQE on the real time queue.

4 If the new TQE is at the head of the real time queue, ensure that it is being timed.

5 Return.

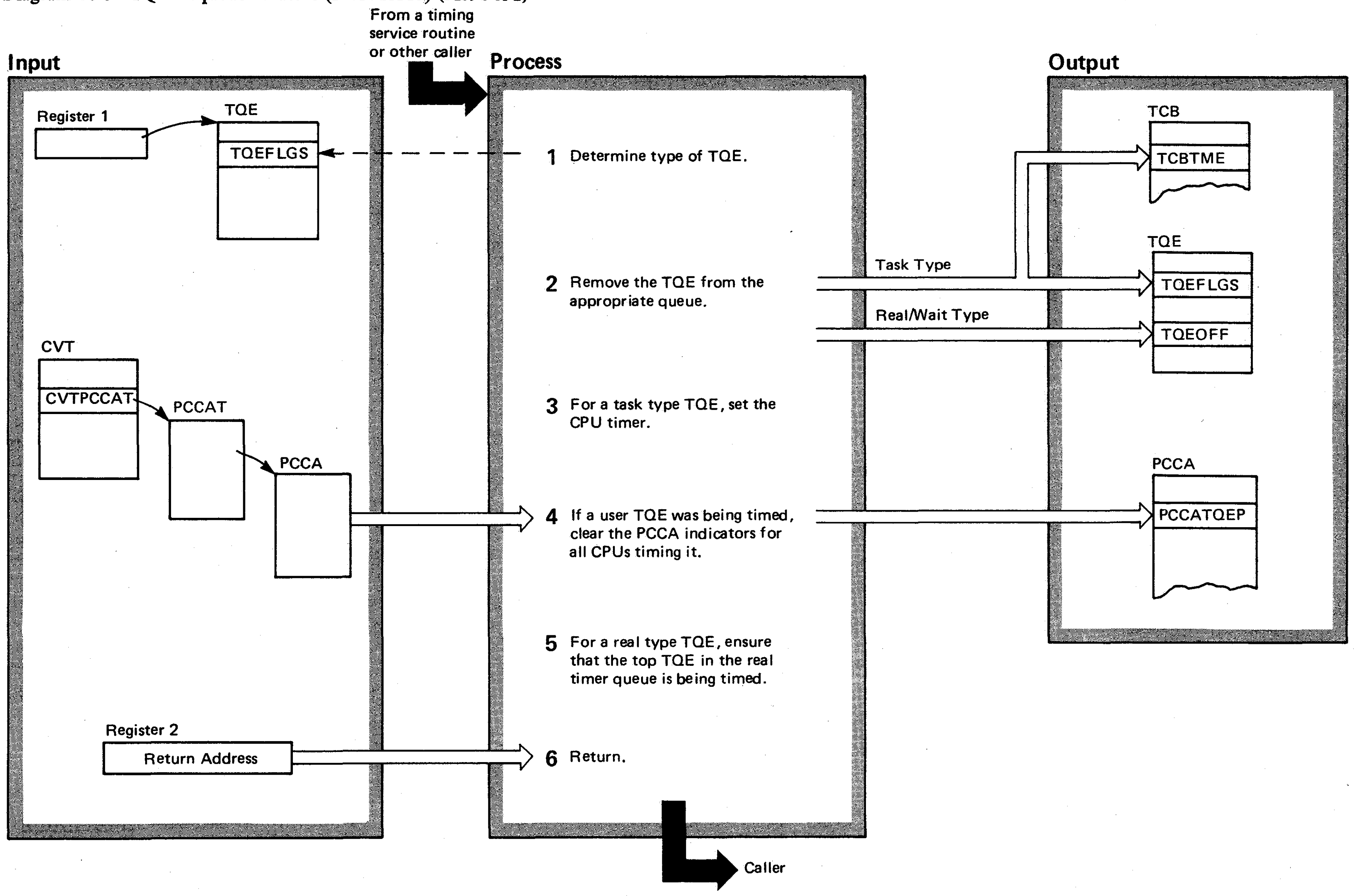
Diagram 18-4. TQE Enqueue Routine (IEAVRTI0) (Part 2 of 2)

Extended Description	Module	Label
The TQE Enqueue routine (IEAVRTI0) enqueues the subject TQE (Timer Queue Element) on the proper timing queue: the real time queue for real or wait type TQEs; or the task queue for a task type TQE.		
1 TQE Enqueue determines from the TQETYPE field what type of TQE is being processed.	IEAVRTI0	IEAQTE00

Field Setting	TQE Type
00	Task
01	Wait
11	Real

Extended Description	Module	Label
2 TQE Enqueue enqueues a task type TQE from a TCB by setting the TCBTME field and setting the TQE fields to indicate the TCB with which the TQE is associated, and to indicate that the TQE is on a timer queue. Then it sets the CPU timer and, for recovery purposes, saves the time-of-day when the CPU timer was set.		
3 TQE Enqueue sets a TCB flag to indicate a real or wait type TQE. This flag is not set, however, for a DIE TQE. Then it searches the real time queue and places the subject TQE in the proper place. It indicates that the TQE is on the real timer queue.		
4 TQE Enqueue verifies that the top TQE in the real timing queue is being timed and if not, sets the clock comparator.	IEAVRTI0	SETCC
5 TQE Enqueue returns control to the caller.		

Diagram 18-5. TQE Dequeue Routine (IEAVRTI0) (Part 1 of 2)



1 Determine type of TQE.

2 Remove the TQE from the appropriate queue.

3 For a task type TQE, set the CPU timer.

4 If a user TQE was being timed, clear the PCCA indicators for all CPUs timing it.

5 For a real type TQE, ensure that the top TQE in the real timer queue is being timed.

6 Return.

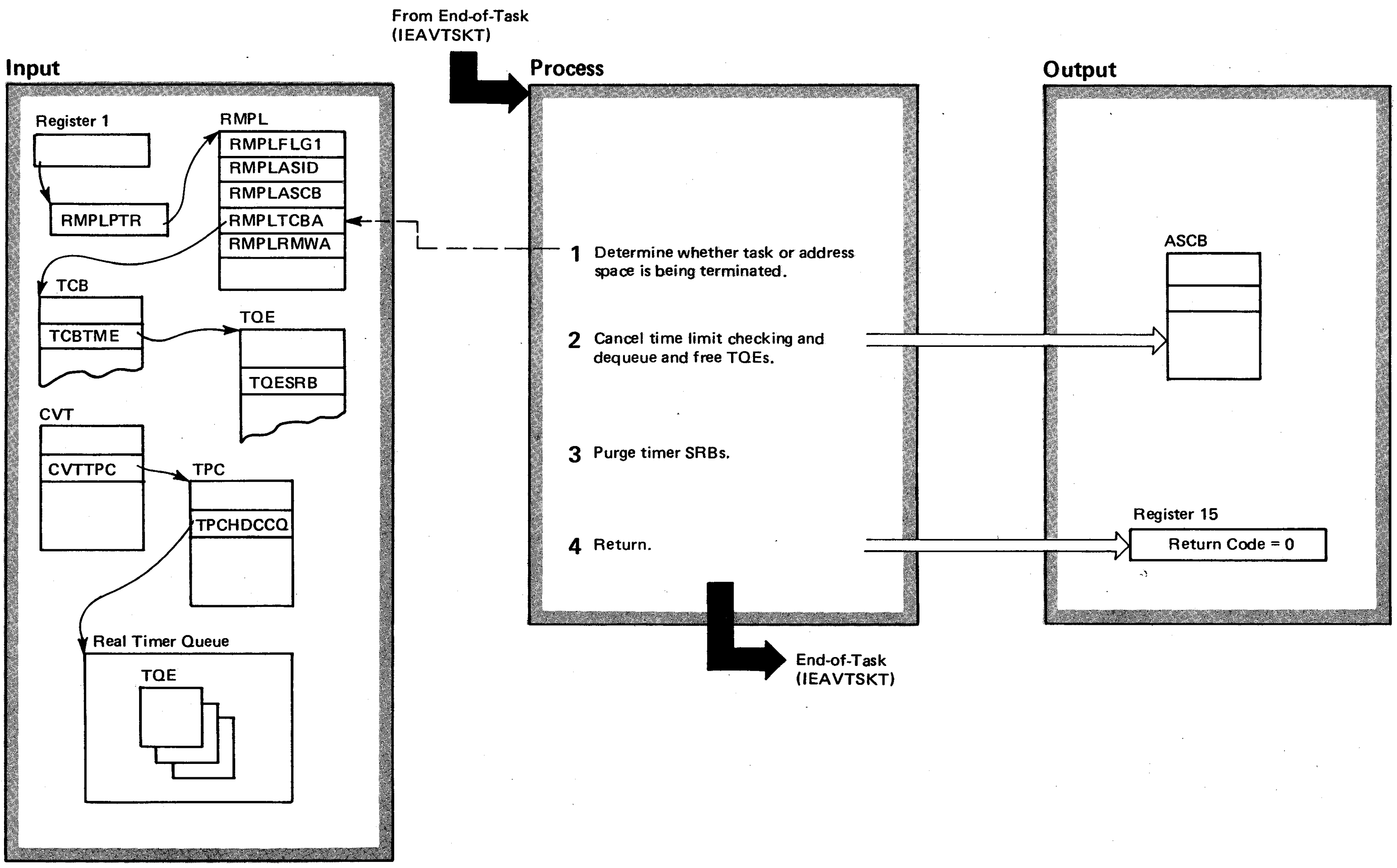
Diagram 18-5. TQE Dequeue Routine (IEAVRTIO) (Part 2 of 2)

Extended Description	Module	Label
The TQE Dequeue routine (IEAVRTIO) dequeues the subject TQE from the real time queue for a real or wait type TQE, or from the task queue for a task type TQE.		
1 TQE Dequeue checks the TQETYPE field in the TQE to determine its type.	IEAVRTIO	IEAVQTD00

Field Setting	TQE Type
00	Task
01	Wait
11	Real

Extended Description	Module	Label
2 TQE Dequeue resets the pointers to the TQE, and marks the TQE to indicate that the TQE is off a timing queue.		
3 TQE Dequeue sets the CPU Timer to a high value to insure against timer interruptions.		
4 TQE Dequeue clears fields in the PCCA (Physical Configuration Communication Area) entries to indicate that the TQE is no longer being timed.		
5 TQE Dequeue verifies that the top TQE in the real timing queue is being timed.	IEAVRTIO	SETCC
6 TQE Dequeue returns control to the caller.		

Diagram 18-6. TQE Purge Routine (IEAVRTI1) (Part 1 of 2)



1 Determine whether task or address space is being terminated.

2 Cancel time limit checking and dequeue and free TQEs.

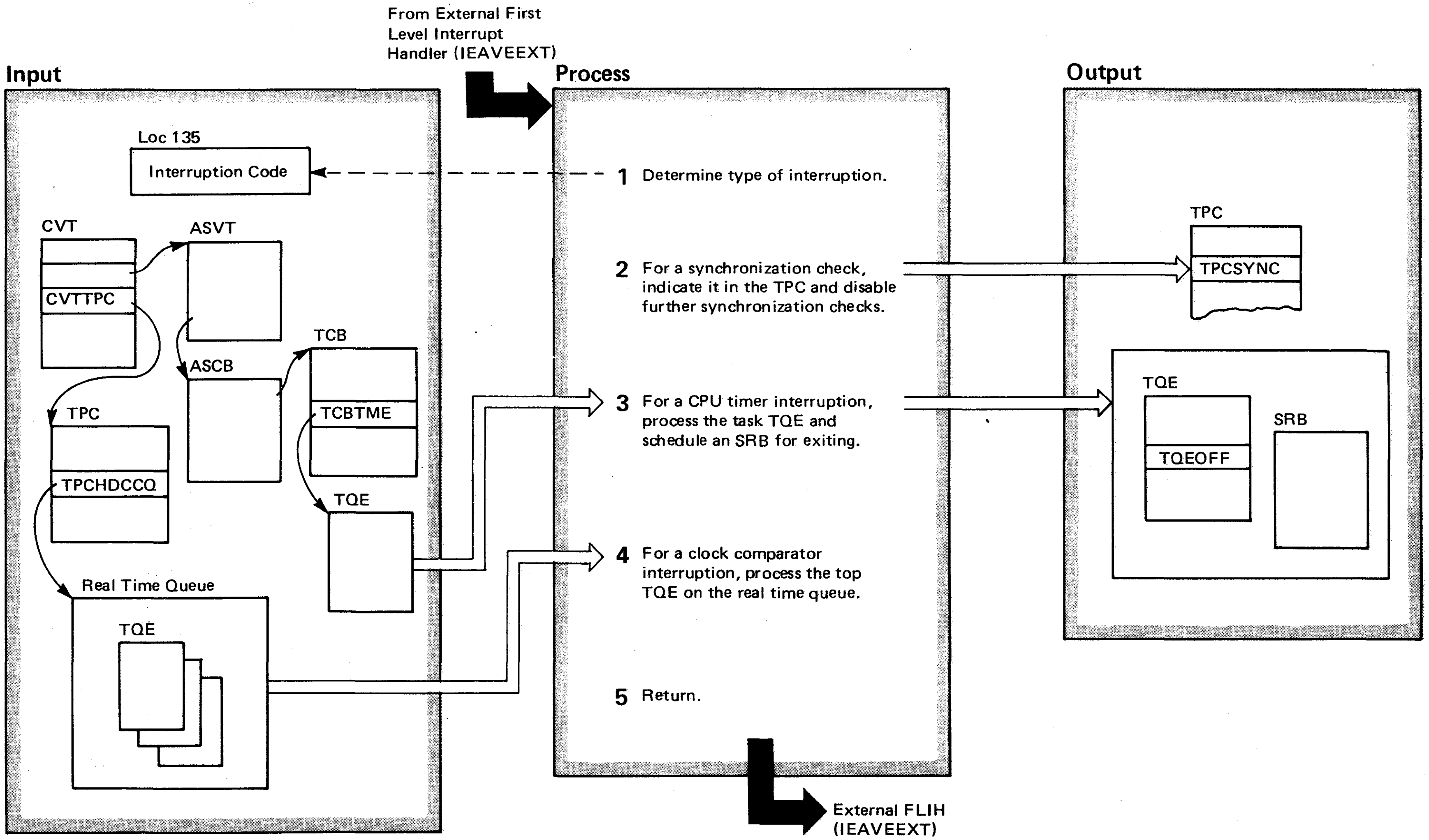
3 Purge timer SRBs.

4 Return.

Diagram 18-6. TQE Purge Routine (IEAVRTI1) (Part 2 of 2)

Extended Description	Module	Label
The TQE Purge routine (IEAVRTI1), entered when a task or address space terminates, purges all TQEs for the task or address space. In addition, the routine purges all timer SRBs that have not yet been scheduled for the specified task or address space.		
1 TQE Purge tests a field in the RMPL to determine whether an address space or a task is being terminated.	IEAVRTI1	IEAQPGTM
2 For address space termination, TQE Purge sets bits in the ASCB to cancel time limit checking. Then, it finds the real or wait type TQEs belonging to the address space, dequeues them and frees the storage.	IEAVRTI0	IEAQTD00
For task termination, TQE Purge cancels time limit checking only if the TCB is a job step TCB or higher. Then, TQE Purge checks for TQEs for the TCB, dequeues them, and frees the space.	IEAVRTI0	IEAQTD00
3 TQE Purge purges all timer SRBs for the address space or task, whether the SRB is embedded in the TQE or is built separately in the SQA.	IEAVRTI1 IEAVEPD0	IEAVRSPG IEAVEPD0
4 TQE Purge returns control to Address Space Purge Processing or Task Purge Processing.		

Diagram 18-7. Timer Second Level Interrupt Handler Routine (IEAVRTI0) (Part 1 of 2)



1 Determine type of interruption.

2 For a synchronization check, indicate it in the TPC and disable further synchronization checks.

3 For a CPU timer interruption, process the task TQE and schedule an SRB for exiting.

4 For a clock comparator interruption, process the top TQE on the real time queue.

5 Return.

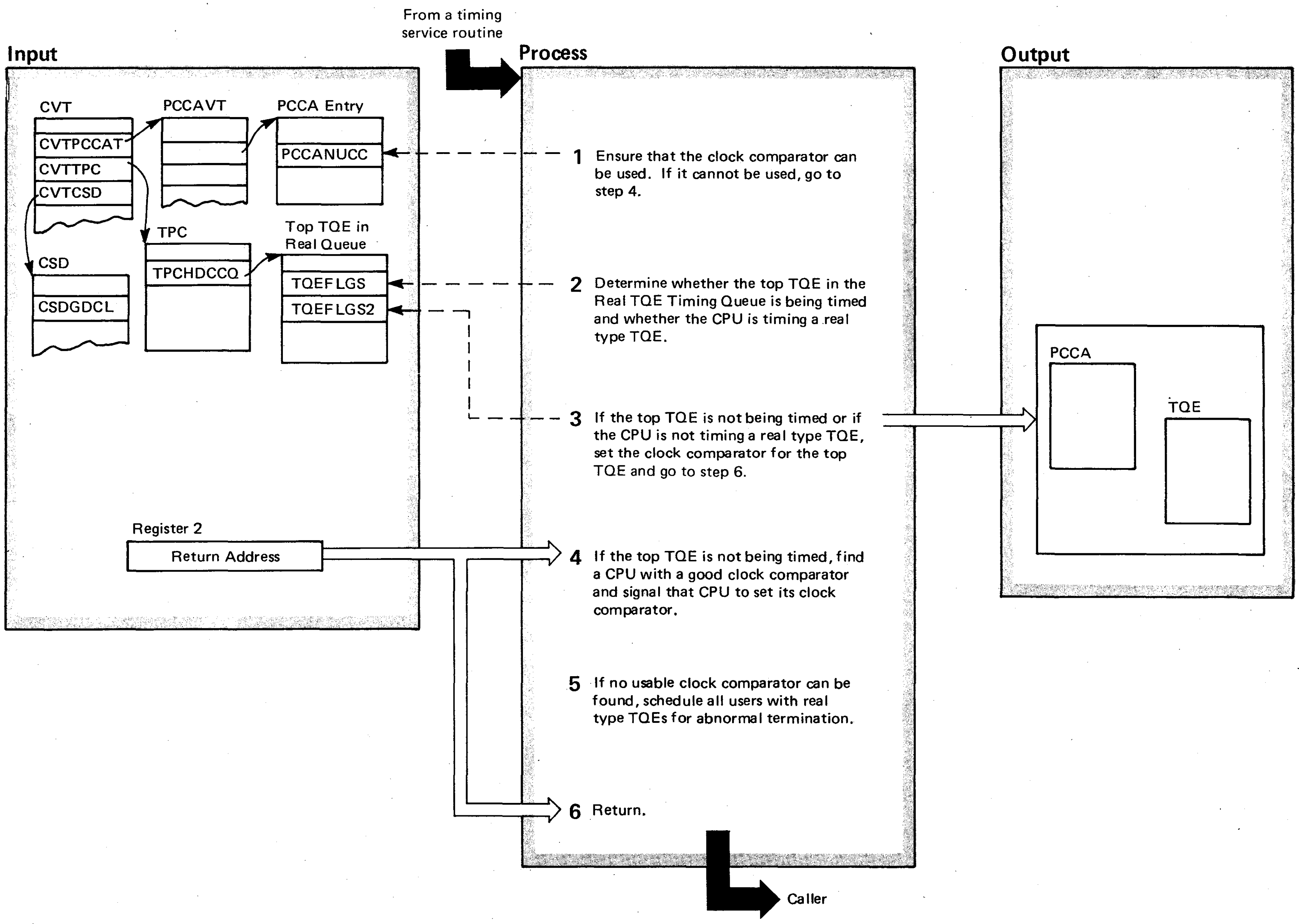
Diagram 18-7. Timer Second Level Interrupt Handler Routine (IEAVRTI0) (Part 2 of 2)

Extended Description	Module	Label
The Timer Second Level Interrupt Handler (TSLIH) routine (IEAVRTI0) processes CPU timer interruptions, clock comparator interruptions, and synchronization check interruptions detected by the external interrupt handler.		
1 TSLIH checks the interruption code in location 135.	IEAVRTI0	IEA0TI00

Code	**Interrupt Type**
X'03'	Synchronization Check
X'04'	Clock Comparator
X'05'	CPU Timer

2 TSLIH indicates the interruption in the TPC (Timer Work Area) and disables synchronization checks.		
3 TSLIH resets the CPU timer and processes any task TQEs, by dequeuing the TQE, and building and scheduling an SRB for it.	IEAVRTI0	IEAQTD00 AECTLRTN
4 TSLIH processes the top TQE in the real time queue by either scheduling the SRB routine for a user TQE or by performing special processing for a system TQE.	IEAVRTI0	SETCC PROCTQE
5 TSLIH passes control to the EXIT routine.		

Diagram 18-8. Set Clock Comparator Routine (IEAVRTI0) (Part 1 of 2)



1 Ensure that the clock comparator can be used. If it cannot be used, go to step 4.

2 Determine whether the top TQE in the Real TQE Timing Queue is being timed and whether the CPU is timing a real type TQE.

3 If the top TQE is not being timed or if the CPU is not timing a real type TQE, set the clock comparator for the top TQE and go to step 6.

4 If the top TQE is not being timed, find a CPU with a good clock comparator and signal that CPU to set its clock comparator.

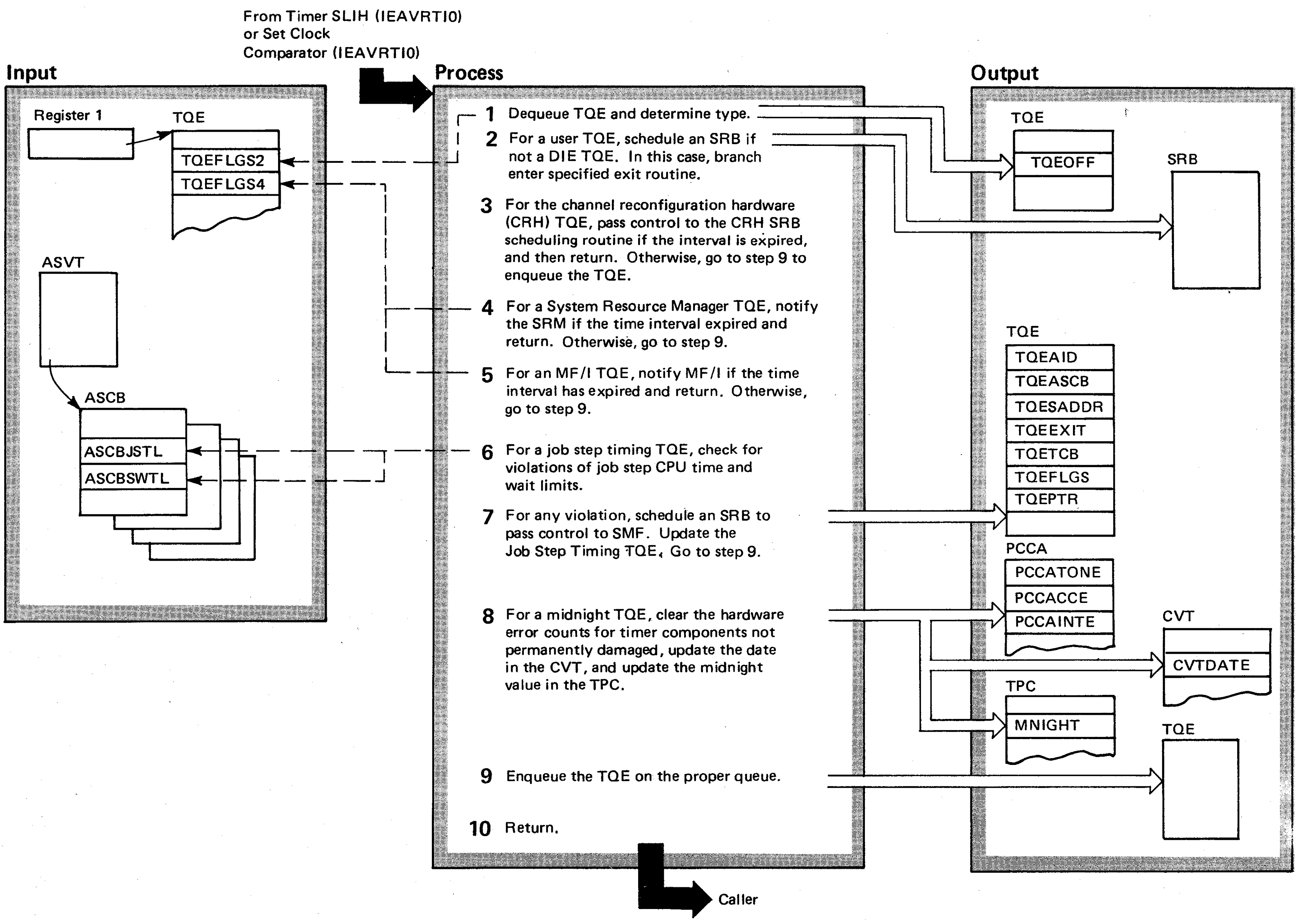
5 If no usable clock comparator can be found, schedule all users with real type TQEs for abnormal termination.

6 Return.

Diagram 18-8. Set Clock Comparator Routine (IEAVRTI0) (Part 2 of 2)

Extended Description	Module	Label
The Set Clock Comparator routine (IEAVRTI0) verifies that the top TQE in the real timer queue is being timed. It also sets the executing CPU's clock comparator, if the CPU is not timing a TQE.		
1 Set Clock Comparator checks the PCCA to determine whether the clock comparator can be used.	IEAVRTI0	IEAVRCKQ
2 Set Clock Comparator determines from the PCCA and the TQE whether the CPU is timing a real TQE and whether the top TQE is being timed.		
3 Set Clock Comparator sets the clock comparator, marks the top TQE as being timed, and indicates, in the PCCA, the TQE being timed. Then it returns to the caller.		
4 Set Clock Comparator searches the PCCA entries for one with a working clock comparator. When it finds one, Set Clock Comparator signals the CPU to set its clock comparator using the Set Clock Comparator routine.	IEAVERP	IEAVERP
5 Set Clock Comparator schedules all users with real type TQEs for abnormal termination.	IEAVRTI0	PROCTQE
6 Set Clock Comparator returns control to the caller.		

Diagram 18-9. TQE Processing Routine (IEAVRTI0) (Part 1 of 2)



1 Dequeue TQE and determine type.

2 For a user TQE, schedule an SRB if not a DIE TQE. In this case, branch enter specified exit routine.

3 For the channel reconfiguration hardware (CRH) TQE, pass control to the CRH SRB scheduling routine if the interval is expired, and then return. Otherwise, go to step 9 to enqueue the TQE.

4 For a System Resource Manager TQE, notify the SRM if the time interval expired and return. Otherwise, go to step 9.

5 For an MF/I TQE, notify MF/I if the time interval has expired and return. Otherwise, go to step 9.

6 For a job step timing TQE, check for violations of job step CPU time and wait limits.

7 For any violation, schedule an SRB to pass control to SMF. Update the Job Step Timing TQE. Go to step 9.

8 For a midnight TQE, clear the hardware error counts for timer components not permanently damaged, update the date in the CVT, and update the midnight value in the TPC.

9 Enqueue the TQE on the proper queue.

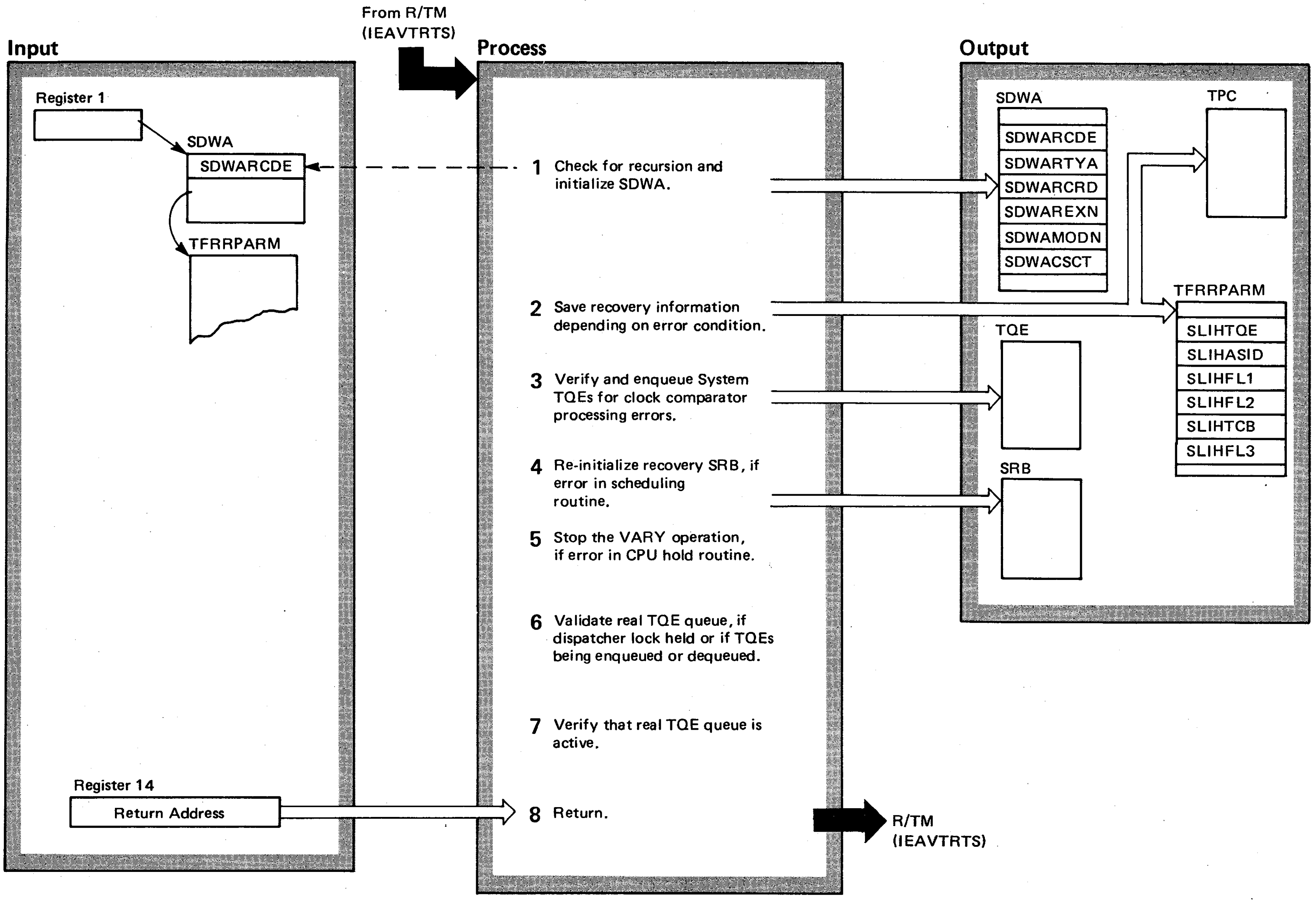
10 Return.

Diagram 18-9. TQE Processing Routine (IEAVRTI0) (Part 2 of 2)

Extended Description	Module	Label
The Process TQE routine dequeues the top TQE on the real timer queue and either does special TQE processing for a system TQE or it schedules an SRB for a user TQE.		
1 PROCTQE dequeues the TQE from the real timer queue and determines from the TQEUSER bit whether the TQE is a user TQE or a system TQE.	IEAVRTI0 IEAVRTI0	PROCTQE IEAQTD00
2 If the user TQE is not a DIE TQE, PROCTQE builds and schedules an SRB (corresponding to the user TQE) into the address space requesting the interval. If the TQE specifies a user exit, an IRB is built by the SRB routine and is scheduled by the Stage II Exit Effector. The user exit will execute under this IRB. If the TQE specifies a wait type request, the SRB routine posts the user. If the user TQE is a DIE TQE, the user's exit (DIE) routine is branch entered directly from PROCTQE.	IEAVRTI0	AECTLRTN
3 If the interval is complete, PROCTQE gets the address of IECVCRHS and branches to it (using BALR).		
4 If the time interval is complete, PROCTQE notifies the System Resources Manager with a SYSEVENT macro instruction that the interval is complete.		
5 PROCTQE notifies MF/I, using the MFROUTER macro instruction, when the interval being timed by the MF/I TQE has completed.		

Extended Description	Module	Label
6 PROCTQE checks all address spaces for violations of a job step's time limit or a waiting job step's wait time limit by checking the ASCBJSTL and ASCBSWTL fields.		
7 For any violations, PROCTQE passes control to SMF via an SRB to check for a possible time interval extension. To do this, PROCTQE builds a partial TQE and then schedules an SRB to pass control to SMF. This SRB will then use the Stage II Exit Effector to schedule the IRB for the initiator's TCB under which IEATLEXT will execute. PROCTQE then updates the Job Step Timing TQE.	IEATLEXT IEAVRTI0	IEATLEXT AECTLRTN
8 If the time interval is complete, PROCTQE clears hardware error counts for the usable timer components in the PCCA. PROCTQE also updates the date in the CVT and adds 24 hours to the TQEVAL field in the midnight TQE.		
9 PROCTQE enqueues the TQE on the real timer queue.	IEAVRTI0	IEAQTE00
10 PROCTQE returns control to the calling routine.		

Diagram 18-10. Timer Functional Recovery Routine (IEAVRTI1) (Part 1 of 2)



1 Check for recursion and initialize SDWA.

2 Save recovery information depending on error condition.

3 Verify and enqueue System TQEs for clock comparator processing errors.

4 Re-initialize recovery SRB, if error in scheduling routine.

5 Stop the VARY operation, if error in CPU hold routine.

6 Validate real TQE queue, if dispatcher lock held or if TQEs being enqueued or dequeued.

7 Verify that real TQE queue is active.

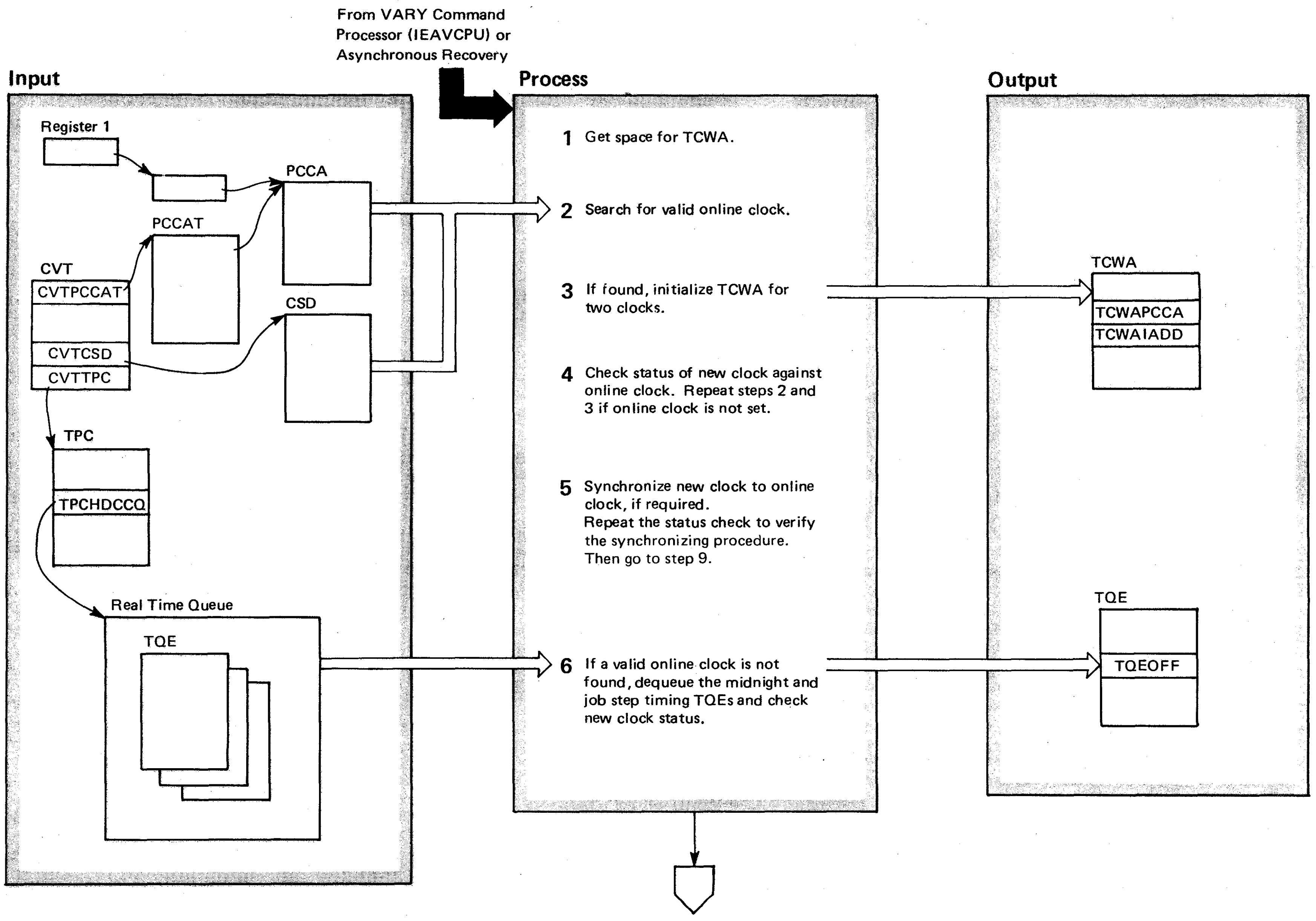
8 Return.

Diagram 18-10. Timer Functional Recovery Routine (IEAVRTI1) (Part 2 of 2)

Extended Description	Module	Label
The Timer Functional Recovery routine (IEAVRTI1) processes program errors in timing components. It records error information and checks the validity of system timer queues.		
1 The Timer Functional Recovery Routine (FRR) checks for possible recursion or re-entry to the recovery routine. If recursion has occurred, the Timer FRR immediately returns to R/TM. Then the Timer FRR initializes identifier fields in the SDWA.	IEAVRTI1	IEAVRFRR
2 The Timer FRR records error information in the TFRR Parameter Area and in the TPC for each type of error for later recording in the SDWA.		
3 If the error is in a clock comparator interruption, the Timer FRR updates the system TQEs and re-enqueues them.	IEAVRTI1 IEAVRTI0	IEAVRTVR IEAQTE00

Extended Description	Module	Label
4 If the error is in the asynchronous recovery scheduling routine, the Timer FRR marks the recovery SRB not-in-use.		
5 If the error occurs in the CPU hold routine, the Timer FRR determines whether to cancel the VARY process.		
6 After the TFRR Parameter List has been recorded in the SDWA, the Timer FRR determines whether the error occurred when a dispatcher lock was held, or when a TQE enqueue or TQE dequeue was in process. If one of the conditions is true, it verifies the real TQE queue. If the TPC is found to be invalid, the Timer FRR indicates the invalid TPC and returns control to R/TM.	IEAVRTI1 IEAVRTI1	IEAVEADV IEAVEQV0
7 The Timer FRR verifies that the real TQE queue is active.	IEAVRTI0	IEAVRQCK
8 The Timer FRR returns to R/TM through register 14.		

Diagram 18-11. Set Specific Clock (SSC) Routine (IEAVRTOD) (Part 1 of 4)



1 Get space for TCWA.

2 Search for valid online clock.

3 If found, initialize TCWA for two clocks.

4 Check status of new clock against online clock. Repeat steps 2 and 3 if online clock is not set.

5 Synchronize new clock to online clock, if required. Repeat the status check to verify the synchronizing procedure. Then go to step 9.

6 If a valid online clock is not found, dequeue the midnight and job step timing TQEs and check new clock status.

Diagram 18-11. Set Specific Clock (SSC) Routine (IEAVRTOD) (Part 2 of 4)

Extended Description	Module	Label
The Set Specific Clock (SSC) routine (IEAVRTOD) processes requests to set a particular TOD clock. It attempts to find a valid clock, synchronizes the new clock to the valid clock, if found, and tests the validity of the synchronization. It is called from the VARY command processor (IEEVCPU) and from the clock asynchronous recovery routine that is located in module IEAVRTOD (entry point IEAVRCLA).		
1 SSC obtains storage for the TCWA (TOD Clock Work Area) from subpool 245. If the GETMAIN is unsuccessful, SSC saves the return code in register 15 and indicates the VARY operation should be halted, or that recovery was unsuccessful, by returning a non-zero return code to the caller.	IEAVRTOD	IEAVRSSC
2 Using the bit mask in the CSD indicating the active processing units in the system, SSC finds an online CPU. Then it tests the status byte in the PCCA (Physical Configuration Communication Area) for a valid TOD clock.		
3 SSC initializes the PCCA address and the CPU address in the TOD Clock Work area entries: one for the CPU being varied online or being recovered, and one for the CPU with the valid TOD clock.		
4 IEAVRSSC tests the new clock against the valid clock for synchronization. If the valid clock is not set (return code 4 from TOD Clock Status Test), SSC continues searching the PCCA entries for another valid clock.	IEAVRTOD	IEAVRTST
5 If the return code from TOD Clock Status Test is 8, 12, or 16, SSC tries to synchronize the new clock to the valid clock.	IEAVRTOD	IEAVRSYN
Then it tests for success of the synchronization.	IEAVRTOD	IEAVRTST
6 If a valid TOD clock is not found, SSC dequeues the midnight and job step timing TQEs and disallows external SET commands.	IEAVRTIO	IEAQTD00
Then it initializes a single TCWA entry and checks the status of the new clock.	IEAVRTOD	IEAVRTST

Diagram 18-11. Set Specific Clock (SSC) Routine (IEAVRTOD) (Part 3 of 4)

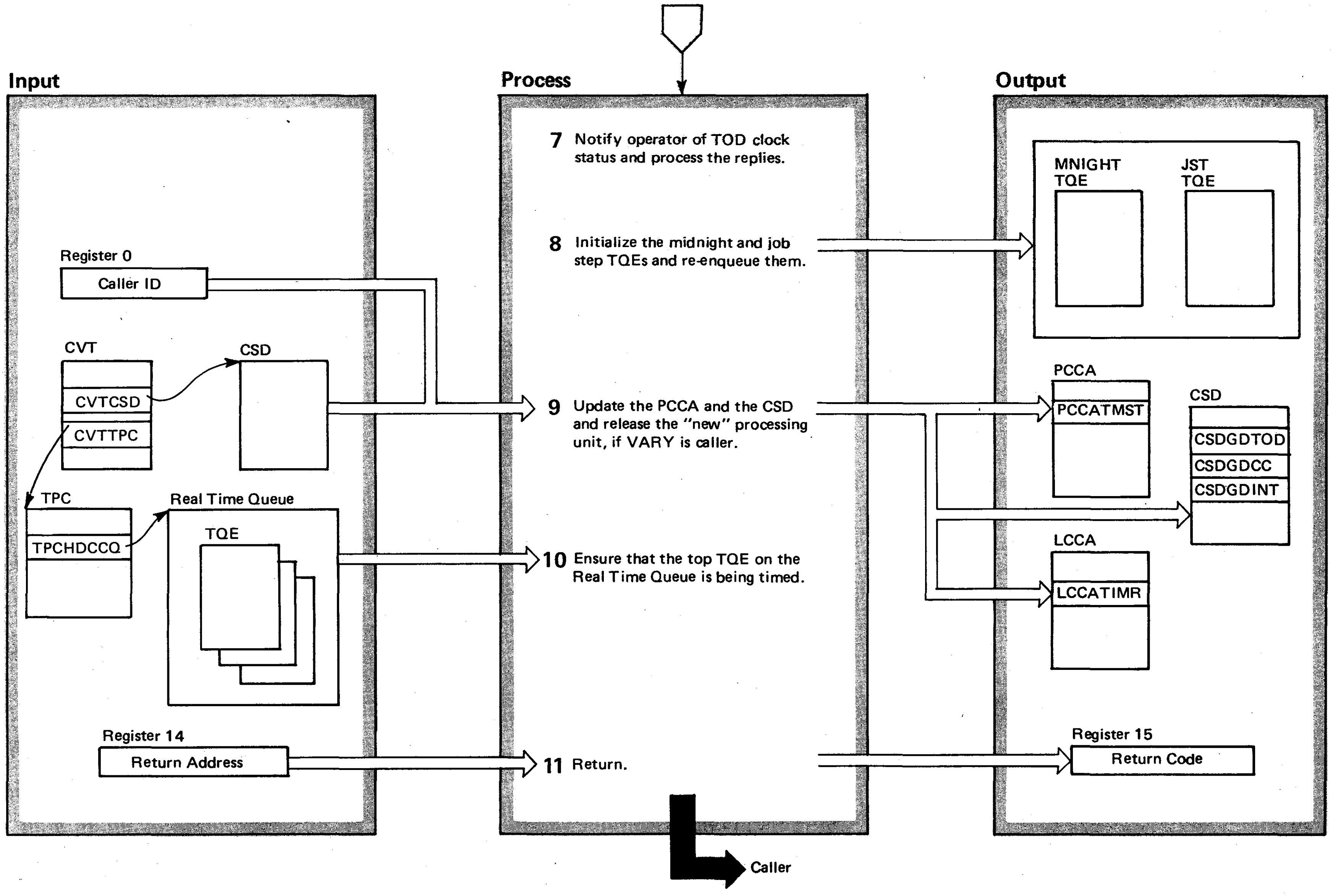
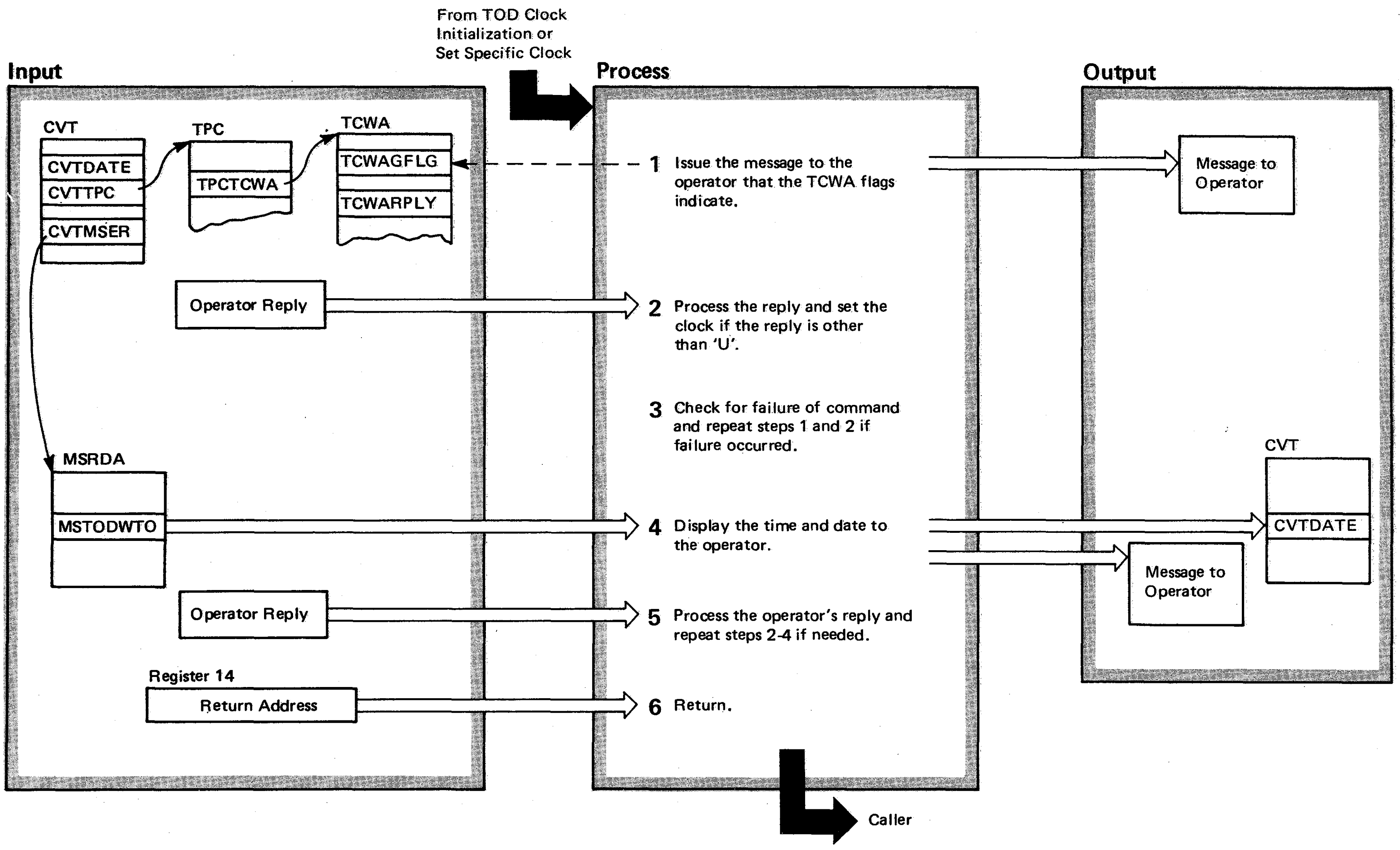

Input
Process
Output
Register 0
Caller ID
CVT
CVTCSD
CVTTPC
CSD
TPC
TPCHDCCQ
Real Time Queue
TQE
Register 14
Return Address
7 Notify operator of TOD clock status and process the replies.
8 Initialize the midnight and job step TQEs and re-enqueue them.
9 Update the PCCA and the CSD and release the "new" processing unit, if VARY is caller.
10 Ensure that the top TQE on the Real Time Queue is being timed.
11 Return.
Caller
MNIGHT TQE
JST TQE
PCCA
PCCATMST
CSD
CSDGDTOD
CSDGDCC
CSDGDINT
LCCA
LCCATIMR
Register 15
Return Code

Diagram 18-11. Set Specific Clock (SSC) Routine (IEAVRTOD) (Part 4 of 4)

Extended Description	Module	Label
7 If the return code from TOD Clock Status Test is 0, SSC issues message IEA888A; if the return code is non-zero, SSC issues message IEA886A. Then it tests the clock to be sure it is set.	IEAVRTOD IEAVRTOD	IEAVRCOM IEAVRTST
8 If the new clock is set, SSC re-enqueues the midnight and job step timing TQEs and notifies the System Resources Manager that there is now a valid TOD clock and clock comparator in the system.	IEAVRTOD	TQEINIT
9 If the caller is VARY, SSC initializes the timer status bytes in the PCCA, updates the count of usable TOD clocks, clock comparators, and CPU timers in the CSD, and releases the new CPU from its holding state.		
10 SSC ensures that the top TQE on the real TQE queue is being timed.	IEAVRTI0	IEAVRQCK
11 SSC allows external SET commands, frees the TCWA space, and returns to the caller.		
Error Recovery When an SSC routine error is detected and passed by R/TM, the ESTAE routine sets up information so that R/TM will cause SSC to be re-entered at a point at which resources can be cleaned up and a return can be made to the caller.	IEAVRTOD	SSCESTAE

Diagram 18-12. TOD Clock Operator Communication Routine (IEAVRTOD) (Part 1 of 2)



1 Issue the message to the operator that the TCWA flags indicate.

2 Process the reply and set the clock if the reply is other than 'U'.

3 Check for failure of command and repeat steps 1 and 2 if failure occurred.

4 Display the time and date to the operator.

5 Process the operator's reply and repeat steps 2-4 if needed.

6 Return.

Diagram 18-12. TOD Clock Operator Communication Routine (IEAVRTOD) (Part 2 of 2)

Extended Description	Module	Label
The TOD Clock Operator Communication routine (IEAVRTOD) handles operator communication for TOD clock status. The routine issues messages and processes the operator's replies, issuing SET commands to update the TOD clock and local time and date. It is called from either TOD Clock Initialization located in IEAVRTOD (entry point IEAVRINT) or Set Specific Clock located in IEAVRTOD (entry point IEAVRSSC).		
1 Operator Communication issues the message that the flags in the TCWA indicate. The flag settings depend on return codes from the Clock Status Test routine.	IEAVRTOD	IEAVRCOM

Return Code	Message
0	IEA888A
8 or 12 (1 clock set)	IEA888A
4 or 16	IEA886A
8 or 12 (>1 clock set)	IEA887A

If message IEA888A is to be issued, go to step 4.

2 Operator Communication processes the reply according to the message sent:

Message	Valid Replies
IEA888A	CLOCK=nn,DATE=nn [,GMT] [,IPS=]
	CLOCK=nn [,GMT] [,IPS=]
	DATE=nn [,GMT] [,IPS=]
	IPS=
IEA886A	CLOCK=nn, DATE=nn [,GMT] [,IPS=]
	DATE=nn [,GMT] [,IPS=]
IEA887A	ID=nn [,IPS=]
	CLOCK=nn, DATE=nn [,GMT] [,IPS=]
	DATE=nn [,GMT] [,IPS=]

If a syntax error is found, the message is repeated until a correct reply is made. When the operator enters a reply, Operator Communication performs the indicated function as requested by issuing an internal SET command.

Extended Description	Module	Label
3 If the SET command fails, Operator Communication issues the message and processes the new reply.		
4 Operator Communication obtains a current TOD clock value, calculates the local time, and initializes the date field in the CVT. It also calculates the GMT time and date and, if prompting is to be done (indicated by bit MSTODWTO), displays both the local and Greenwich Mean time values in message IEA888A.	IEAVRTOD IEAVRTOD	DELTA3 CUTODATE
5 If the operator accepts the values or if no prompting is allowed, Operator Communication returns. If the operator enters new values, Operator Communication sets the clock or local time and date values accordingly with the SET command, reissues message IEA888A, and processes the reply until the values are accepted.		
6 Operator Communication returns to the caller.		

Diagram 18-13. TOD Clock Synchronization Routine (IEAVRTOD) (Part 1 of 2)

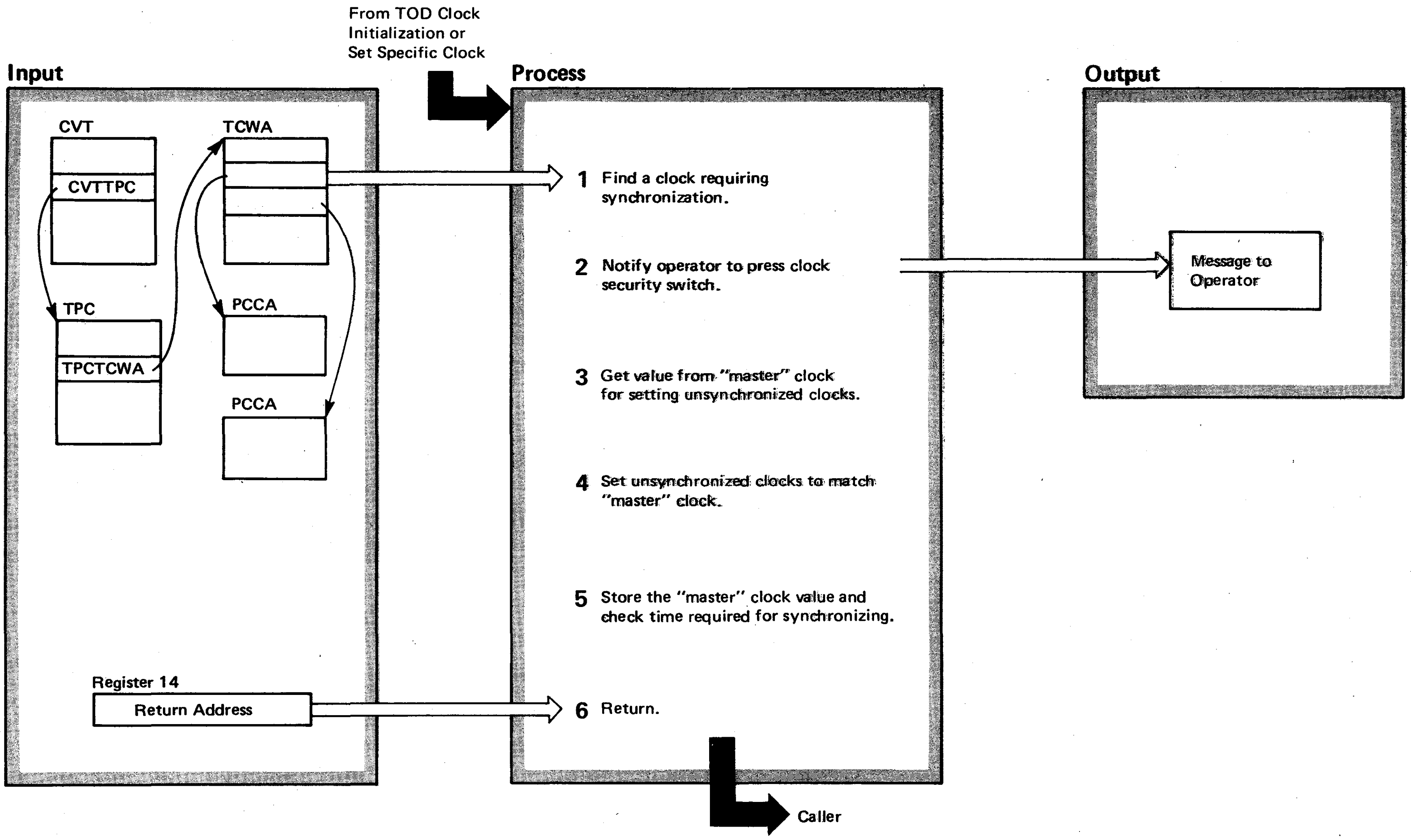

From TOD Clock Initialization or Set Specific Clock
Input
CVT
CVTTPC
TPC
TPCTCWA
TCWA
PCCA
PCCA
Register 14
Return Address
Process
1 Find a clock requiring synchronization.
2 Notify operator to press clock security switch.
3 Get value from "master" clock for setting unsynchronized clocks.
4 Set unsynchronized clocks to match "master" clock.
5 Store the "master" clock value and check time required for synchronizing.
6 Return.
Caller
Output
Message to Operator

Diagram 18-13. TOD Clock Synchronization Routine (IEAVRTOD) (Part 2 of 2)

Extended Description	Module	Label
The TOD Clock Synchronization routine (IEAVRTOD) processes requests to synchronize TOD clocks. After getting a master clock value, it sets all clocks that need synchronization. It is called from either TOD Clock Initialization located in IEAVRTOD (entry point IEAVRINT) or Set Specific Clock located in IEAVRTOD (entry point IEAVRSSC).		
1 Synchronization searches the PCCA (Physical Configuration Communication Area) via the TCWA entries to find a TOD clock requiring synchronization.	IEAVRTOD	IEAVRSYN
2 Synchronization issues message IEA889A asking the operator to press the TOD clock security switch and waits for him to acknowledge receipt of the message. If the switch is not pressed within 30 seconds after acknowledgement, Synchronization repeats the message.		
3 Synchronization calculates the master TOD clock value and initializes a TCWA field with the value.	IEAVRTOD	DELTA4
4 Synchronization checks each PCCA, via its TCWA entry, for a flag indicating that the clock needs synchronizing. If it finds one, Synchronization tries to set the clock with the master clock value. Then it repeats step 4.		
5 When all clocks have been set, Synchronization stores the master clock and verifies that all clocks were set within 2^{20} microseconds.	IEAVRTOD	DELTA3
6 If, during the synchronizing process, the operator releases the TOD clock security switch, or if the operation cannot be completed within 2^{20} microseconds, Synchronization repeats the procedure from step 2. If the operation has been completed successfully, Synchronization returns to the caller.		

Diagram 18-14. TOD Clock Status Test Routine (IEAVRTOD) (Part 1 of 2)

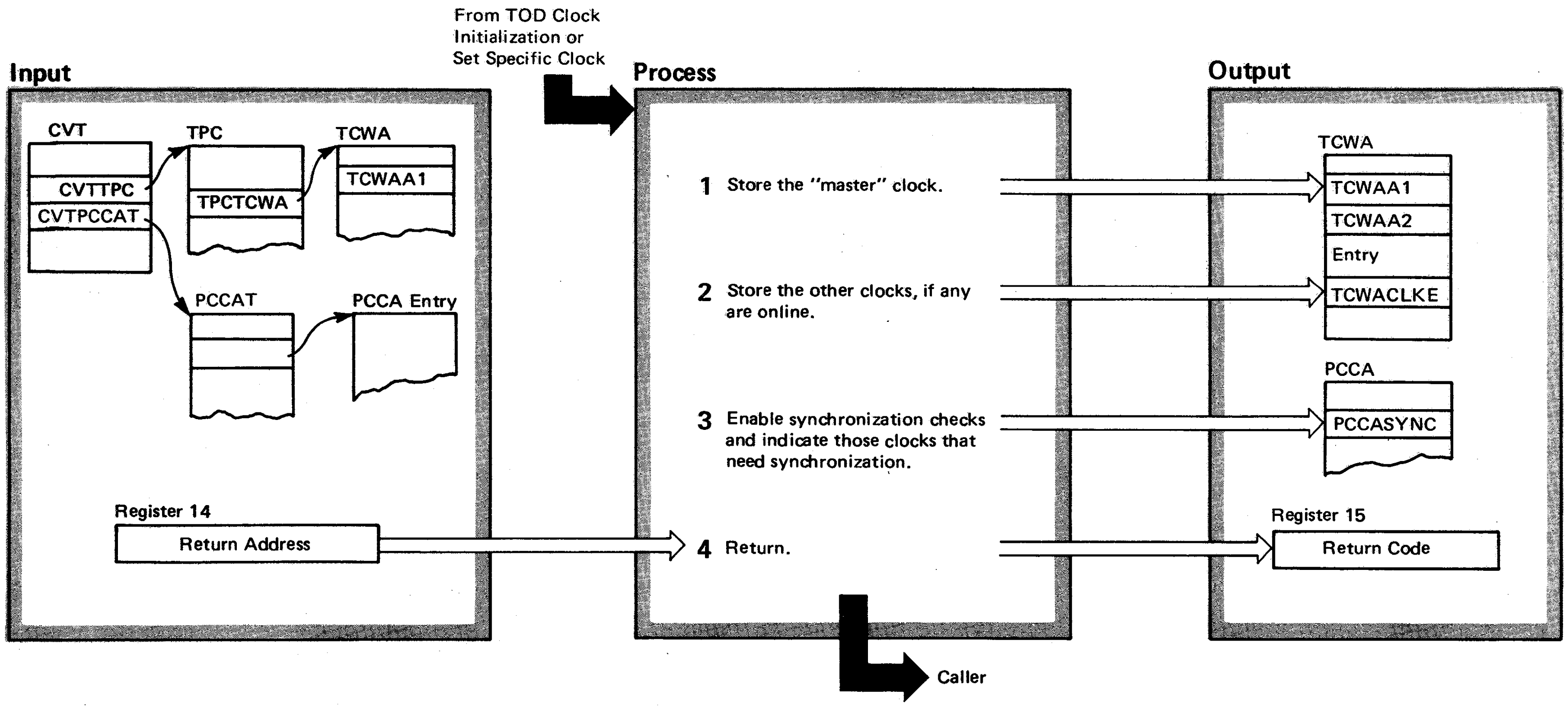
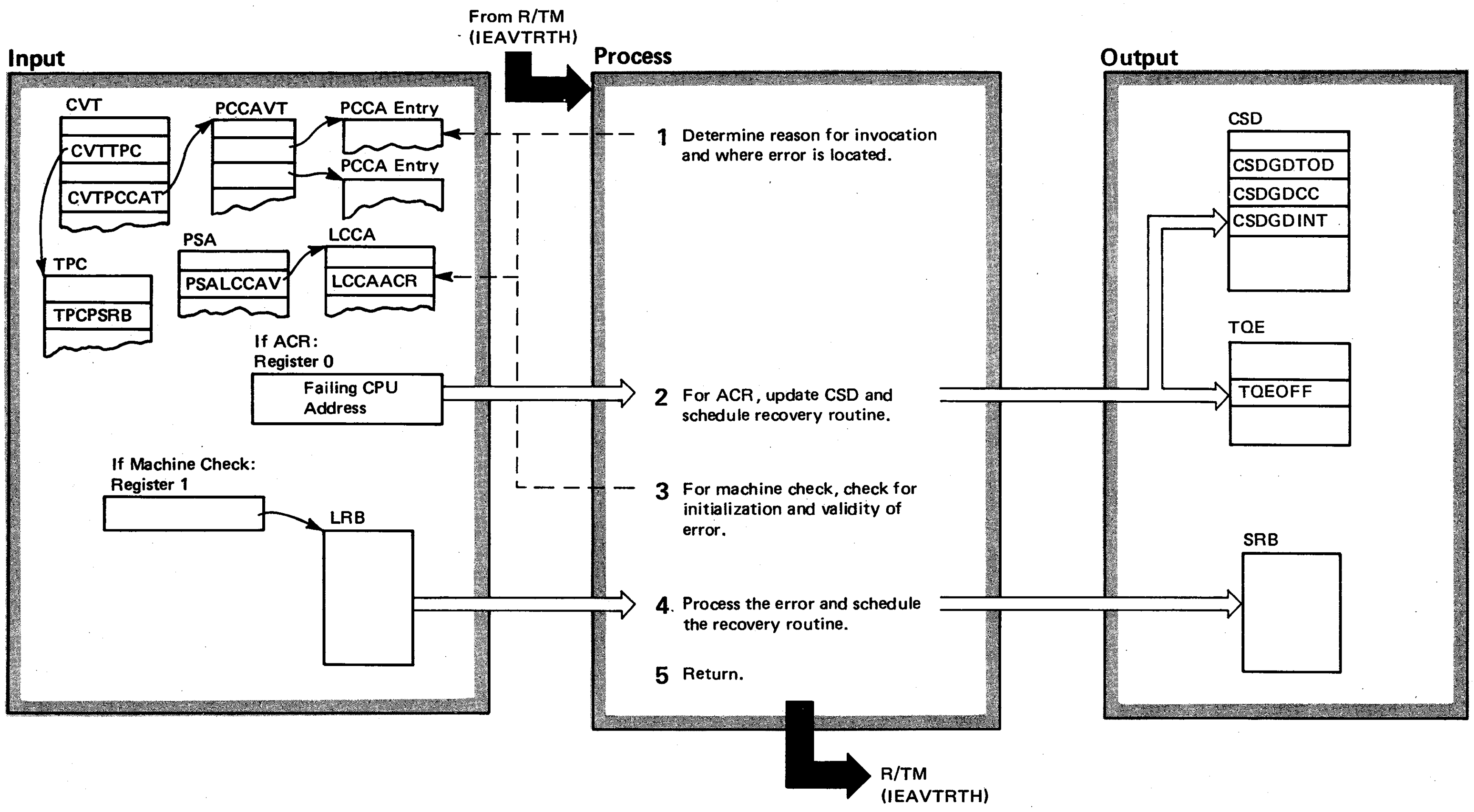

From TOD Clock Initialization or Set Specific Clock
Input
CVT
CVTTPC
CVTPCCAT
TPC
TPCTCWA
TCWA
TCWAA1
PCCAT
PCCA Entry
Register 14
Return Address
Process
1 Store the "master" clock.
2 Store the other clocks, if any are online.
3 Enable synchronization checks and indicate those clocks that need synchronization.
4 Return.
Caller
Output
TCWA
TCWAA1
TCWAA2
Entry
TCWACLKE
PCCA
PCCASYNC
Register 15
Return Code

Diagram 18-14. TOD Clock Status Test Routine (IEAVRTOD) (Part 2 of 2)

Extended Description	Module	Label
The TOD Clock Status Test routine (IEAVRTOD) processes requests for the status of the TOD clocks in a system. The routine determines which clocks are set and which clocks needs synchronization. It then indicates these conditions through a return code in register 15. It is called from either TOD Clock Initialization located in IEAVRTOD (entry point IEAVRINT) or Set Specific Clock located in IEAVRTOD (entry point IEAVRSSC).		
1 Status Test stores the first clock listed in the TCWA.	IEAVRTOD	IEAVRTST
If the clock is not set, Status Test puts a return code of 4 in register 15 to indicate that fact.	IEAVRTOD	DELTA1
2 If other clocks exist, Status Test stores them in the system. It then verifies that the operation of steps 1 and 2 took less than 2^{20} microseconds.	IEAVRTOD	DELTA1
3 Status Test allows synchronization checks through the interruption handler. If such an interruption occurs, a bit in the TPC is set by the timer SLIH, and Status Test sets the return code to 8 and marks the PCCA entries out-of-synchronization. If no check occurs, Status Test disables synchronization checks and then tests for high-order synchronization, with a return code of 12 indicating an out-of-synchronization condition.		
4 If all clocks are set and synchronized, Status Test sets a return code of 0 and returns to the caller. If no status can be determined after 5 tries (steps 1 and 2 cannot be performed within 2^{20} microseconds), Status Test sets a return code of 16 and returns.		

Diagram 18-15. Synchronous Timer Recovery Routine (IEAVRTI1) (Part 1 of 2)



1 Determine reason for invocation and where error is located.

2 For ACR, update CSD and schedule recovery routine.

3 For machine check, check for initialization and validity of error.

4 Process the error and schedule the recovery routine.

5 Return.

Diagram 18-15. Synchronous Timer Recovery Routine (IEAVRTI1) (Part 2 of 2)

Extended Description	Module	Label
The Synchronous Timer Recovery routine (IEAVRTI1) processes timer hardware errors, either machine checks or alternate CPU retry (ACR) checks. The routine checks each timer component for permanent damage and indicates its condition.		
1 By checking a bit in the LCCA, Synchronous Recovery determines whether an ACR or a machine check caused it to be called.	IEAVRTI1	IEAVRCLS
2 For alternate CPU recovery (ACR), Synchronous Recovery checks the PCCA for indications of permanent damage to the timer components, updates the count of the component in error in the CSD, and marks any TQEs as no longer being timed. Then it schedules the asynchronous recovery routine.	IEAVRTI1	IEAVRCLX
3 For the machine checks, IEAVRCLS verifies that the timing components have been initialized and that the machine check is valid. If either condition is false, Synchronous Recovery exits.		
4 Synchronous Recovery checks each component for errors and determines whether that component is permanently damaged. If a clock comparator or CPU timer is permanently damaged, Synchronous Recovery indicates that message IEA898I should be issued. If either of the components is not permanently damaged, Synchronous Recovery schedules the asynchronous recovery routine for further recovery.	IEAVRTI1	IEAVRCLX
5 When it has completed its processing, Synchronous Recovery returns control to R/TM (IEAVTRTH).		

Diagram 18-16. Asynchronous Timer Recovery Routine (IEAVRTOD) (Part 1 of 2)

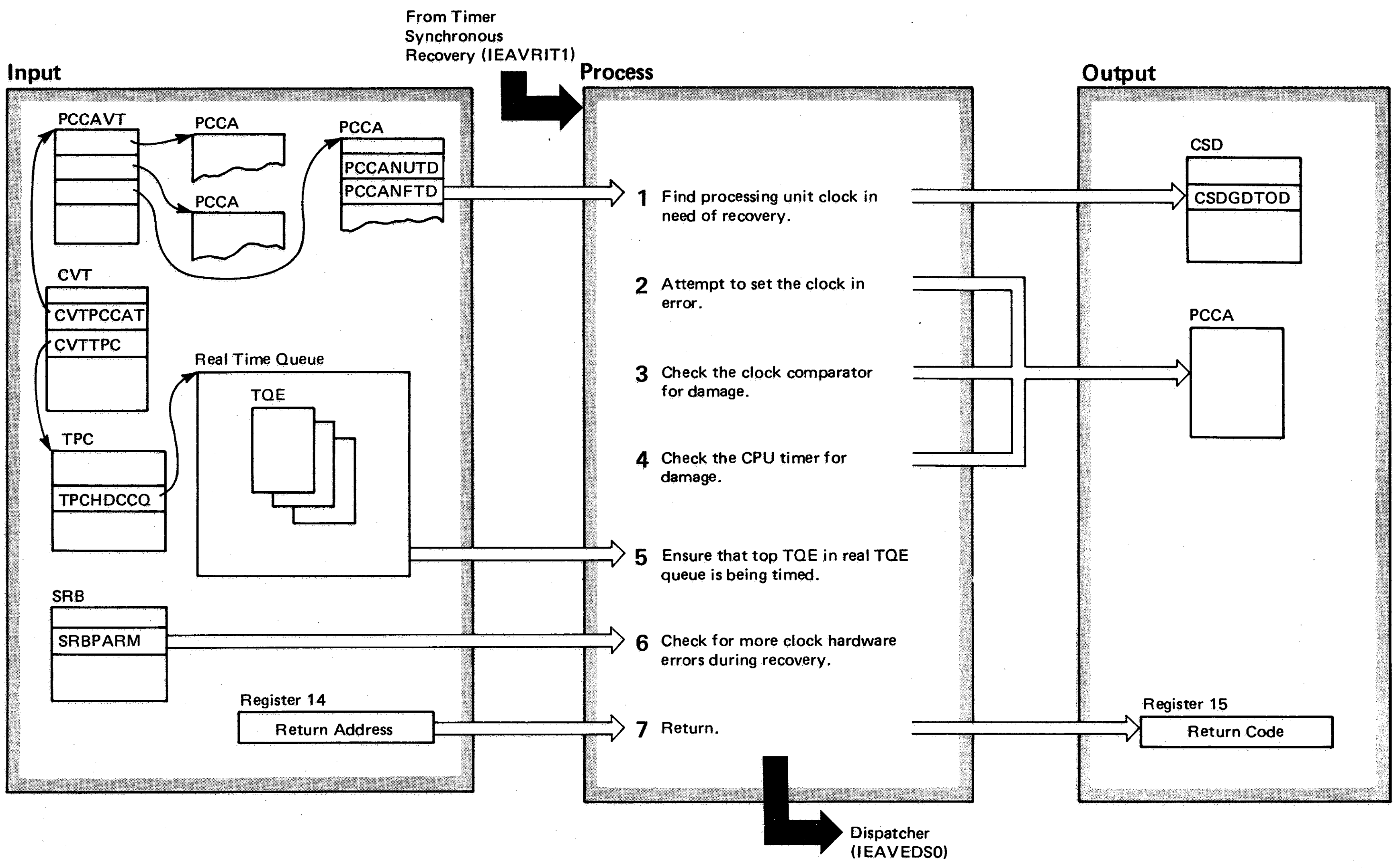

From Timer Synchronous Recovery (IEAVRIT1)
Input
Process
Output
PCCAVT
PCCA
PCCA
PCCA
PCCANUTD
PCCANFTD
CVT
CVTPCCAT
CVTTPC
TPC
TPCHDCCQ
Real Time Queue
TQE
SRB
SRBPARM
Register 14
Return Address
1 Find processing unit clock in need of recovery.
2 Attempt to set the clock in error.
3 Check the clock comparator for damage.
4 Check the CPU timer for damage.
5 Ensure that top TQE in real TQE queue is being timed.
6 Check for more clock hardware errors during recovery.
7 Return.
CSD
CSDGDTOD
PCCA
Register 15
Return Code
Dispatcher (IEAVEDS0)

Diagram 18-16. Asynchronous Timer Recovery Routine (IEAVRTOD) (Part 2 of 2)

Extended Description	Module	Label
The Asynchronous Timer Recovery routine (IEAVRTOD) attempts recovery of TOD clock hardware errors processed by the Synchronous Recovery routine. It also issues messages indicating permanent damage to any timer components, if necessary.		
1 Asynchronous Recovery, after fixing the page containing itself, scans the timer status bytes in the PCCA entries for error indications.	IEAVRTOD	IEAVRCLA
2 When it finds an error indication, Asynchronous Recovery checks for a TOD clock error and, if it finds one, tries to reset or resynchronize the clock. If it cannot be fixed, Asynchronous Recovery marks the TOD clock and, if necessary, the clock comparator, permanently damaged, and issues message IEA898I to inform the operator of the TOD clock failure.	IEAVRTOD	IEAVRSSC
3 Asynchronous Recovery checks the clock comparator status. If it needs recovery, Asynchronous Recovery checks to see if Synchronous Recovery tried to recover and failed. If so, Asynchronous Recovery marks the clock comparator permanently damaged and issues message IEA898I to the operator to inform him of the clock comparator failure.		
4 Asynchronous Recovery checks the CPU timer status. If it needs recovery, Asynchronous Recovery tests to see if Synchronous Recovery tried to recover it and failed. If so, Asynchronous Recovery marks the CPU timer permanently damaged and issues message IEA898I to the operator to inform him of the CPU timer failure.		
5 Asynchronous Recovery ensures that the real TQE queue is active.	IEAVRTIO	IEAVRQCK
6 Asynchronous Recovery tests whether another error, that requires recovery, occurred during the recovery process. If so, Asynchronous Recovery is re-entered at the PCCA scanning part of step 1.		
7 Asynchronous Recovery frees the page containing itself and returns to the dispatcher.		
Error Processing Errors in Asynchronous Recovery code are processed on entry from R/TM. The SDWA is initialized or, if no SDWA is supplied, a return code of 4, indicating retry, is placed in register 15. Then control is returned to R/TM, which causes Asynchronous Recovery to be re-entered at a point where resources can be cleaned up and a return made to the dispatcher.	IEAVRTOD	CLAESTAE

Supervisor Control performs control routing services. These include:
- The service manager which schedules requests.
- The dispatcher which dispatches work.
- The various interruption handlers which route control to appropriate routines for given interruptions.
- Interprocessor communications (IPC).
- The exit effectors which provide a mechanism for scheduling asynchronous exits.
- The lock manager which serializes system resources.
- A validity checking routine which validates a storage location's protect key.
- Supervisor control recovery routines which provide functional recovery for supervisor control.

Service management is a set of functions which allows system components to execute their own routines enabled, in parallel in an MP (multiprocessor) environment, in a mode independent of the normal task structure. This consists of the following services:
- SCHEDULE. The SCHEDULE macro instruction allows a requester to schedule a system service.
- PURGEDQ. The PURGEDQ macro instruction allows a requester to cancel a previously-scheduled service request. In effect, PURGEDQ is the inverse of SCHEDULE.

Dispatching work consists of the following services:
- Dispatcher. The dispatcher chooses the units of work to be executed. The dispatcher may receive control after a task is interrupted or enters a wait state, after a service request completes or is suspended, or from other system routines that want the highest priority work dispatched.
- Memory switch. Memory switch chooses the highest priority address space (memory) that can process ready work.

Handling interruptions consists of the following services:
- SVC IH (interruption handler). The SVC IH routes control to the appropriate SVC routine after a requester issues an SVC (supervisor call) instruction. The SVC IH receives control from the SVC new PSW (program status word). The extended SVC router, a logical extension of the SVC IH, routes control to extended SVC routines.
- I/O IH. The I/O IH routes control to the I/O supervisor after the hardware receives an I/O interruption. The I/O IH receives control from the I/O new PSW.
- External FLIH (first level interruption handler). The external FLIH routes control to the appropriate SLIH (second level interruption handler) after an external interruption occurs. The external FLIH receives control from the external new PSW.
- Restart IH. The restart IH routes control to either DSS (dynamic system support) or R/TM (recovery/termination-management) after the operator strikes the restart button on the console, or after a system program issues a restart SIGP (signal processor) instruction. The restart IH receives control from the restart new PSW.
- Program IH. The program IH routes control to the appropriate service routine after a program check interruption occurs. The program IH receives control from the program new PSW.

Interprocessor communications consists of the following services:
- Signal Service Routines. The signal service routines — the remote immediate, the remote pendable, and the direct routines — provide the signal sending capability between CPUs in a multiprocessor system.
- Emergency Signal SLIH. The emergency signal SLIH routes control to the appropriate receiving routine after a requester issues a remote immediate signal request. Control comes from the external FLIH.
- External Call SLIH. The external call SLIH routes control to one of six service routines after a requester issues a remote pendable signal request. The external call SLIH receives control from the external FLIH.

The Exit effectors provide a mechanism to schedule supervisor or problem program routines (ETXR, Timer Exit, attention exit) to execute under the normal task structure.

Serializing of system resources is provided by the SETLOCK interface. The SETLOCK service obtains and releases the necessary "locks" to prevent one CPU on a multiprocessor system from using a resource needed by another CPU.

Validity check determines the validity of an address or address range for the requester.

Supervisor control recovery routines receive control from:

- A direct interface with R/TM.
- The normal FRR (functional recovery routine) mechanism.
- The "super FRR" mechanism.

They may verify and correct queues or other system data, abnormally terminate ABEND the current routine, or simply resume system operation through the dispatcher.

Service Manager

In order to facilitate multiprocessing, OS/VS2 uses a new catagory of facilities, called service management, to schedule system services. Service management consists of:

- A new macro instruction, SCHEDULE, which allows new service requests to be entered into the queue of dispatchable work with a minimal amount of overhead.
- A new control block, supplied to SCHEDULE as input and called a Service Request Block (SRB), (44 bytes), represents a service request. The SRB contains information needed to dispatch the routine.
- A second new macro instruction, PURGEDQ, which allows service requests to be terminated.

The dispatcher (described under the topic, "Dispatching Work") utilizes new queues to dispatch SRBs.

Figure 2-33 depicts the basic pointer structure utilized by the service management facilities. This structure incorporates two levels of system priority, global and local. Service requests queued at the global level have a priority above that of *a* address space, regardless of the actual address space in which they will be dispatched. Service requests queued at the local level have a priority equal to that of the address space in which they will be dispatched but higher than that of any task within that address space.

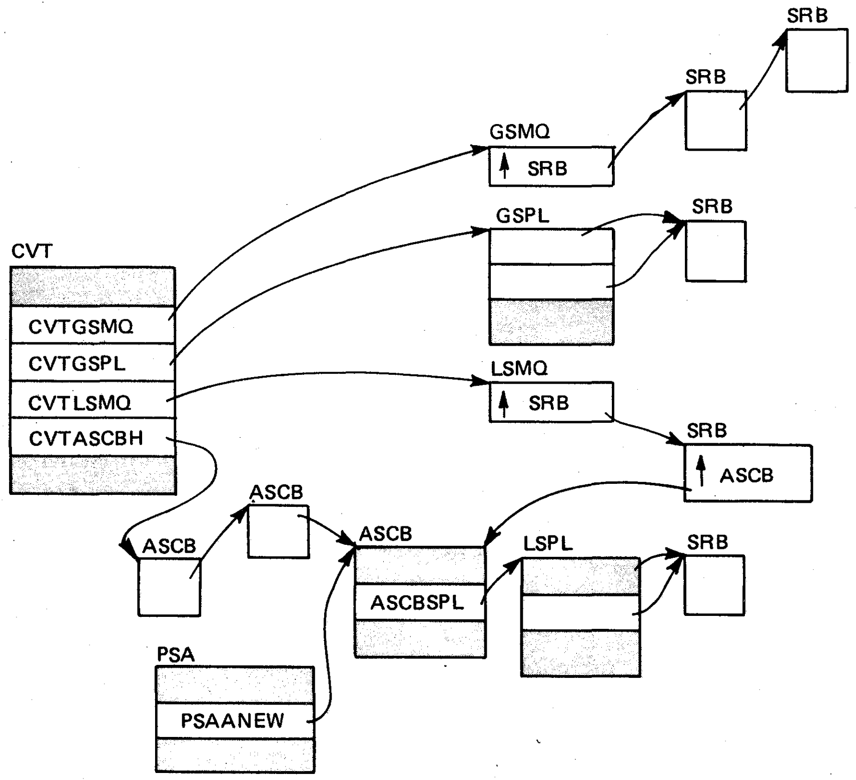


Figure 2-33. SRB Scheduling Pointer Structure

A Service Priority List (SPL) exists at each level (global or local). This lists consists of a static, contiguous list of queue anchors and simply serves to give priorities to the various types of service requests. Each element of the SPL serves as an anchor for a queue of service requests so that the dispatcher can start at the top of an SPL and take any request queued at the first element prior to looking for a request queued at a lower element. Thus, the SPL is effectively a list of priority levels, with a single global SPL for the system and one local SPL per 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 page fault or are suspended off a lock.
- They may free the SRB storage once they get control.
- They may take page faults.
- They may not issue SVCs.
- They may execute in any designated address space, and thus provide the primary vehicle for cross-memory communication.

Dispatching Work

Dispatching consists of routing control to the highest priority ready unit of work. The units of work consist of the following:

- Special exits, which are subroutines branched to by the dispatcher when indicated.
- Service requests, which are represented by SRBs on the dispatcher queues, queued via the SCHEDULE macro.
- Tasks, which are represented by TCBs. There is one TCB queue per address space, residing in the private storage of that address space.

The dispatcher searches for ready work in a specific order:

- Special exits.
- Global priority SRBs.
- The highest priority ready address space and within that address space the local priority SRBs and tasks.
- Wait task.

The process involved in scheduling service requests is as follows:

1. The user must:
   - construct the SRB.
   - schedule it, via the SCHEDULE macro instruction.
2. The SCHEDULE macro instruction, via the CS instruction, places the SRB on one of the two staging queues (the GSMQ or the LSMQ) in LIFO order. The SCHEDULE macro performs the queueing in line. The user will continue to execute until he is interrupted, causing a dispatcher entry.
3. The dispatcher checks for SRBs on the GSMQ, and completes scheduling of global SRBs using the global scheduling routine (IEAVESC1).
4. Global schedule moves the SRBs from the GSMQ to the correct priority level of the GSPL.
5. When all SRBs have been moved, an external call SIGP (issued via RPSGNL) will signal any waiting CPUs.
6. The dispatcher then dispatches SRBs from the global SPL. It dequeues SRBs from the GSPL and passes them to the SRB routine. If an SRB cannot be dispatched for some reason, (for example, the address space is swapped out) it will be rescheduled at local priority.
7. The dispatcher checks for SRBs on the LSMQ and schedules them using the local schedule routine (IEAVESC2).
8. Local schedule moves each SRB to the local SPL specified by the SRBASCB value.
9. The SRM (system resource manager) is notified if the address space is swapped out. The address space will be eventually swapped in.
10. If the address space is still on the dispatching queue, Memory Switch will be invoked to eventually update the PSAANEW indication. When all SRBs have been moved, control returns to the dispatcher.
11. The dispatcher looks for the highest priority ready address space, beginning its search with the value specified by PSAANEW.
12. SRBs will next be dequeued and dispatched from the local SPL of the highest priority ready address space. Then, tasks will be dispatched from that address space.

Handling Interruptions

The interruption handlers route control to the appropriate routines after machine interruptions occur. Any interruption causes CPU 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 IH 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 IH 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 CPU timer expiration, clock comparator interruption, or clock synchronization failure); hitting interrupt key (when the operator presses interrupt key on the console); external calls (when remote pendable signal routine signals another CPU); emergency signals (when machine check handler or remote immediate signal routine signals another CPU); or malfunction alerts (caused by machine failure of another CPU). The external IH determines the cause of the interruption and branches to the external service routine.
- Restart interruptions, which occur when the operator strikes the restart button on the system operator's console, or when a system program issues a SIGP (signal processor) instruction for a restart. The restart IH routes control to DSS (dynamic support system), R/TM (recovery/termination management) or both.
- Program interruptions, which may be caused by program errors (invalid operation, protection exception, segment exception); page fault (caused by referencing a page not in main storage); event monitoring (caused by a monitor call instruction called an MC or a program event recording interruption, called a PER). The program IH determines the cause of the interruption, and does one or more of the following:
  - Calls Real Storage Mangement 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 GTF for tracking.
  - Calls R/TM if the program exception appears to be a program error.
  - Sets up to give control to a user's SPIE exit.

Interprocessor Communications (IPC)

Interprocessor communications 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 CPU or causing special routines to be invoked on another CPU.

The signal service routines perform two different types of signal services — direct and remote. Direct signal service, invoked via the DSGNL macro instruction, uses the SIGP (signal processor) instruction to modify, sense or alter the physical state of one or more CPUs.

The SIGP functions are:

- External call
- Emergency signal
- Start
- Stop
- Sense
- Program reset
- Initial program reset
- Stop and store status
- Initial microprogram load
- Initial CPU reset
- CPU reset
- Restart.

Remote services route control to specified routines on one or more CPUs, using the emergency signal and external call functions of DSGNL to issue the signals. There are two types of remote signal functions: Remote Immediate Signal, provided via the RISGNL macro and Remote Pendable Signl, provided via the RPSGNL macro. They cause designated receiving routines to receive control on a specified CPU. A comparison of the two functions follows:

	Remote Immediate	Remote Pendable
1.	The entry point to the receiving routine is provided by the issuer of the RISGNL.	The receiving routines are predefined to the system. The issuer of the RPSGNL designates which one of those routines is to receive control.
2.	The receiving routine is synchronized with the sending routine. If the caller desginates a "serial" request, the sender will not receive control back from the RISGNL routine until the receiving routine has completed. If the caller designates a "parallel" request, then the caller will receive control after the signal was received on the other CPU, but will be able to operate at the same time as the receiving routine.	The receiving routine is not synchronized with the sending routine. The receiving routine cannot receive control until the CPU is enabled for external interruptions. The sending routine cannot be ensured that the receiving routine on the other CPU received the signal.
3.	An Emergency Signal (EMS) class of external interrupt is generated on the receiving CPU.	The External Call class of external interrupt is generated on the receiving CPU.
4.	The receiving routines for both RISGNL and RPSGNL operate as subroutines of the external first level interruption handler.	

Remote Pendable Signal Operation

The Remote Pendable signal function consists of three object modules:

- IEAVERP (executing in the sending CPU) tells the receiving CPU what functions to perform.
- IEAVEDR (Direct Signal) also executing in the sending CPU, issues the external call SIGP.
- IEAVEXS (the External Call SLIH) which receives the signal and routes control to the receiving routine executes in the receiving CPU.

Remote Immediate Signal Operation

The Remote Immediate signaling function consists of:

- IEAVERI (remote immediate), executing in the sending CPU, sets up one interface to the receiving routine.
- IEAVEDR (Direct Signal) executing in the sending CPU issues the emergency signal SIGP instruction.
- IEAVEES (the EMS SLIH) receives the signal executing in the receiving CPU and routes control to the receiving routine.

Scheduling Exit Routines

A user program may request the future execution of an exit routine to handle an asynchronous event, such as an 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. (See Figure 2-34).
2. The user must set up the interface to 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 will contain the address of the DEB, which will contain the TCB and IRB addresses.
   c) SRB. This is used by only IOS when scheduling a non-resident Error Recovery Procedure. In each address space there is a pre-determined task designated as the error task. (Its address is contained in ASXBETSK. Each address space also has a preformatted System IRB (SIRB). An SRB passed to Stage 2 Exit Effector represents a request to schedule the SIRB to the error task. The SIRB will always give control to the IOS error recovery procedure loader.

   The user branch enters 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. (See Figure 2-34).
3. The user will eventually lose control, and the dispatcher will be entered. 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 will then process the queued requests. Stage 3 dequeues the requests (IQE, RQE, or SRB) from the asynchronous exit queue and places the IRB on that task's RB chain. (See Figure 2-34).

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

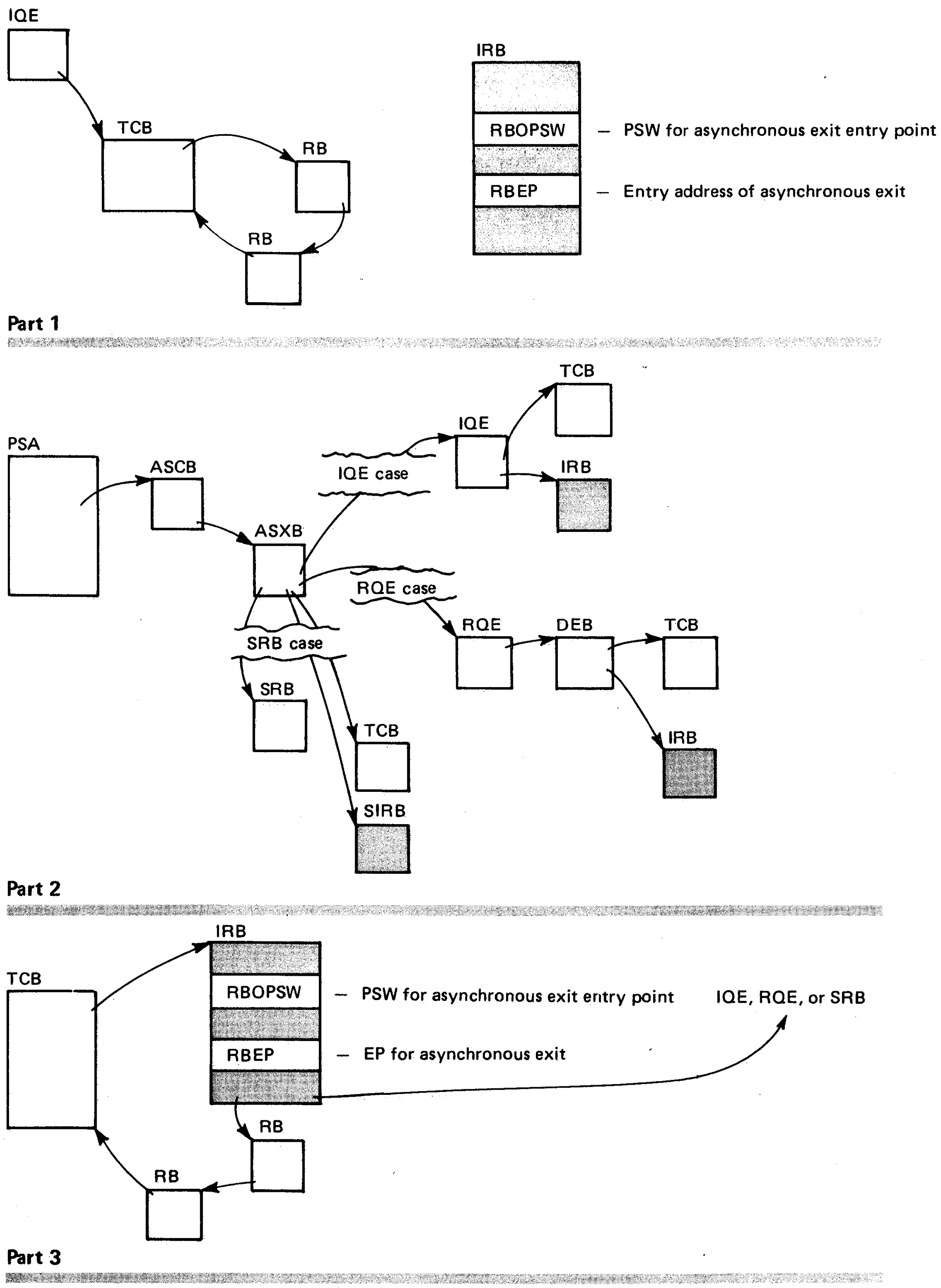


Figure 2-34. Asynchronous Exit Effector Data Structure

Serializing System Resources

In a multiprocessing system, some method of serialization must be employed to prevent interference between CPUs competing for a resource. OS/VS2 utilizes "locking" to serialize resources.

Locks consist of two types - spin and suspend. A request for a spin lock causes a disabled loop on the CPU until the lock becomes available if it cannot be immediately obtained. A request for a suspend lock suspends the requester, if it cannot be obtained immediately, to allow that CPU to process other work. The local and CMS locks are suspend type locks; all others are spin locks. The owner of a spin lock must run disabled (cannot take a page fault or I/O or external interruptions). The owner of a suspend lock may run enabled (taking page faults and I/O or external interruptions).

Since a request for a spin lock results in a disabled loop until the other CPU releases the lock, some mechanism is necessary to receive an emergency signal or malfunction alert interrupt in the event of a machine failure on the other CPU.

Therefore, in the course of spinning, they must open a "window" or a series of instructions that enable for those interruptions. The WINDOW macro instruction provides this facility.

To prevent deadlocks between CPUs, the locks must be requested in a specific order. For this reason, a lock hierarchy is defined. Certain spin locks, called class locks, have multiple locks at a specific level in the hierarchy. (For example, there is one lock per UCB.) The caller of SETLOCK, for a class lock request, must supply the lockword address.

The following locks have been defined and are listed in hierarchical order, highest first:

Dispatcher (DISP) - This is a global, spin type lock. Its function is to serialize all functions associated with dispatching.

Auxiliary Storage Management (ASM) - This is a global spin class lock used by ASM for global serialization.

Space Allocation (SALLOC) - This is a global, spin type lock. It will serialize the global portions of real storage management (RSM) and virtual storage management (VSM).

IOS SYNCHRONIZATION (IOSSYNCH) - This is a global spin class lock. This lock serializes the IOS Purge function and other parts of IOS.

I/O Supervisor Channel Availability Table (IOSCAT) - This is a global spin class lock. There is only one. IOS uses this lock when selecting a channel.

IOS Unit Control Block (IOSUCB) - This is a global spin class lock. There is one of these locks per UCB. IOS uses this lock to serialize the changing of status in the UCB.

IOS Logical Channel Queue (IOSLCH) - This is a global spin class lock. There is one of these per logical channel.

System Resource Manager (SRM) - This is a global spin lock. It is used by the SRM to serialize its control blocks when not using CS.

Cross Memory Services (CMS) - This is a global, suspend lock. This lock will be used by all other global functions in the system. This is the only enabled global lock. The local lock *must* be held when this lock is requested, and not released before CMS.

Local - There is one local lock per address space. It is a suspend lock used by functions needing to serialize address space related resources. The lockword for this lock is in the ASCB for the private address space.

The hierarchy scheme works as follows:

- May only unconditionally request lock(s) higher in the hierarchial structure, than lock(s) currently held.
- May only request locks of type different from locks already held (e.g., may not request IOSUCB if already hold a different IOSUCB lock).
- It is not necessary to hold any locks lower in the hierarchy.
- Owning the CMS lock requires that the Local Memory lock be held.

Supervisor Control Recovery

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

- Direct interface with R/TM
- Normal FRR stack
- Supervisor control FRR stack

Special Interface With RTM: There are a number of routines (IEAVELCR, IEAVELKR, IEAVEVRR) called on every entry to R/TM 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.

IEAVEPDR - PURGEDQ FRR and ESTAE

IEAVEVAL - Validity check FRR
IEAVEPC - Program IH SPIE processing
IEAVEIPR - IPC recovery
IEAVELKR - Setlock FRR
IEAVEVRR - ASVT reconstruct FRR

Super Stack Mechanism: In order to bypass SETFRR overhead on high-performance paths, a multiple FRR stack mechanism was employed to provide recovery for Supervisor Control routines.

Control Structure For Multiple Stacks

There is a pointer in the PSA to the FRR stack that this CPU is using currently. When an error occurs, R/TM will route control only to FRRs on that stack. (See the Recovery/Termination Management section for a description of routing to FRRs.)

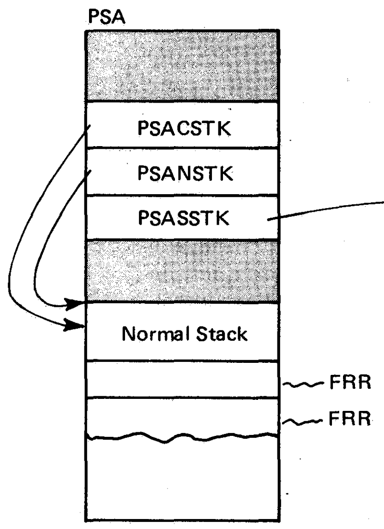
For each CPU there are 8 FRR stacks - a normal stack and 7 "super stacks", which are used to provide recovery for supervisor control functions. The current stack pointer will always point to one of the stacks. (See Figure 2-35.)

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. (See Figure 2-35).

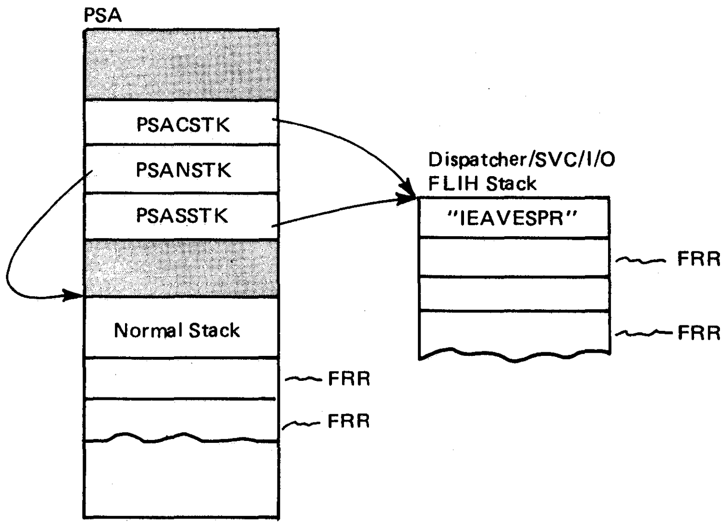
If a routine called by a supervisor control function issues a SETFRR, the FRR entry will appear on the super stack. If an error occurs while a super stack is current, then R/TM will first route control to all the FRRs on that stack and will then route control to the Super FRR Routine (IEAVESPR). (See Figure 2-35.)

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.



Part 1



Part 2

Figure 2-35. Supervisor Control Recovery Data Structure

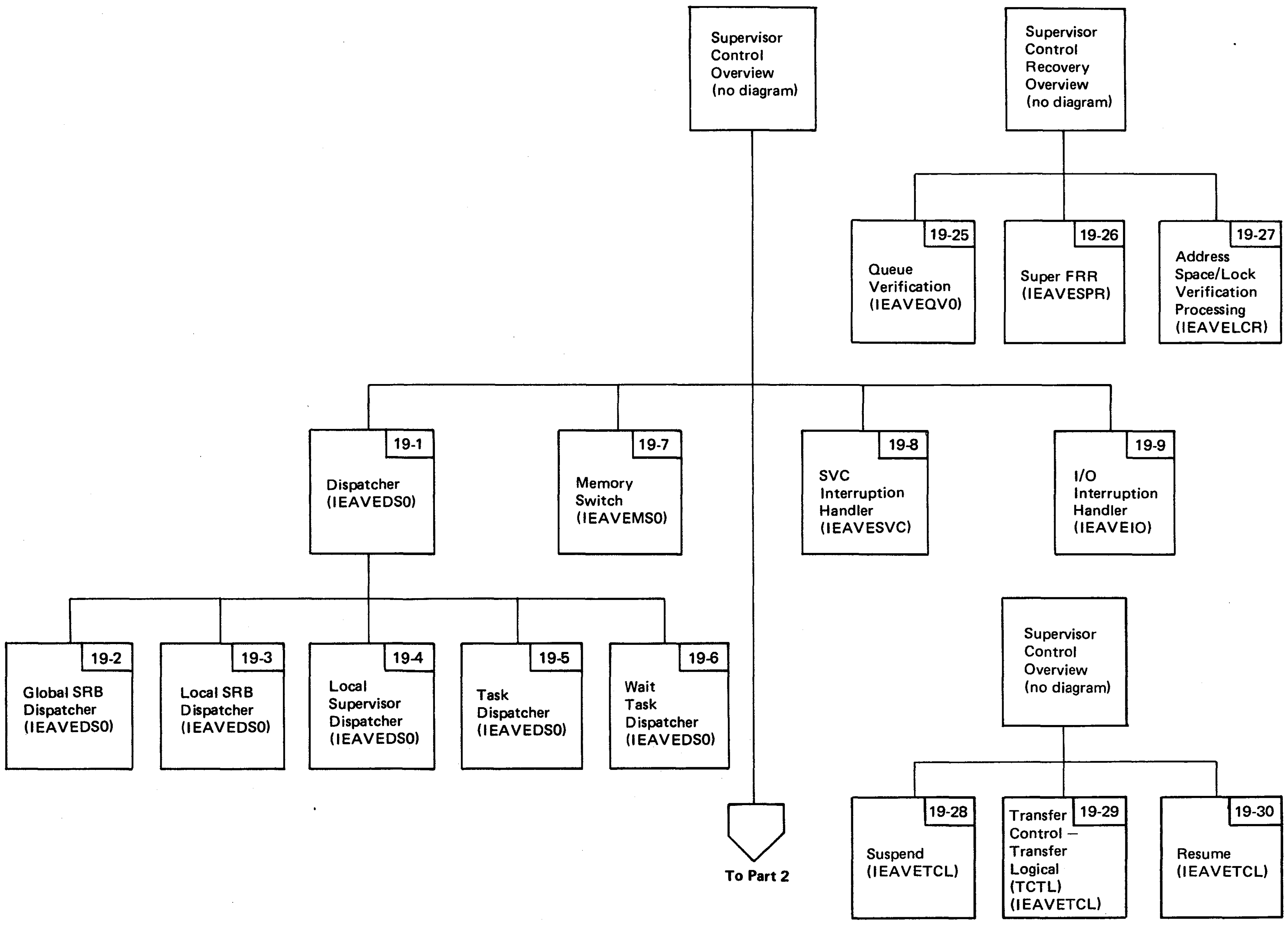
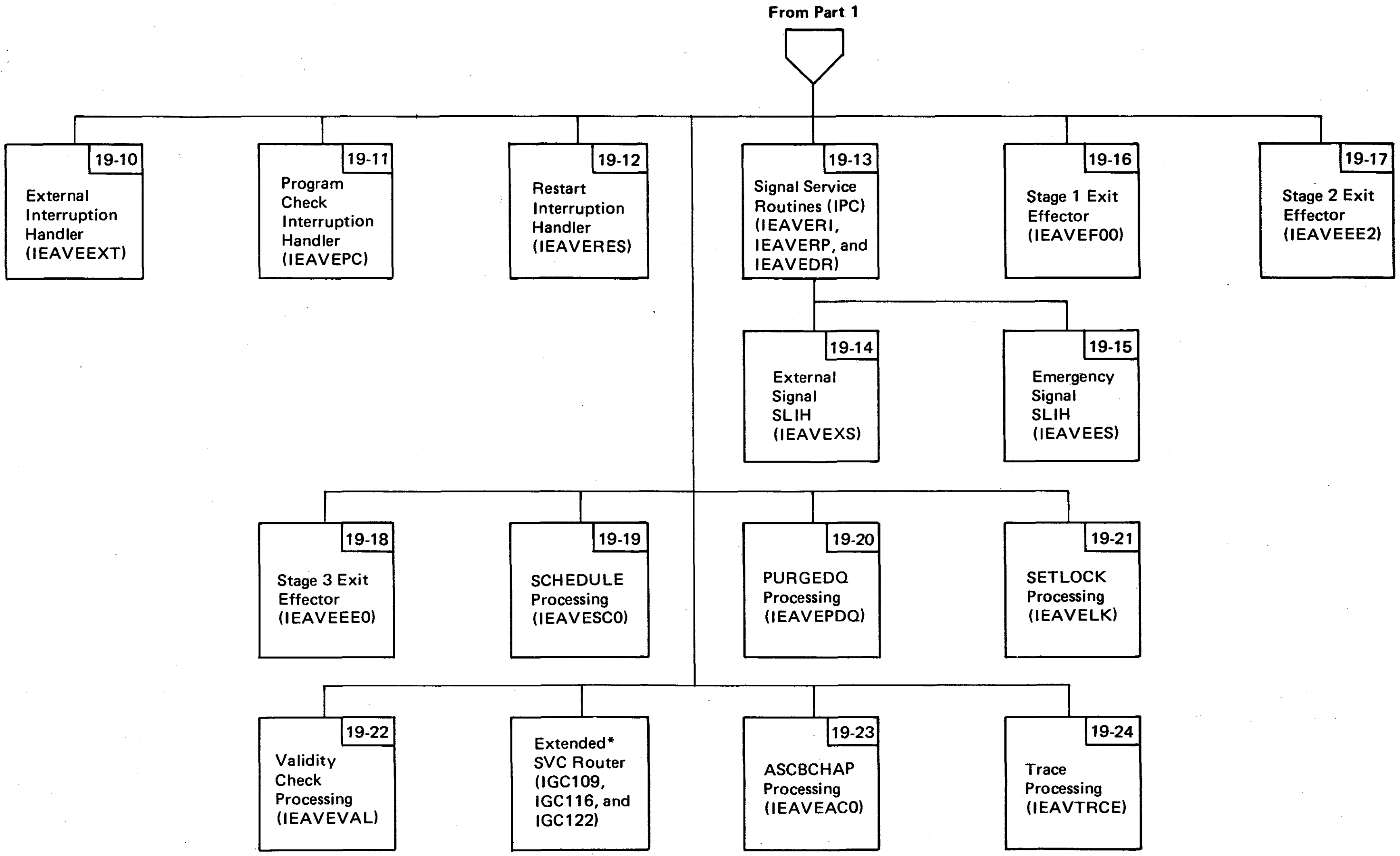

Supervisor Control Overview (no diagram)
Supervisor Control Recovery Overview (no diagram)
19-25 Queue Verification (IEAVEQV0)
19-26 Super FRR (IEAVESPR)
19-27 Address Space/Lock Verification Processing (IEAVELCR)
19-1 Dispatcher (IEAVEDS0)
19-7 Memory Switch (IEAVEMS0)
19-8 SVC Interruption Handler (IEAVESVC)
19-9 I/O Interruption Handler (IEAVEIO)
19-2 Global SRB Dispatcher (IEAVEDS0)
19-3 Local SRB Dispatcher (IEAVEDS0)
19-4 Local Supervisor Dispatcher (IEAVEDS0)
19-5 Task Dispatcher (IEAVEDS0)
19-6 Wait Task Dispatcher (IEAVEDS0)
To Part 2
Supervisor Control Overview (no diagram)
19-28 Suspend (IEAVETCL)
19-29 Transfer Control — Transfer Logical (TCTL) (IEAVETCL)
19-30 Resume (IEAVETCL)

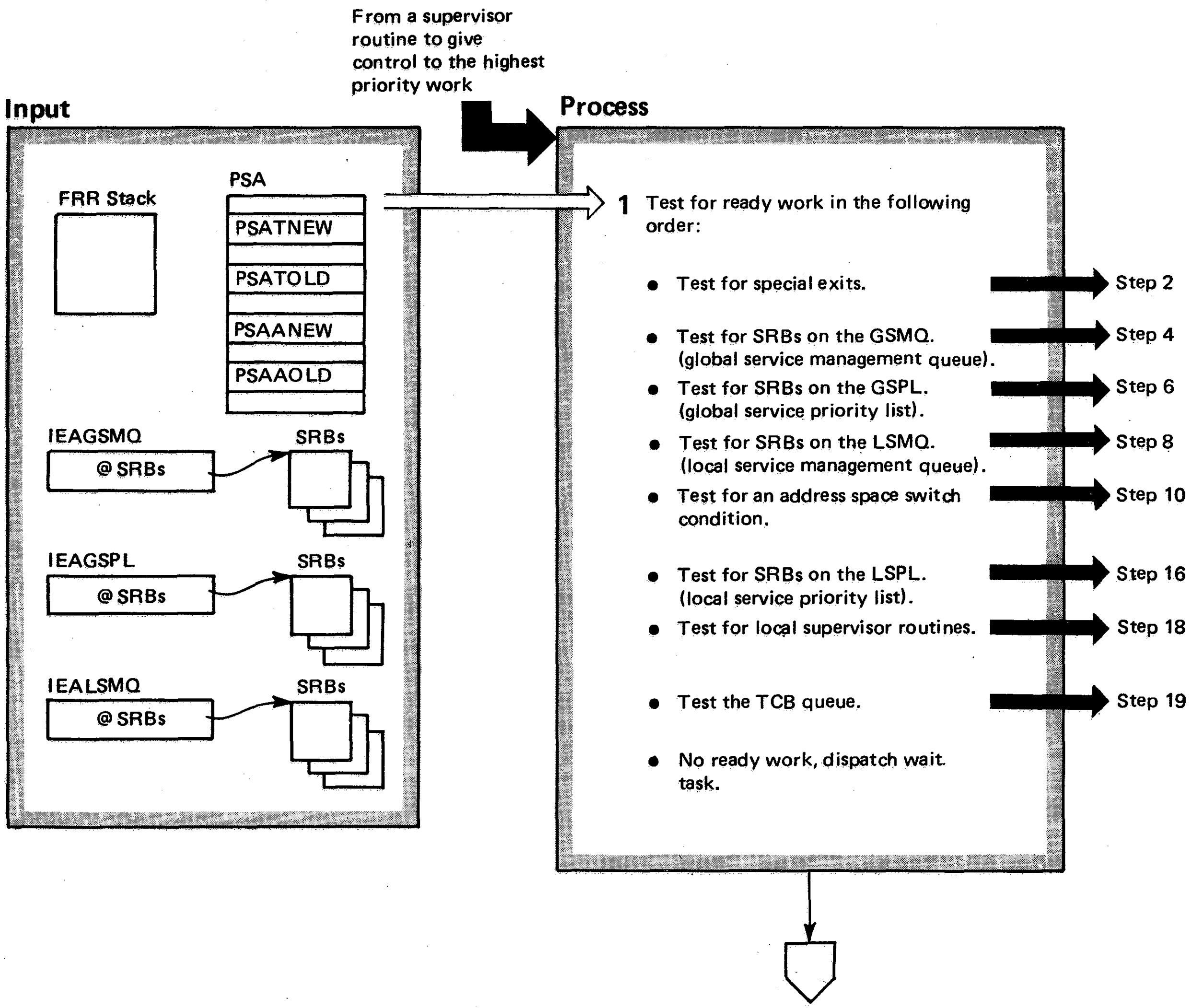
Figure 2-36. Supervisor Control Visual Contents (Part 1 of 2)


From Part 1
19-10
External Interruption Handler (IEAVEEXT)
19-11
Program Check Interruption Handler (IEAVEPC)
19-12
Restart Interruption Handler (IEAVERES)
19-13
Signal Service Routines (IPC) (IEAVERI, IEAVERP, and IEAVEDR)
19-16
Stage 1 Exit Effector (IEAVEF00)
19-17
Stage 2 Exit Effector (IEAVEEE2)
19-14
External Signal SLIH (IEAVEXS)
19-15
Emergency Signal SLIH (IEAVEES)
19-18
Stage 3 Exit Effector (IEAVEEE0)
19-19
SCHEDULE Processing (IEAVESC0)
19-20
PURGEDQ Processing (IEAVEPDQ)
19-21
SETLOCK Processing (IEAVELK)
19-22
Validity Check Processing (IEAVEVAL)
Extended* SVC Router (IGC109, IGC116, and IGC122)
19-23
ASCBCHAP Processing (IEAVEAC0)
19-24
Trace Processing (IEAVTRCE)

*These diagrams are discussed with the SVC First Level Interrupt Handler.

Figure 2-36. Supervisor Control Visual Contents (Part 2 of 2)

Diagram 19-1. Dispatcher (IEAVEDS0) (Part 1 of 18)



1 Test for ready work in the following order:

- Test for special exits.
- Test for SRBs on the GSMQ. (global service management queue).
- Test for SRBs on the GSPL. (global service priority list).
- Test for SRBs on the LSMQ. (local service management queue).
- Test for an address space switch condition.
- Test for SRBs on the LSPL. (local service priority list).
- Test for local supervisor routines.
- Test the TCB queue.
- No ready work, dispatch wait task.

Diagram 19-1. Dispatcher (IEAVEDS0) (Part 2 of 18)

Extended Description	Module	Label
The dispatcher selects the highest priority ready work from various queues and gives it control. The ready work that the dispatcher searches for can be either service requests – represented by SRBs – or tasks – represented by TCBs. The dispatcher searches for work in a particular order, by first searching for ready SRBs, and next searching for TCBs. This ensures that the most important work in the system receives control first. When the dispatcher finds ready work, the status of the previous work is saved and job step timing is completed. The dispatcher receives control at the following entry points:		
• IEA0DS. This is the main dispatcher entry point.		
• IEAPDS2. The SETLOCK suspend routine enters the dispatcher at this entry point.		
• IEAPDS6. EXIT processing uses this entry point for end-of-task processing.		
• IEAPDSRT. SRBs return to the dispatcher at this entry point.		
• IEAPDS7. The I/O FLIH, SVC FLIH use this entry point.		
1 The dispatcher searches for ready work in the order indicated. The dispatcher follows this sequence to dispatch ready work:	IEAVEDS0	
• Give control to special exits.		
• Dispatch a GLOBAL SRB.		
• Redispatch a suspended or local SRB.		
• Dispatch a locally locked routine.		
• Dispatch a task.		

Diagram 19-1. Dispatcher (IEAVEDS0) (Part 3 of 18)

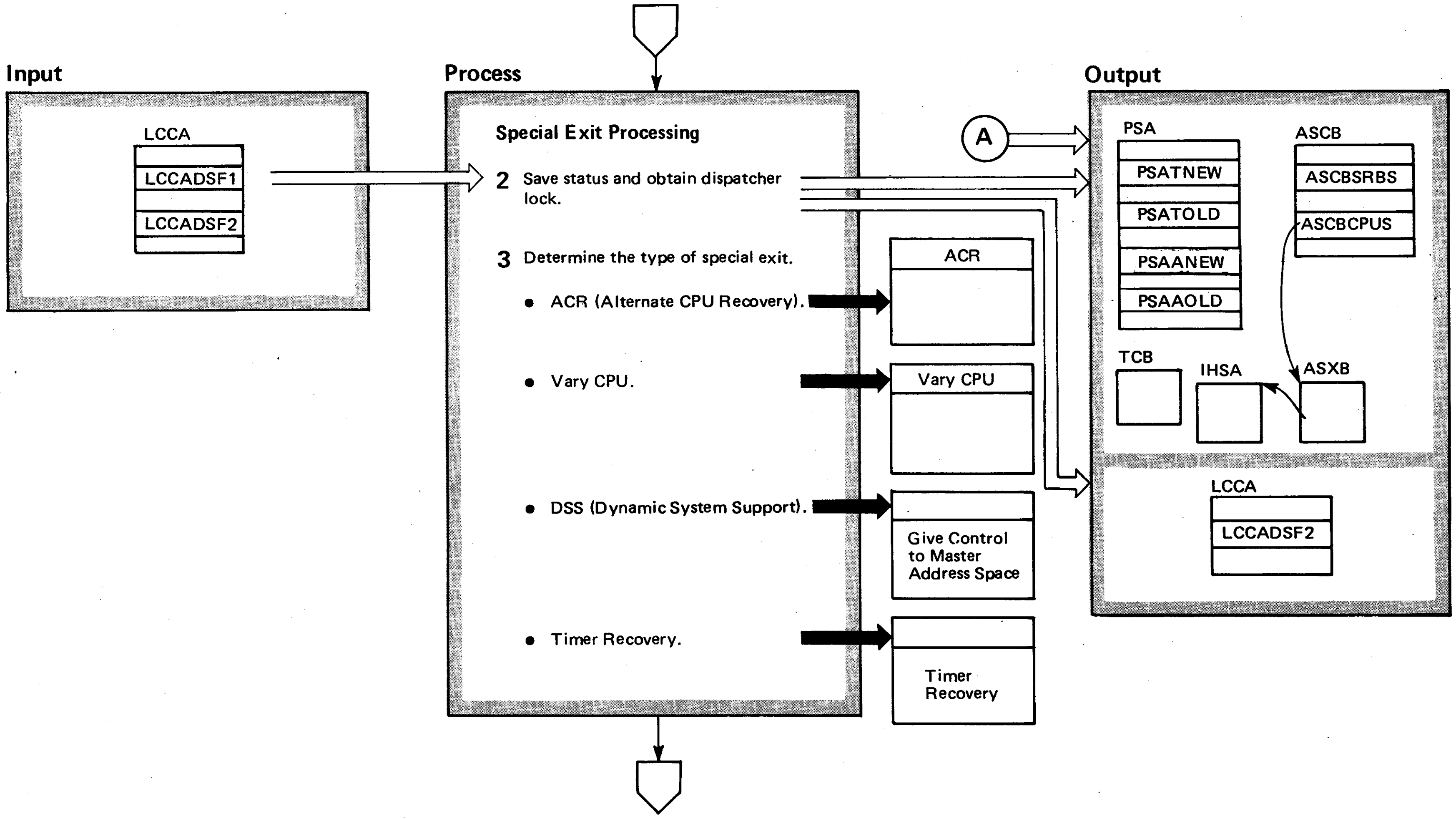


Diagram 19-1. Dispatcher (IEAVEDS0) (Part 4 of 18)

Extended Description	Module	Label
2 Special exits require immediate response. The dispatcher checks first for this condition, and then saves the status of the interrupted task, and obtains the dispatcher lock.		
3 The dispatcher determines the type of special exit, and gives it control except for DSS. In that case, control is given to the Memory Switch routine to switch to the master address space. Special exit returns control to the dispatcher.		

Diagram 19-1. Dispatcher (IEAVEDS0) (Part 5 of 18)

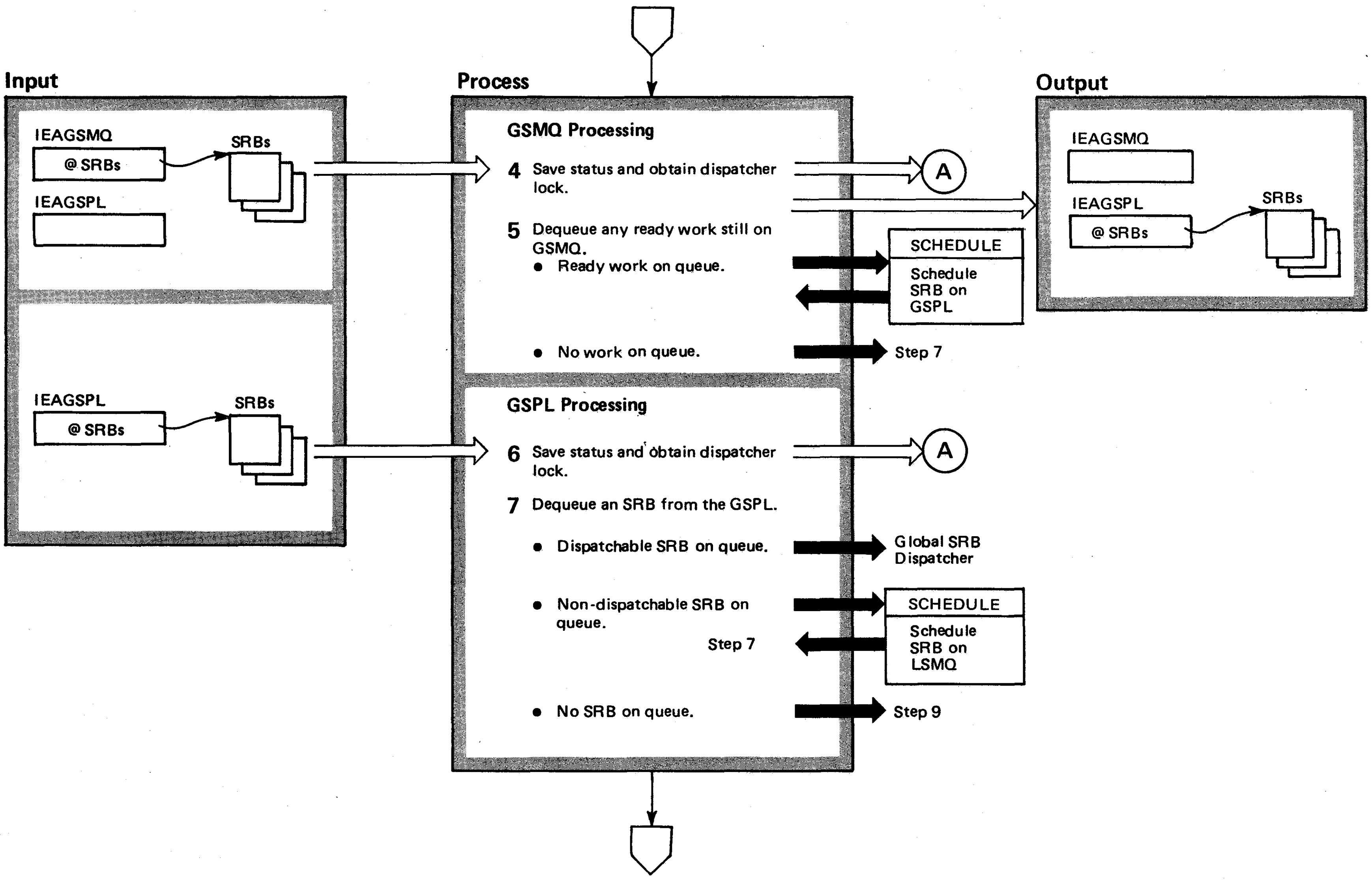
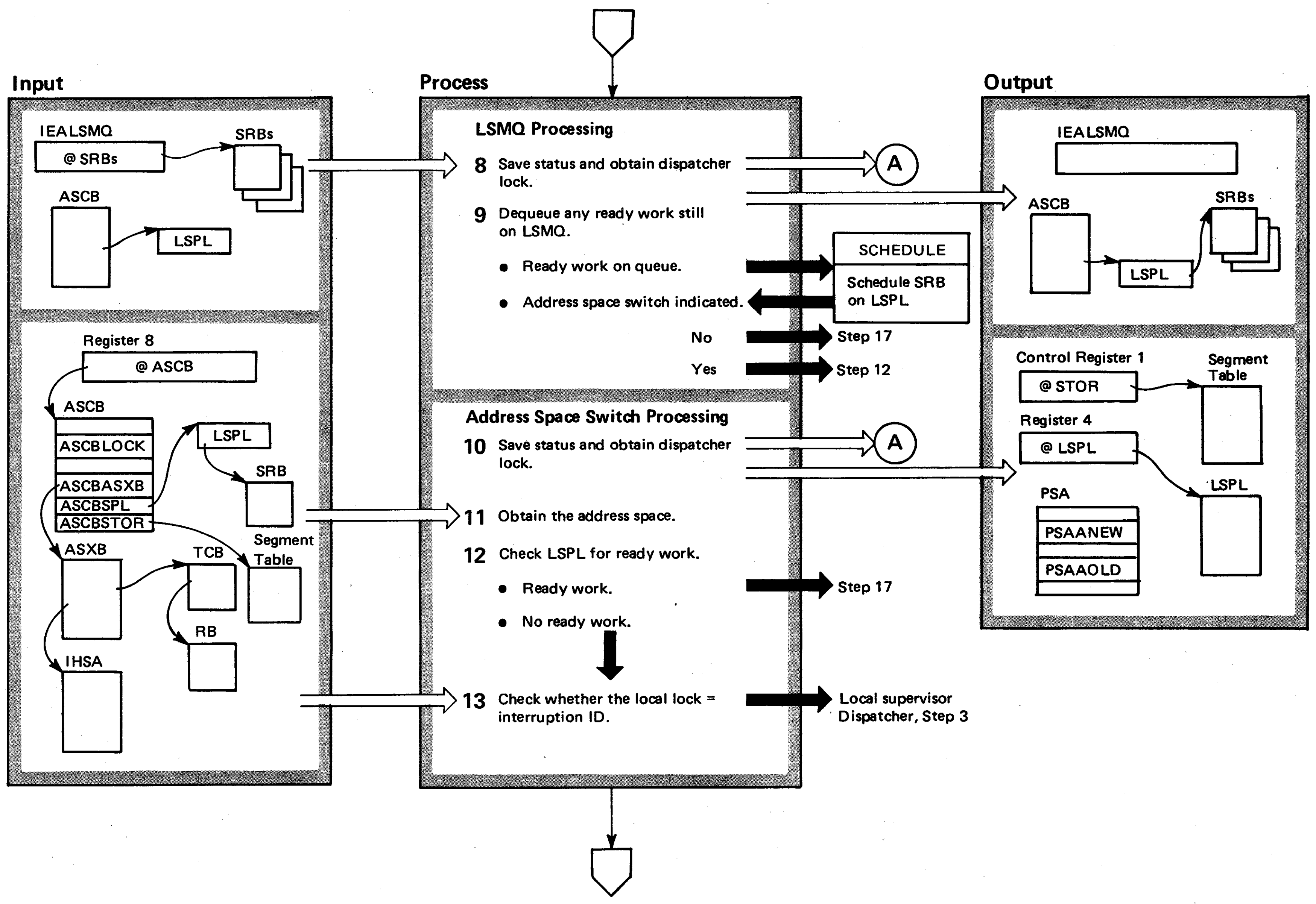


Diagram 19-1. Dispatcher (IEAVEDS0) (Part 6 of 18)

Extended Description	Module	Label
4 The dispatcher next checks the GSMQ (global service management queue) for any ready work. The dispatcher saves the status of the interrupted program and obtains the necessary locks if it finds SRBs on the GSMQ.		
5 The dispatcher dequeues any ready work on the GSMQ via the CS (Compare and Swap) instruction. Work on the GSMQ will be placed, via the SCHEDULE service, on the GSPL (global service priority list). If the GSMQ has no ready work, the dispatcher next checks for work on the GSPL, as indicated by step 7.		
6 The third check the dispatcher makes is for ready work on the GSPL. The dispatcher saves the status of the interrupted program and obtains the necessary locks if it finds ready work.		
7 The dispatcher dequeues any ready work (the first dispatchable SRB) on the GSPL and gives control to the global SRB dispatcher subroutine. If an SRB is not immediately dispatchable, it is dequeued and scheduled to the LSMQ. If the GSPL has no ready work, the dispatcher next checks for work on the LSMQ (local service manager queue), as indicated by step 9.		

Diagram 19-1. Dispatcher (IEAVEDS0) (Part 7 of 18)



LSMQ Processing

8 Save status and obtain dispatcher lock.

9 Dequeue any ready work still on LSMQ.

- Ready work on queue.
- Address space switch indicated.

No → Step 17

Yes → Step 12

Address Space Switch Processing

10 Save status and obtain dispatcher lock.

11 Obtain the address space.

12 Check LSPL for ready work.

- Ready work. → Step 17
- No ready work.

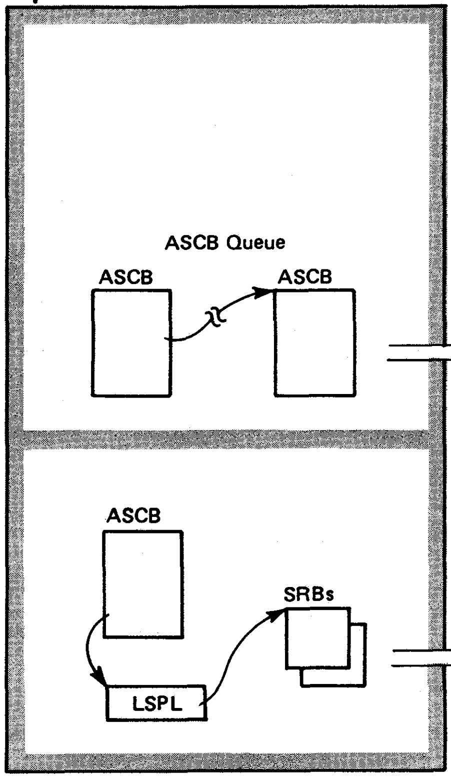
13 Check whether the local lock = interruption ID. → Local supervisor Dispatcher, Step 3

Diagram 19-1. Dispatcher (IEAVEDS0) (Part 8 of 18)

Extended Description	Module	Label
8 The dispatcher saves the status of the interrupted program and obtains the necessary locks.		
9 The dispatcher dequeues any ready work on the LSMQ (via the CS instruction) and schedules it to be placed on the LSPL (local service priority list). When the LSMQ has no ready work, control goes to step 12 if an address space switch has been indicated.		
10 The dispatcher saves the status of the interrupted program and obtains the dispatcher lock.		
11 If ready work exists in the new address space, the dispatcher updates the PSAANEW and PSAAOLD to reflect the new address space, and loads the STOR (segment table origin register) for that address space.		
12 The dispatcher then checks the LSPL for ready work. If the LSPL has ready work queued on it, control goes to step 17; otherwise, processing continues at step 13.		
13 By checking the local lock lockword for an interruption ID, the dispatcher can determine whether a local supervisor routine was processing (Note: A local supervisor routine would be a supervisory-type service, such as ATTACH, performing a service needed at a local level, such as by a problem program). If the dispatcher finds an interruption ID, control goes to the local supervisor dispatcher subroutine.		

Diagram 19-1. Dispatcher (IEAVEDS0) (Part 9 of 18)

Input



Process

14 Determine whether the local lock contains this CPU's ID.

- ID in local lock = CPU. → Local Supervisor Dispatcher, Step 3.f
- Local lock available. → Step 19
- Local lock not available, continue.

15 Check ASCB queue.for end.

- End of queue. → Wait Task Dispatcher
- Next ASCB. → Step 11

LSPL Processing

16 Save status and obtain dispatcher lock. → A

17 Dequeue first dispatchable SRB from the LSPL.

- Ready work on queue. → Local SRB Dispatcher
- No dispatchable work on queue.

Local Supervisor Routine Processing

18 Determine whether this is a local supervisor routine. → Local Supervisor Dispatcher, Step 1 (Interrupt ID) or Step 3.f (CPU ID)

Diagram 19-1. Dispatcher (IEAVEDS0) (Part 10 of 18)

Extended Description	Module	Label
14 If the local lock contains the CPU ID, this indicates that a local supervisor routine received an interruption and status has not yet been saved. This routine can be redispatched immediately (step 18). If the local lock does not contain the CPU ID, the dispatcher attempts to obtain the local lock, via a CS instruction. If the dispatcher obtains the local lock, the task dispatcher receives control (step 19).		
15 When the CS instruction fails, the dispatcher tests if this is the WAIT ASCB. If this is the WAIT ASCB, and a recursive search of the dispatching queues, the dispatcher gives control to the WAIT ASCB. If this is the WAIT ASCB, but not a recursive search through the dispatching queues, the dispatcher searches the dispatching queues for a second time. If this ASCB is not the WAIT ASCB, control goes to step 10 to dispatch the next ASCB.		
16 The dispatcher saves the status of the interrupted program and obtains the necessary locks.		
17 The dispatcher dequeues any ready work on the LSPL and gives control to the local SRB dispatcher. If the SRB is a suspended SRB (from local lock or page fault suspension processing), the dispatcher restores status from the SSRB and redispatches the SRB. Otherwise, the dispatcher uses the global SRB dispatcher to dispatch the SRB. If the LSPL has no ready work, control goes to step 18.		
18 The dispatcher determines whether a local supervisor routine should receive control, and gives control to the local supervisor dispatcher subroutine, if necessary.		

Diagram 19-1. Dispatcher (IEAVEDS0) (Part 11 of 18)

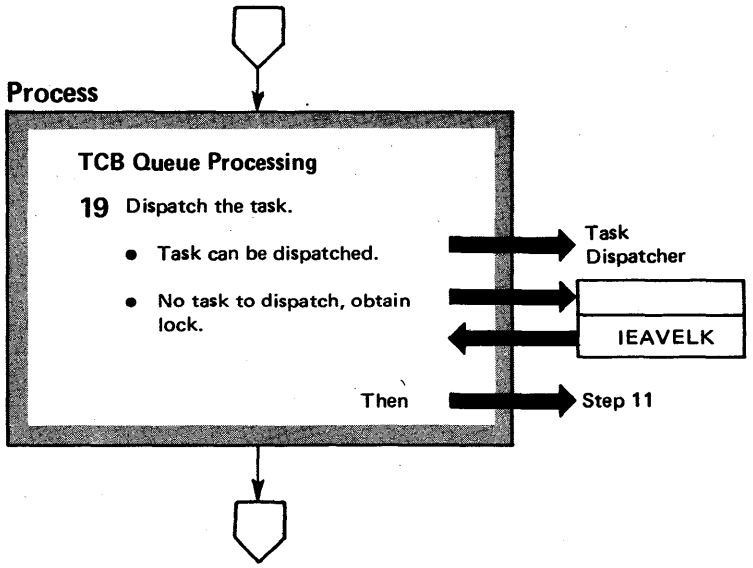


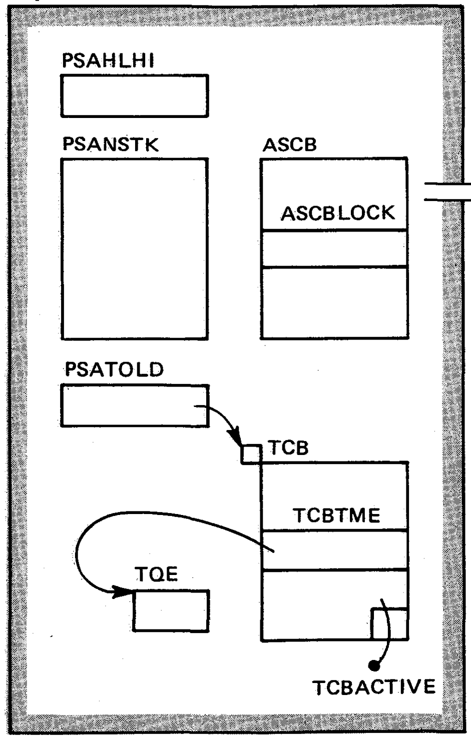
Diagram 19-1. Dispatcher (IEAVEDS0) (Part 12 of 18)

Extended Description	Module	Label
19 The dispatcher checks for ready work on the TCB queue, and gives control to the task dispatcher to dispatch ready tasks. The dispatcher tests the ASCBS3S field of the ASCB to determine whether the stage 3 exit effector has any asynchronous exits to process.		
The dispatcher will always begin searching from the top of the TCB ready queue. If it finds a dispatchable TCB, it tests this TCB to determine whether the task is active on another CPU. If the TCB is active, the dispatcher searches for the next ready TCB.		
The dispatcher does not save the status of TCBs active on the current CPU; these TCBs can be redispatched after restoring the registers and PSW. After there are no more ready tasks left on the TCB queue, control goes to step 11 to process any ready work in any address spaces.		

Diagram 19-1. Dispatcher (IEAVEDS0) (Part 13 of 18)

From Dispatcher (IEAVEDS0) when ready work has been found

Input



Process

20 Save Status Routine

- Wait task
- Unlocked task
  - Save FPRs in TCB prefix.
  - Save task TQE CPU timer value, if one exists.
  - Clear task active and CPU-id indicators.
  - Call the job step timing subroutine.
- Locked task
  - Save FPRs in IHSA.
  - Save FRR stack in IHSA.
  - Save CPU timer value in IHSA.
  - Save PSATNEW/PSATOLD in IHSA.
  - If CMS lock held, set indicator in ASCB.
  - Compare and swap interrupt ID in local lock.
  - Clear 'CPU locks held' indicators (PSAHLHI).
  - Call job step timing subroutine.
- For all three modes
  - Clear PSATNEW/PSATOLD.
  - Decrement ASCBCPUS.
  - Return to caller.

To Caller

Output

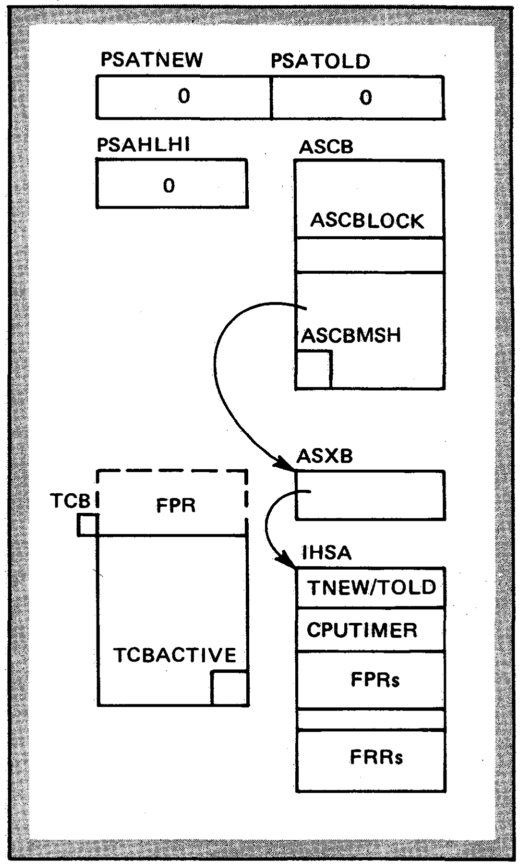
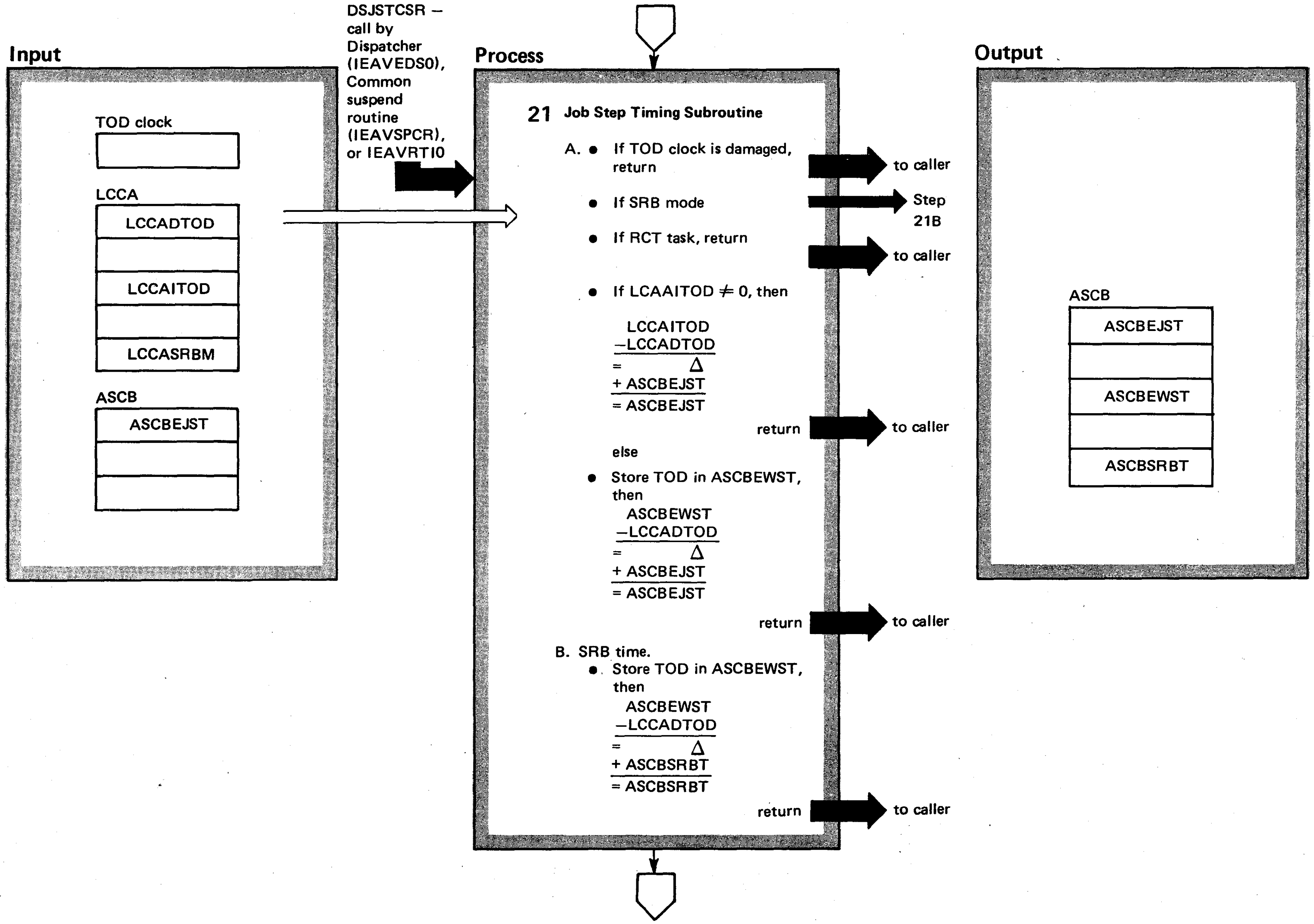


Diagram 19-1. Dispatcher (IEAVEDS0) (Part 14 of 18)

Extended Description	Module	Label
20 When the dispatcher finds ready work of higher priority than the current work or if the current work is no longer dispatchable, the status of the current work is saved and its elapsed job step time is calculated and accumulated. The type of status saved and where it is saved depends on whether the current work is the wait task, an unlocked task, or a locked task. SRB status is never saved in the dispatcher since a SRB is non-preemptable. When a SRB returns to the dispatcher entry point, IEAPDSRT, the count of active SRBs are decremented and the SRB mode bit is turned off.		
Whenever a SRB is suspended for a lock or a page fault, the common suspend routine (IEAVSPCR) in lock manager (IEAVELK) saves the SRB status and calls the job step timing routine (DSJSTCSR) in the dispatcher (see step 21 below).		

Diagram 19-1. Dispatcher (IEAVEDS0) (Part 15 of 18)



21 **Job Step Timing Subroutine**

A.
- If TOD clock is damaged, return
- If SRB mode
- If RCT task, return
- If LCAAITOD ≠ 0, then

$$\begin{array}{r}\text{LCCAITOD}\\ \underline{-\text{LCCADTOD}}\\ = \quad \Delta\\ \underline{+\text{ASCBEJST}}\\ =\text{ASCBEJST}\end{array}$$

return

else
- Store TOD in ASCBEWST, then

$$\begin{array}{r}\text{ASCBEWST}\\ \underline{-\text{LCCADTOD}}\\ = \quad \Delta\\ \underline{+\text{ASCBEJST}}\\ =\text{ASCBEJST}\end{array}$$

return

B. SRB time.
- Store TOD in ASCBEWST, then

$$\begin{array}{r}\text{ASCBEWST}\\ \underline{-\text{LCCADTOD}}\\ = \quad \Delta\\ \underline{+\text{ASCBSRBT}}\\ =\text{ASCBSRBT}\end{array}$$

return

Diagram 19-1. Dispatcher (IEAVEDS0) (Part 16 of 18)

Extended Description	Module	Label
21 Job step timing subroutine (DSJSTCSR). Whenever the dispatcher is switching away from current work, its elapsed job step time must be accumulated. If the dispatcher was entered from an interrupt handler, the time of day of interrupt (LCCAITOD) will have been stored on interrupt to eliminate the time spent in the interrupt handler. The dispatched time (LCCADTOD) will be subtracted from the LCAAITOD to obtain the task time. This time will be accumulated into ASCBEJST field. If the dispatcher was entered from another caller, the dispatcher will first store the TOD clock in ASCBEWST and then perform the above calculation and accumulation. SRB time is accumulated in a separate field (ASCBSRBT). Time spent in RCT's task is not accumulated to eliminate the swap out/swap in time. Job step initiation/termination, SMF, and job step time limit will use this accumulated time, initialize the value to zero, and record the appropriate value in SMF records.		

Diagram 19-1. Dispatcher (IEAVEDS0) (Part 17 of 18)

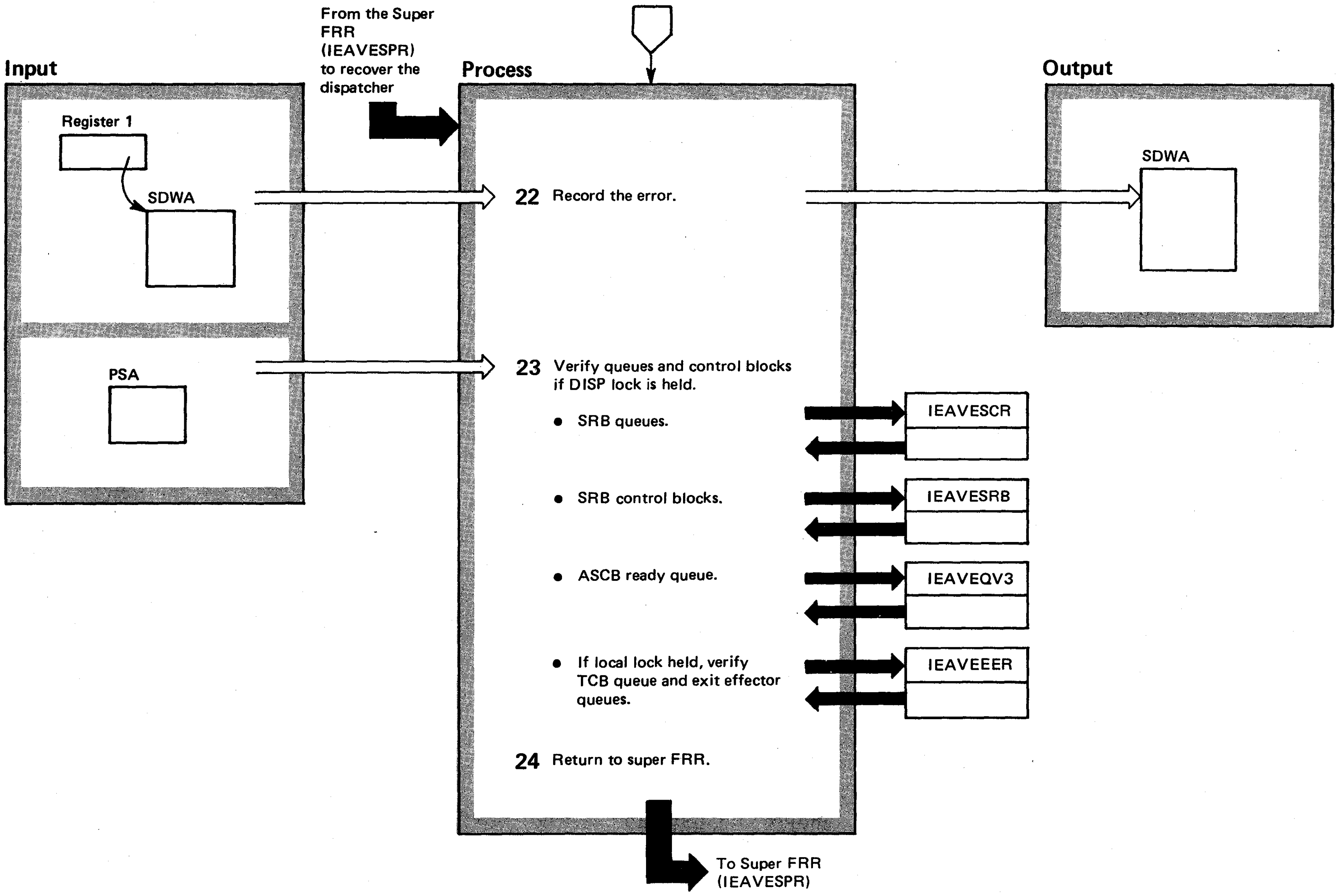


Diagram 19-1. Dispatcher (IEAVEDS0) (Part 18 of 18)

Extended Description	Module	Label
22 The dispatcher FRR records the error in the SDWA.	IEAVEDSR	
23 The dispatcher FRR verifies the queues and control blocks only if the dispatcher lock was held at the time of the error. The FRR verifies: ● The SRB queue. ● The SRB control block. ● The ASCB ready queue. The dispatcher FRR verifies the TCB queue if the local lock was held at the time of the error, and if the error was not a DAT (dynamic address translation) error and if control register 1 is valid. The dispatcher FRR routes control to the stage 3 exit effector FRR at this time.		
24 The dispatcher FRR returns control to the Super FRR.		

Diagram 19-2. Global SRB Dispatcher (IEAVEDS0) (Part 1 of 2)

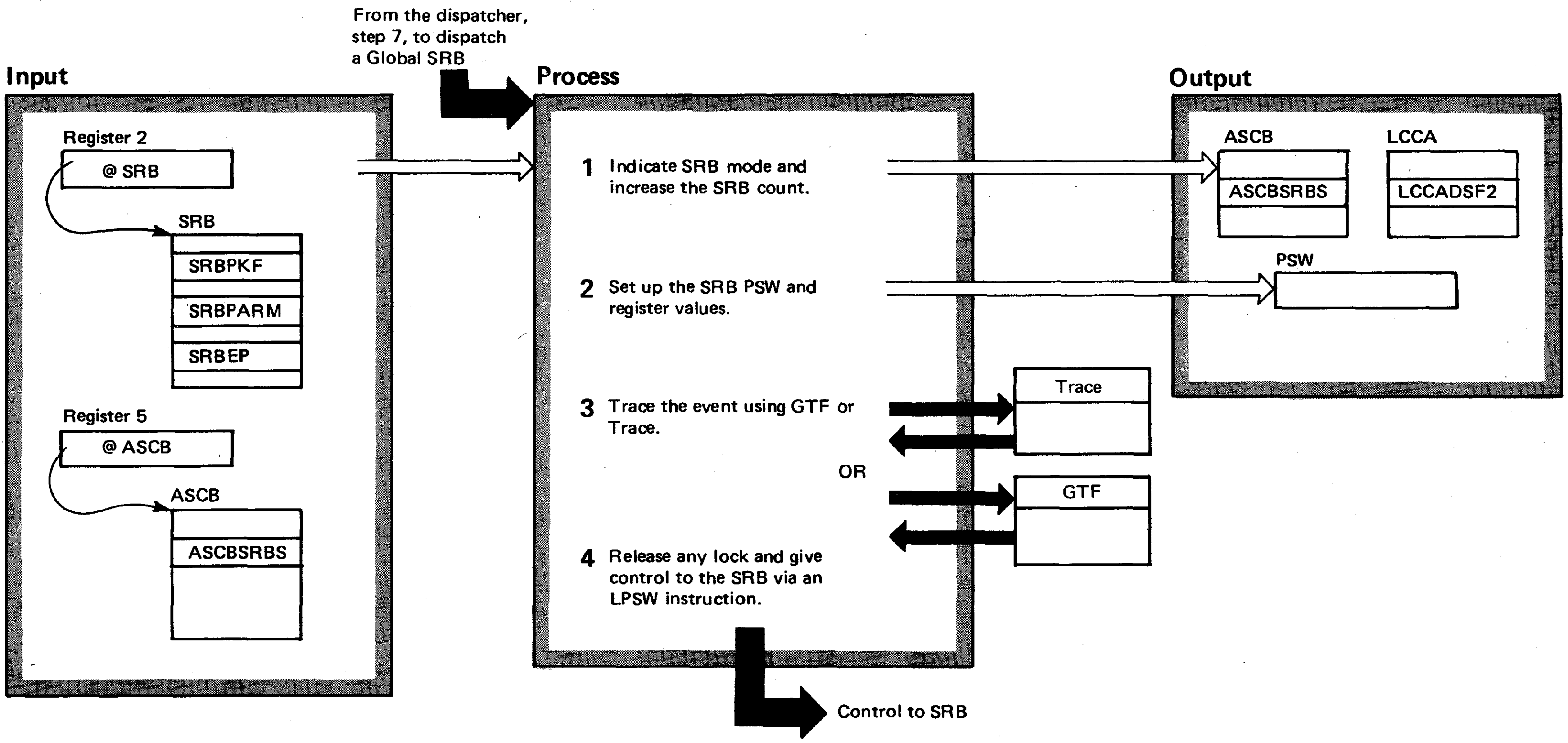

From the dispatcher, step 7, to dispatch a Global SRB
Input
Register 2
@ SRB
SRB
SRBPKF
SRBPARM
SRBEP
Register 5
@ ASCB
ASCB
ASCBSRBS
Process
1 Indicate SRB mode and increase the SRB count.
2 Set up the SRB PSW and register values.
3 Trace the event using GTF or Trace.
OR
4 Release any lock and give control to the SRB via an LPSW instruction.
Trace
GTF
Control to SRB
Output
ASCB
ASCBSRBS
LCCA
LCCADSF2
PSW

Diagram 19-2. Global SRB Dispatcher (IEAVEDS0) (Part 2 of 2)

Extended Description	Module	Label
The global SRB dispatcher subroutine of the dispatcher gives control to the ready SRB that has been dequeued from the GSPL by issuing an LPSW (load PSW) instruction.		
1 The global SRB dispatcher indicates SRB mode in the LCCA and increases the count of SRBs in the ASCBSRBS field.	IEAVEDS0	
2 Next, the global SRB dispatcher places values in the PSW that will allow the SRB to gain control. These values include desired key, supervisor state, enabled, and the SRB's entry point address. The registers contain: ● Register 0 – SRB address ● Register 1 – parameter list address ● Register 14 – return address in dispatcher ● Register 15 – entry point		
3 Either the trace routine or GTF traces the occurrence of the event.		
4 The global SRB dispatcher releases the dispatcher lock, and issues an LPSW instruction to give the selected SRB control.		

Diagram 19-3. Local SRB Dispatcher (IEAVEDS0) (Part 1 of 2)

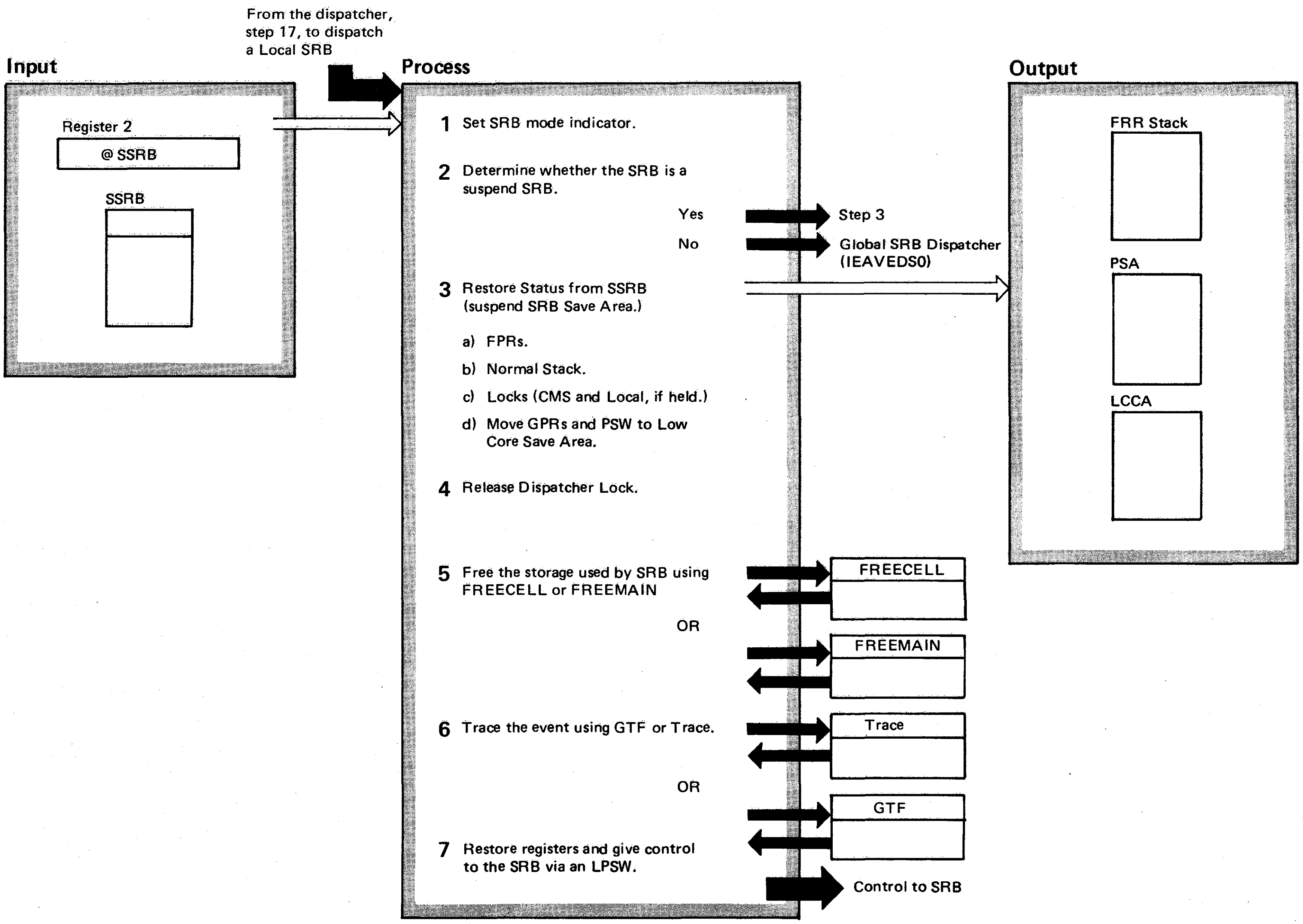

From the dispatcher, step 17, to dispatch a Local SRB
Input
Register 2
@ SSRB
SSRB
Process
1 Set SRB mode indicator.
2 Determine whether the SRB is a suspend SRB.
Yes
Step 3
No
Global SRB Dispatcher (IEAVEDS0)
3 Restore Status from SSRB (suspend SRB Save Area.)
a) FPRs.
b) Normal Stack.
c) Locks (CMS and Local, if held.)
d) Move GPRs and PSW to Low Core Save Area.
4 Release Dispatcher Lock.
5 Free the storage used by SRB using FREECELL or FREEMAIN
FREECELL
OR
FREEMAIN
6 Trace the event using GTF or Trace.
Trace
OR
GTF
7 Restore registers and give control to the SRB via an LPSW.
Control to SRB
Output
FRR Stack
PSA
LCCA

Diagram 19-3. Local SRB Dispatcher (IEAVEDS0) (Part 2 of 2)

Extended Description	Module	Label
The local SRB dispatcher subroutine of the dispatcher gives control to the ready SRBs that have been dequeued from the LSPL by using an LPSW instruction.		
1 The local SRB dispatcher first indicates SRB mode in the LCCA.	IEAVEDS0	
2 The local SRB dispatcher checks the SRBRMTR field to determine whether the SRB represents a suspended SRB (suspended for a page fault or suspend lock request). Control goes to the local SRB dispatcher for SRBs not representing suspend processing.		
3 The local SRB dispatcher restores status from the SSRB (suspended SRB) save area.		
4 The local SRB dispatcher frees the dispatcher lock.		
5 The local SRB dispatcher frees the SSRB with either FREEMAIN or FREECELL, depending on how the storage for the SSRB was obtained initially.		
6 Either the trace routine or GTF traces the occurrence of the event.		
7 The local SRB dispatcher gives control to the selected local SRB by issuing an LPSW instruction.		

Diagram 19-4. Local Supervisor Dispatcher (IEAVEDS0) (Part 1 of 2)

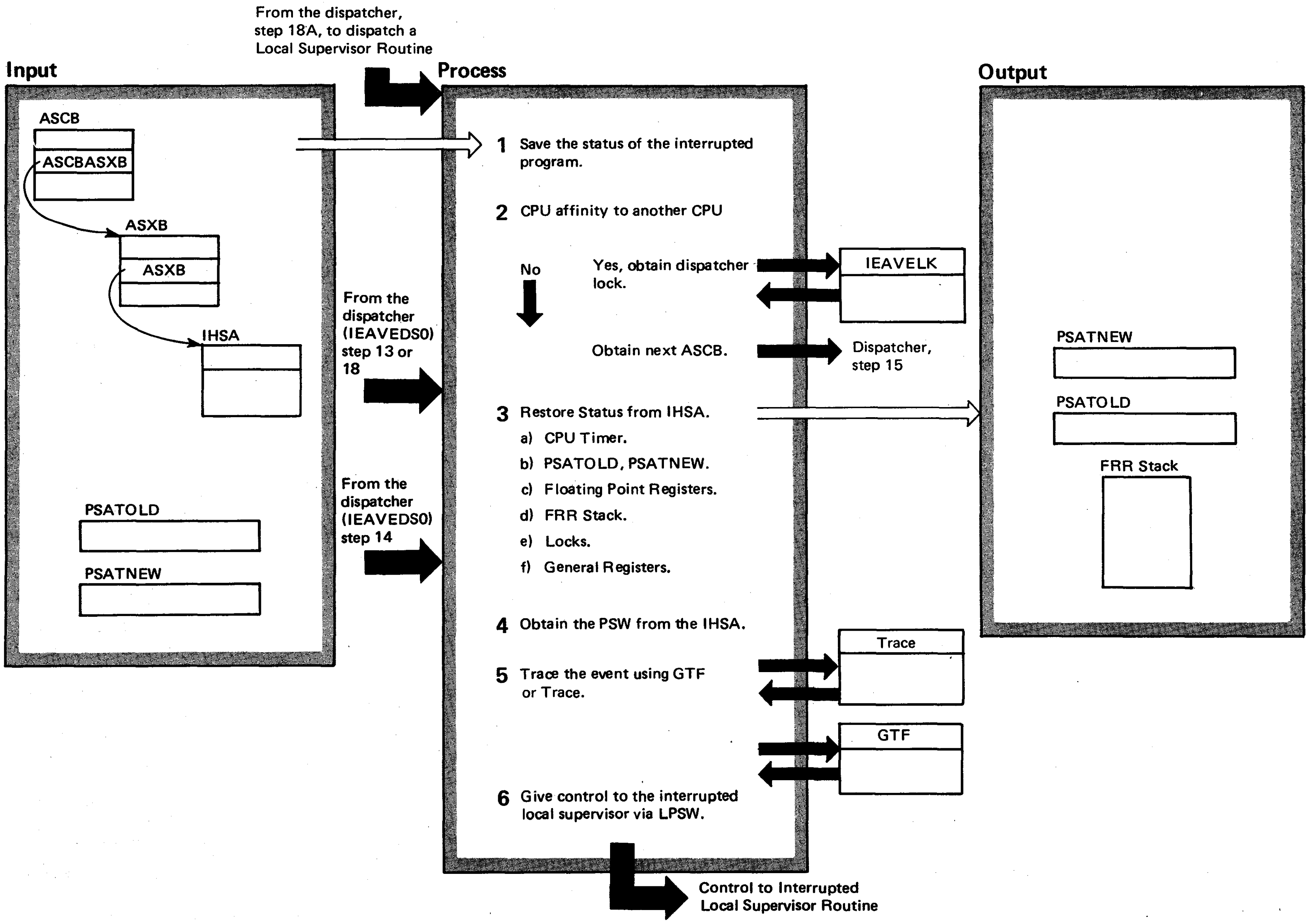
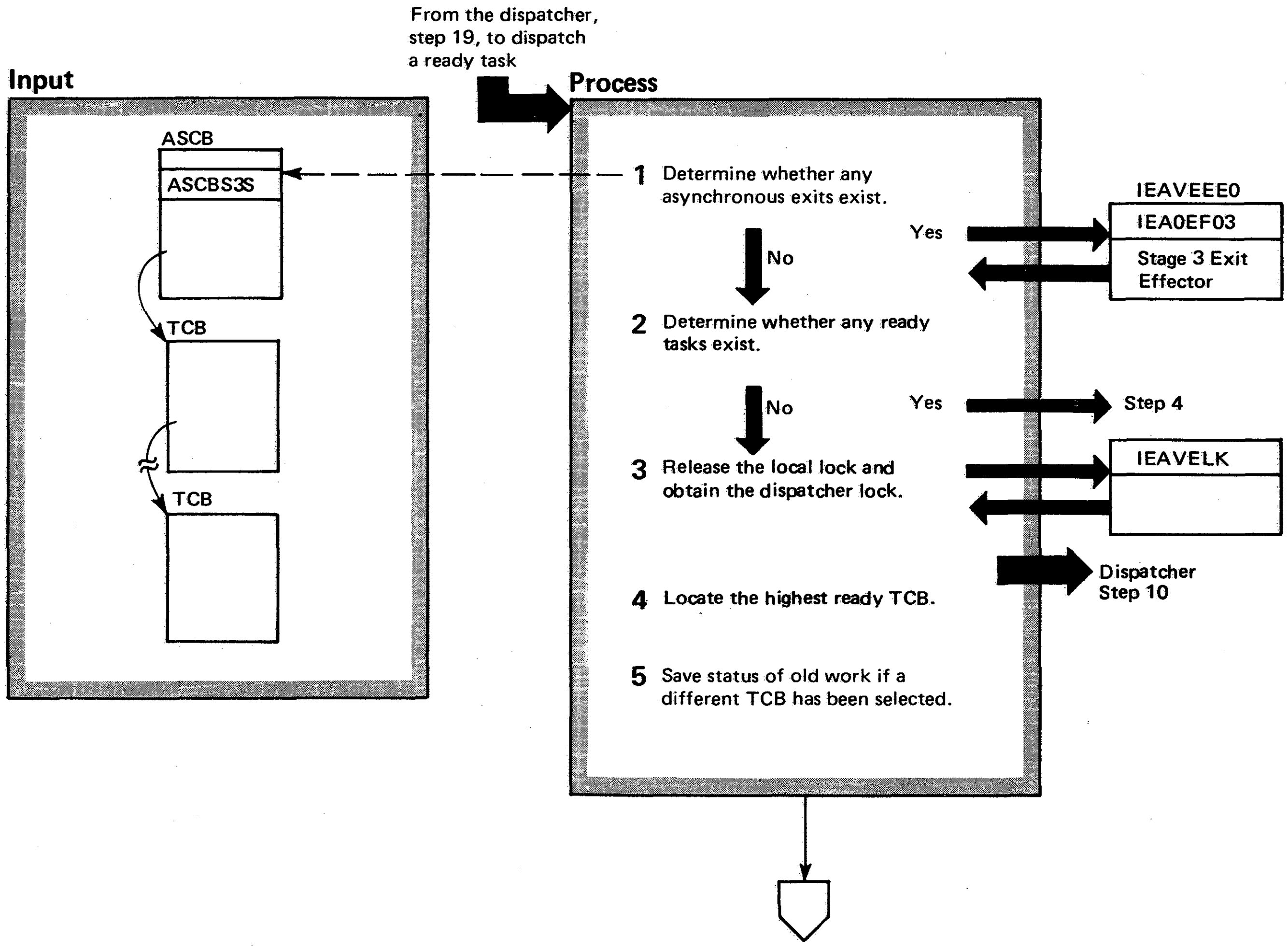

From the dispatcher, step 18A, to dispatch a Local Supervisor Routine
Input
Process
Output
ASCB
ASCBASXB
ASXB
ASXB
IHSA
PSATOLD
PSATNEW
From the dispatcher (IEAVEDS0) step 13 or 18
From the dispatcher (IEAVEDS0) step 14
1 Save the status of the interrupted program.
2 CPU affinity to another CPU
No
Yes, obtain dispatcher lock.
IEAVELK
Obtain next ASCB.
Dispatcher, step 15
3 Restore Status from IHSA.
a) CPU Timer.
b) PSATOLD, PSATNEW.
c) Floating Point Registers.
d) FRR Stack.
e) Locks.
f) General Registers.
4 Obtain the PSW from the IHSA.
5 Trace the event using GTF or Trace.
Trace
GTF
6 Give control to the interrupted local supervisor via LPSW.
Control to Interrupted Local Supervisor Routine
PSATNEW
PSATOLD
FRR Stack

Diagram 19-4. Local Supervisor Dispatcher (IEAVEDS0) (Part 2 of 2)

Extended Description	Module	Label
The local supervisor dispatcher subroutine gives control to interrupted supervisory routines that were performing a local service for a single address space. The interrupted supervisory routine receives control via an LPSW instruction.		
1 The local supervisor dispatcher saves the status of the interrupted routine in the appropriate area.	IEAVEDS0	
2 The local supervisor dispatcher determines whether the interrupted routine had CPU affinity, and if it can process on this CPU. The local supervisor dispatcher stores the interruption ID in the local lock if the routine cannot be processed, and processes the next address space .		
3 The interrupted routine has its status restored from the IHSA (interruption handler save area).		
4 The local supervisor dispatcher obtains the PSW from the IHSA and moves it to the PSA.		
5 Either the trace routine or GTF traces the occurrence of the event.		
6 The local supervisor gives control to the interrupted supervisory routine by issuing an LPSW instruction.		

Diagram 19-5. Task Dispatcher (IEAVEDS0) (Part 1 of 4)

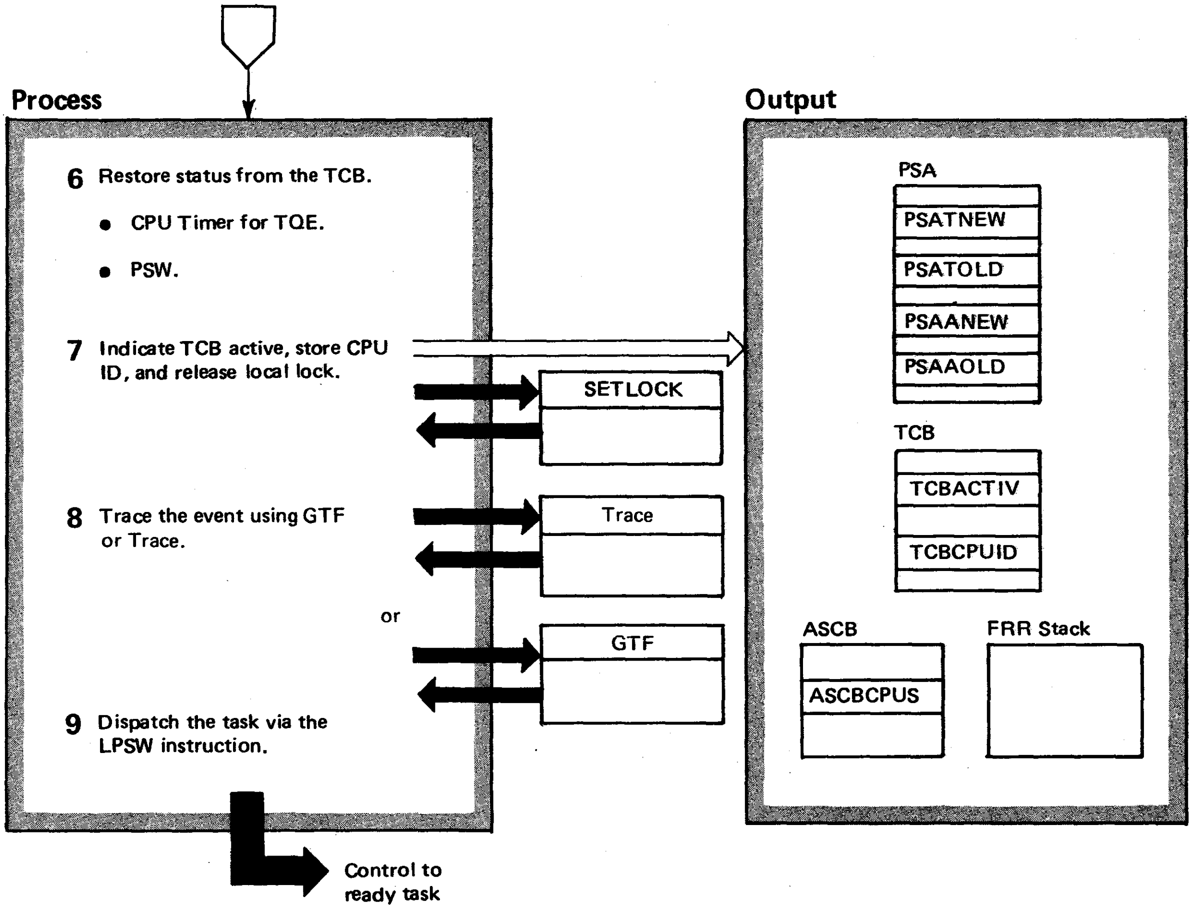


From the dispatcher, step 19, to dispatch a ready task
Input
ASCB
ASCBS3S
TCB
TCB
Process
1 Determine whether any asynchronous exits exist.
Yes
IEAVEEE0
IEA0EF03
Stage 3 Exit Effector
No
2 Determine whether any ready tasks exist.
Yes
Step 4
No
3 Release the local lock and obtain the dispatcher lock.
IEAVELK
Dispatcher Step 10
4 Locate the highest ready TCB.
5 Save status of old work if a different TCB has been selected.

Diagram 19-5. Task Dispatcher (IEAVEDS0) (Part 2 of 4)

Extended Description	Module	Label
The task dispatcher subroutine gives ready tasks control by issuing an LPSW instruction. If no ready tasks can be dispatched, control goes to the dispatcher to dispatch a ready address space.		
1 The TD will give control to the stage 3 exit effector to process any asynchronous exits if the stage 3 switch indicates any requests.	IEAVEDS0	
2 If there are ready tasks on the TCB queue, control goes to step 4. If no ready tasks exist, control goes to step 3.		
3 The task dispatcher will obtain the dispatcher lock and go to the main dispatcher routine to locate a ready address space.		
4 The task dispatcher locates the highest priority, ready task from the TCB queue. The task dispatcher tests if this task is active on another CPU (in multiprocessing systems); the task will not be dispatched if active.		
5 Next, the task dispatcher saves the status of any pre-empted work.		

Diagram 19-5. Task Dispatcher (IEAVEDS0) (Part 3 of 4)

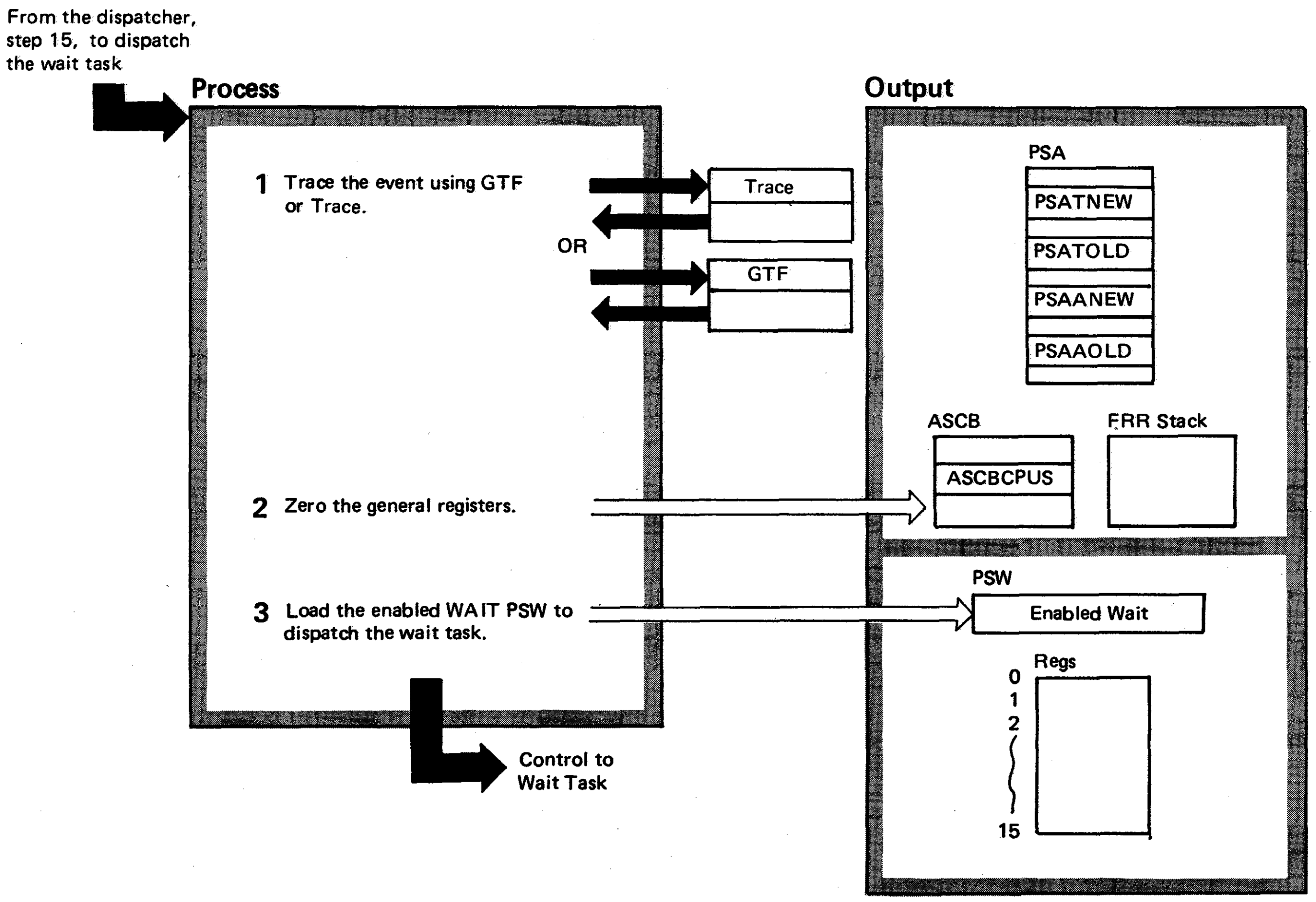


Process
Output
6 Restore status from the TCB.
● CPU Timer for TQE.
● PSW.
7 Indicate TCB active, store CPU ID, and release local lock.
8 Trace the event using GTF or Trace.
9 Dispatch the task via the LPSW instruction.
SETLOCK
Trace
or
GTF
Control to ready task
PSA
PSATNEW
PSATOLD
PSAANEW
PSAAOLD
TCB
TCBACTIV
TCBCPUID
ASCB
ASCBCPUS
FRR Stack

Diagram 19-5. Task Dispatcher (IEAVEDS0) (Part 4 of 4)

Extended Description	Module	Label
6 The task dispatcher prepares the highest priority, ready task by restoring its status from its TCB and RB. The TCTL function enters the dispatcher at this special entry to transfer control to a selected TCB.		IEAVDSTC
7 The task dispatcher releases the local lock.		
8 Either the trace routine or GTF traces the occurrence of the event.		
9 The task dispatcher gives the ready task control by issuing an LPSW instruction for that task.		

Diagram 19-6. Wait Task Dispatcher (IEAVEDS0) (Part 1 of 2)


From the dispatcher, step 15, to dispatch the wait task

Process

1 Trace the event using GTF or Trace.

2 Zero the general registers.

3 Load the enabled WAIT PSW to dispatch the wait task.

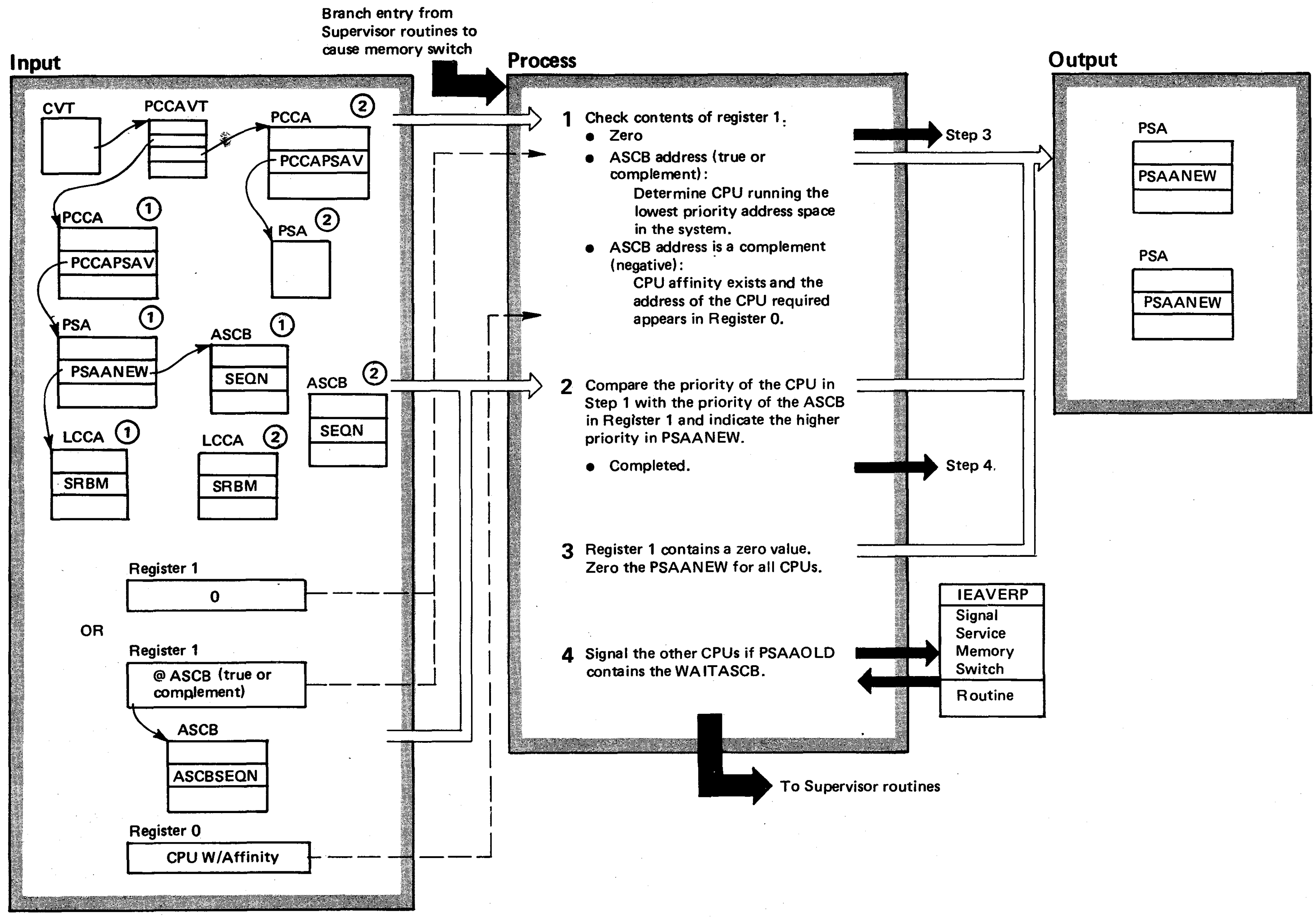
Control to Wait Task

Output

Diagram 19-6. Wait Task Dispatcher (IEAVEDS0) (Part 2 of 2)

Extended Description	Module	Label
The wait task dispatcher gives control to the wait task by using an LPSW instruction, if no ready work can be found in the system.		
1 Either the trace routine or GTF traces the occurrence of the event.	IEAVEDS0	
2 The wait task dispatcher zeros the general registers. The PSATNEW, PSATOLD, PSAANEW, and PSAAOLD fields reflect the wait task and wait ASCB, and the STOR (segment table origin register) contains the wait ASCB's STOR value.		
3 The wait task dispatcher gives the wait task control via an LPSW instruction.		

Diagram 19-7. Memory Switch (IEAVEMS0) (Part 1 of 2)



1 Check contents of register 1.
- Zero
- ASCB address (true or complement):
  Determine CPU running the lowest priority address space in the system.
- ASCB address is a complement (negative):
  CPU affinity exists and the address of the CPU required appears in Register 0.

2 Compare the priority of the CPU in Step 1 with the priority of the ASCB in Register 1 and indicate the higher priority in PSAANEW.
- Completed.

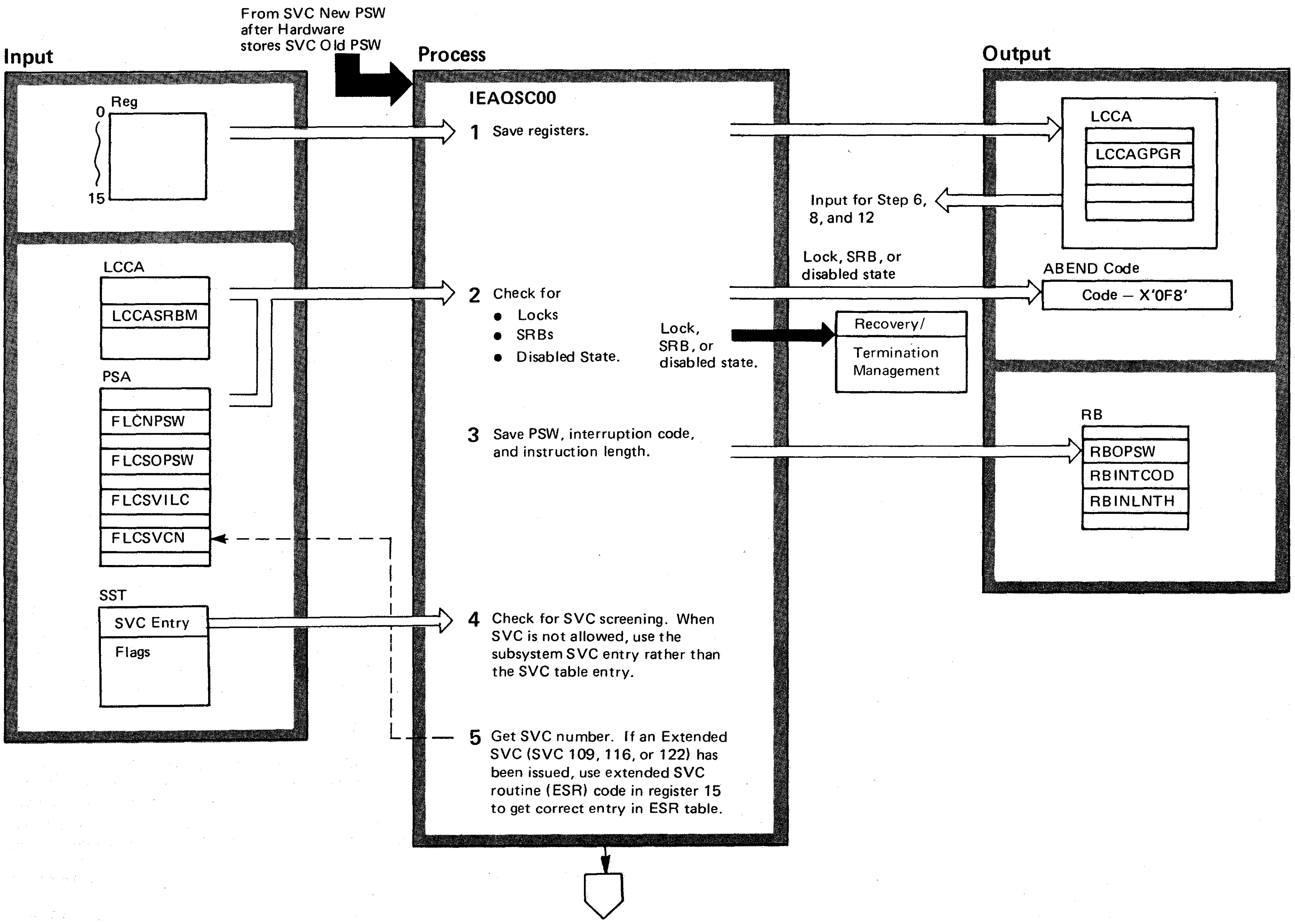
3 Register 1 contains a zero value. Zero the PSAANEW for all CPUs.

4 Signal the other CPUs if PSAAOLD contains the WAITASCB.

Diagram 19-7. Memory Switch (IEAVEMS0) (Part 2 of 2)

Extended Description	Module	Label
Memory switch selects the next address space that can be dispatched on a specific CPU. Memory switch compares the priority of the address space currently selected against the input address space's priority. The address space with the highest priority will be indicated in PSAANEW.		
1 Memory switch checks the contents of register 1. If zero, proceed at step 3. If ASCB address, memory switch finds the CPU running the lowest priority address space in the system. Additionally, if register 1 contains a negative ASCB address (a complemented value), this indicates that the task to execute in the address space requires CPU affinity, and only those CPUs will be considered. When register 1 contains a negative ASCB address, register 0 contains the address of the CPU with affinity.	IEAVEMS0	IEAVEMS0
2 Memory switch stores the input ASCB in the PSAANEW field, if the input ASCB has a higher priority than the ASCB in PSAANEW on the selected CPU. Otherwise, memory switch leaves the original value in PSAANEW, indicating that the input ASCB was of a lower priority.		
3 If register 1 contains a 0, memory switch stores a 0 value in each PSAANEW field. The dispatcher, upon receiving control, will search from the top of the ASCB dispatching queues.		
4 If memory switch was required on another CPU (whose PSAANEW value was changed) and if the previous address space was the WAIT ASCB, then memory switch calls the remote pendable service routine. This causes an external call interruption to be presented to the selected CPU. This interruption causes entry into the dispatcher to find the selected work.		

Diagram 19-8. SVC Interruption Handler (IEAVESVC) (Part 1 of 10)


From SVC New PSW after Hardware stores SVC Old PSW
Input
Reg
0
15
LCCA
LCCASRBM
PSA
FLCNPSW
FLCSOPSW
FLCSVILC
FLCSVCN
SST
SVC Entry
Flags
Process
IEAQSC00
Lock, SRB, or disabled state.
Recovery/ Termination Management
Output
LCCA
LCCAGPGR
Input for Step 6, 8, and 12
Lock, SRB, or disabled state
ABEND Code
Code – X'0F8'
RB
RBOPSW
RBINTCOD
RBINLNTH

1 Save registers.

2 Check for
- Locks
- SRBs
- Disabled State.

3 Save PSW, interruption code, and instruction length.

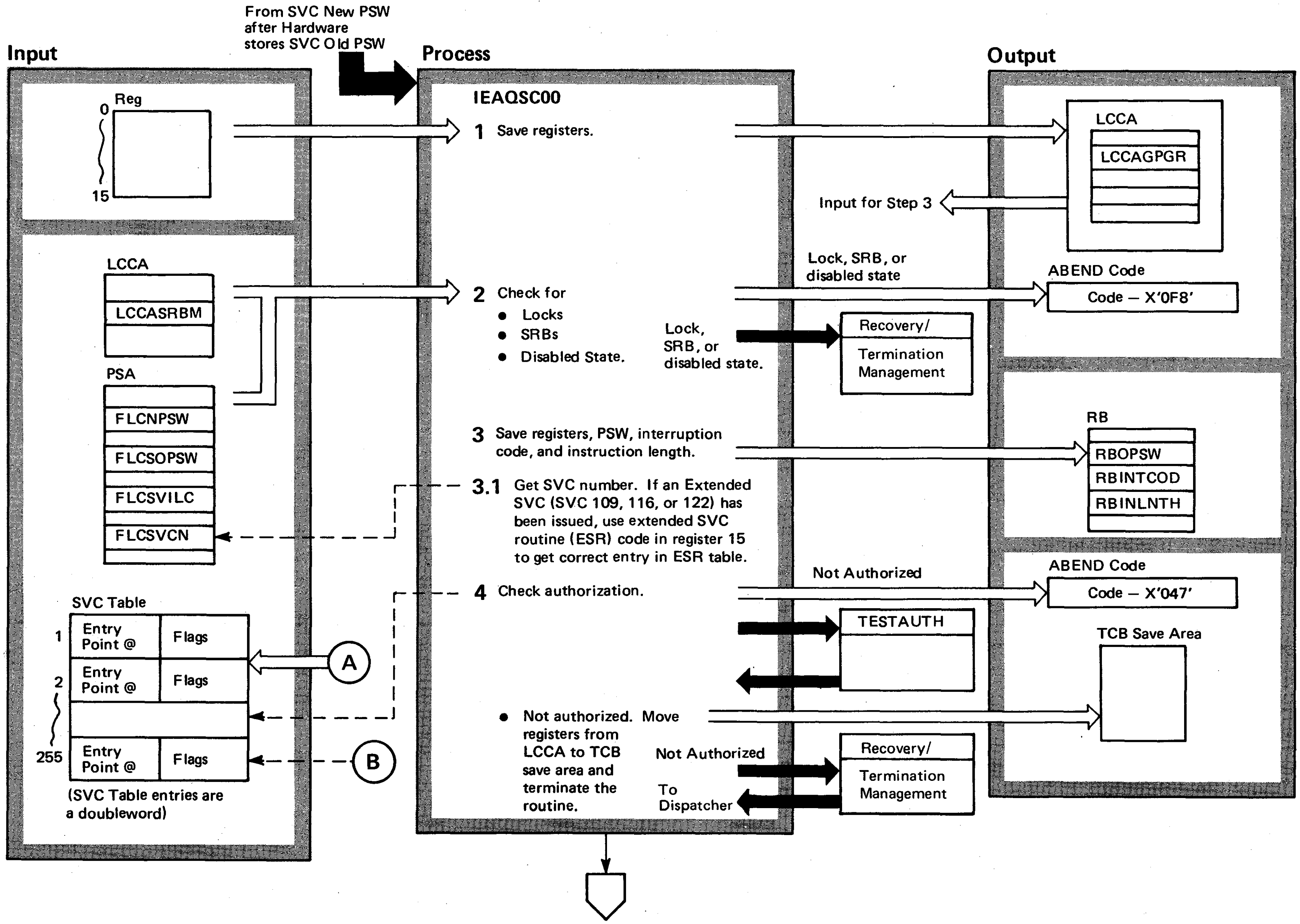
4 Check for SVC screening. When SVC is not allowed, use the subsystem SVC entry rather than the SVC table entry.

5 Get SVC number. If an Extended SVC (SVC 109, 116, or 122) has been issued, use extended SVC routine (ESR) code in register 15 to get correct entry in ESR table.

Diagram 19-7. Memory Switch (IEAVEMS0) (Part 2 of 2)

Extended Description	Module	Label
Memory switch selects the next address space that can be dispatched on a specific CPU. Memory switch compares the priority of the address space currently selected against the input address space's priority. The address space with the highest priority will be indicated in PSAANEW.		
1 Memory switch checks the contents of register 1. If zero, proceed at step 3. If ASCB address, memory switch finds the CPU running the lowest priority address space in the system. Additionally, if register 1 contains a negative ASCB address (a complemented value), this indicates that the task to execute in the address space requires CPU affinity, and only those CPUs will be considered. When register 1 contains a negative ASCB address, register 0 contains the address of the CPU with affinity.	IEAVEMS0	IEAVEMS0
2 Memory switch stores the input ASCB in the PSAANEW field, if the input ASCB has a higher priority than the ASCB in PSAANEW on the selected CPU. Otherwise, memory switch leaves the original value in PSAANEW, indicating that the input ASCB was of a lower priority.		
3 If register 1 contains a 0, memory switch stores a 0 value in each PSAANEW field. The dispatcher, upon receiving control, will search from the top of the ASCB dispatching queues.		
4 If memory switch was required on another CPU (whose PSAANEW value was changed) and if the previous address space was the WAIT ASCB, then memory switch calls the remote pendable service routine. This causes an external call interruption to be presented to the selected CPU. This interruption causes entry into the dispatcher to find the selected work.		

Diagram 19-8. SVC Interruption Handler (IEAVESVC) (Part 1 of 10)



IEAQSC00

1 Save registers.

2 Check for
- Locks
- SRBs
- Disabled State.

3 Save registers, PSW, interruption code, and instruction length.

3.1 Get SVC number. If an Extended SVC (SVC 109, 116, or 122) has been issued, use extended SVC routine (ESR) code in register 15 to get correct entry in ESR table.

4 Check authorization.

- Not authorized. Move registers from LCCA to TCB save area and terminate the routine.

Diagram 19-8. SVC Interruption Handler (IEAVESVC) (Part 2 of 10)

Extended Description	Module	Label
The SVC interruption handler sets up the proper operating environment for a requested SVC (supervisor call) by obtaining any necessary locks and initializing registers. The SVC interruption handler routes control to the appropriate SVC routine after setting up the operating environment.		
1 The SVC interruption handler (IH) immediately saves the requester's registers in the LCCA. The LCCA resides in the SQA and acts as a temporary location to save the requester's status. The SVC IH will later move the status to a permanent location.	IEAVESVC	IEAQSC00
2 Requesters in the disabled state, that are SRBs, or that are locked cannot issue SVCs. Therefore, the SVC IH passes control to R/TM, which begins termination of those requesters. The caller will be abnormally terminated with an ABEND code of X'0F8'.		
3 The SVC IH, after determining that the requester can issue SVCs, saves the interruption code, and saves the instruction length.		
4 If screening is active for this task (TCBSVCS=1), the SVC interrupt handler determines if the caller can issue the SVC as indicated in the Subsystem Screen Table. If the SVC cannot be issued, control is given to the screening SVC rather than the SVC that was requested.		
5 If the Extended SVC Router (ESR) has been invoked (that is, either SVC 109, 116, or 122 has been issued), the caller's ESR code in register 15 is used to obtain the appropriate ESR table entry. The table entry provides the proper entry environment (locking, type, APF authorization, etc.) for the invoked service routine.		

Diagram 19-8. SVC Interruption Handler (IEAVESVC) (Part 3 of 10)

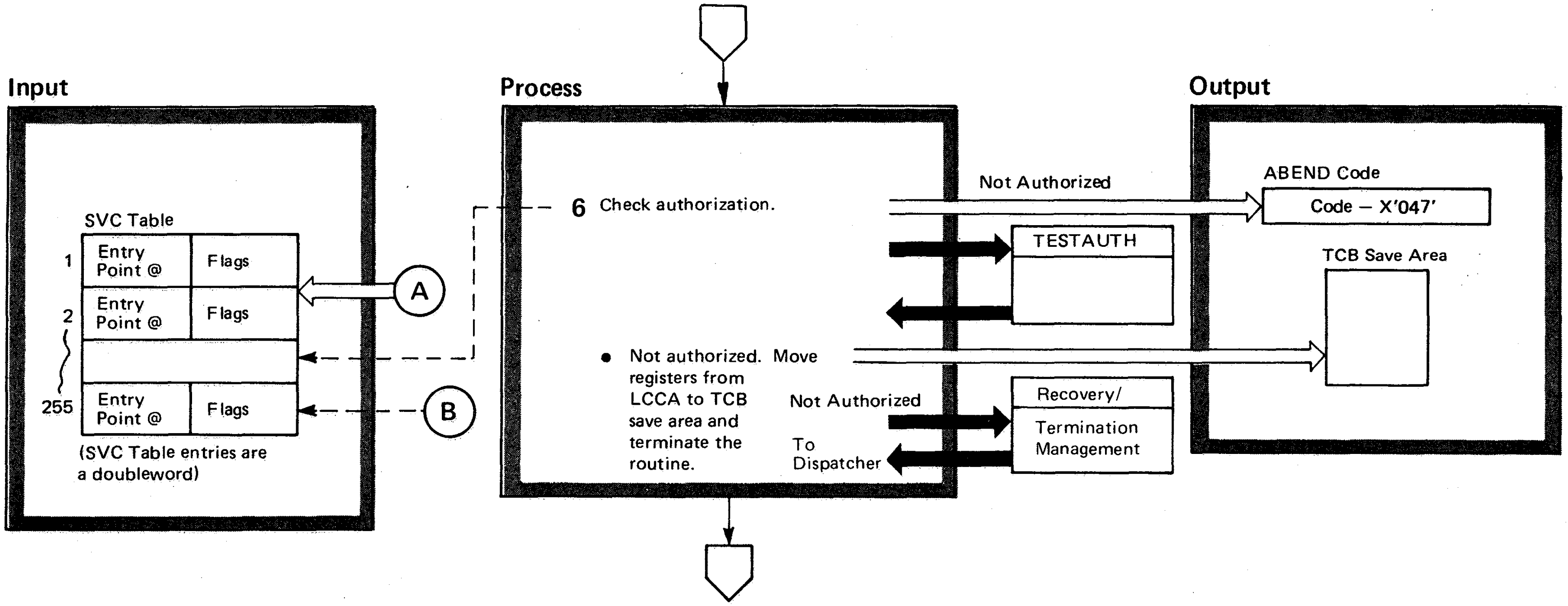
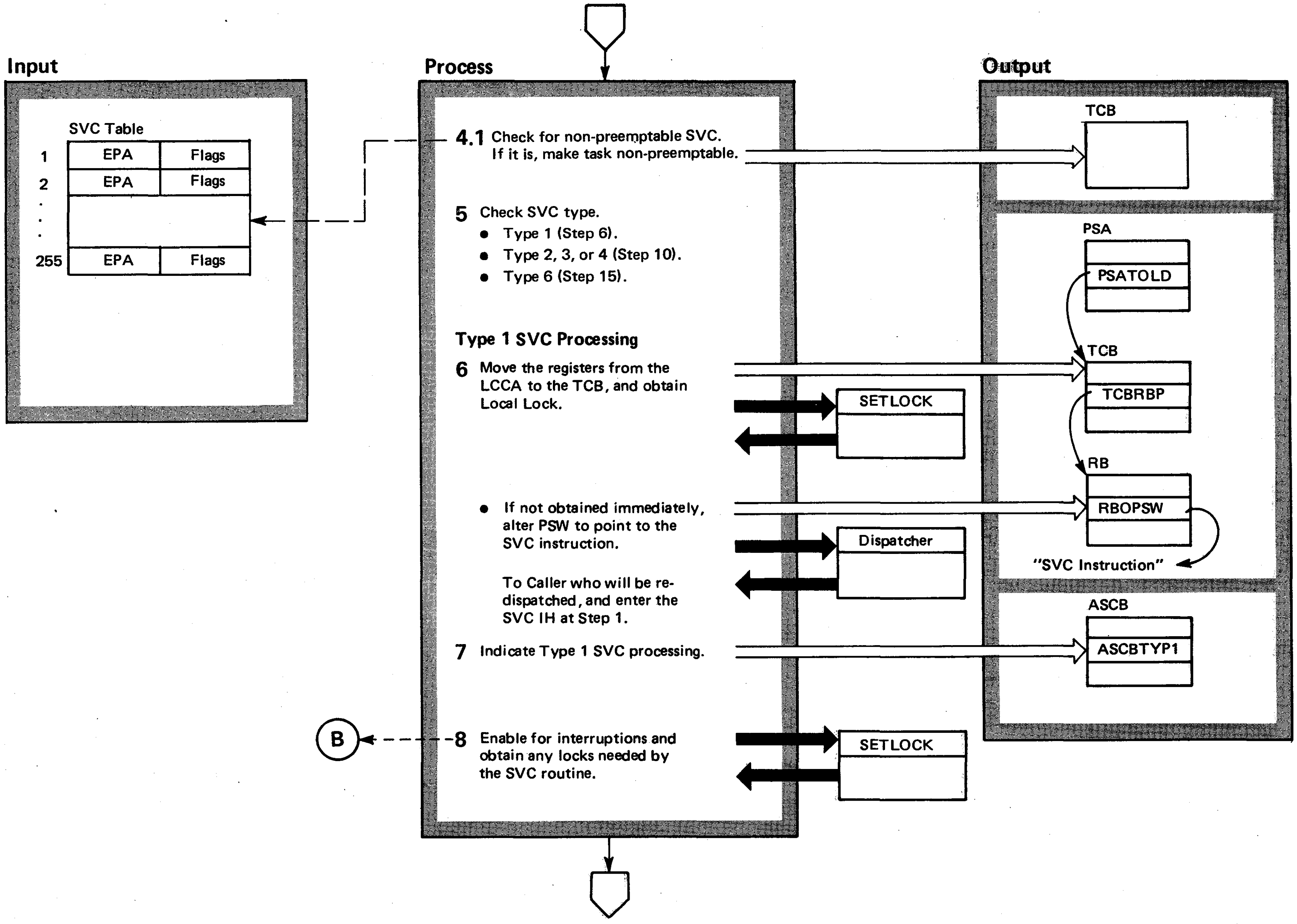


Diagram 19-8. SVC Interruption Handler (IEAVESVC) (Part 2 of 10)

Extended Description	Module	Label
The SVC interruption handler sets up the proper operating environment for a requested SVC (supervisor call) by obtaining any necessary locks and initializing registers. The SVC interruption handler routes control to the appropriate SVC routine after setting up the operating environment.		
1 The SVC interruption handler (IH) immediately saves the requester's registers in the LCCA. The LCCA resides in the SQA and acts as a temporary location to save the requester's status. The SVC IH will later move the status to a permanent location.	IEAVESVC	IEAQSC00
2 Requesters in the disabled state, that are SRBs, or that are locked cannot issue SVCs. Therefore, the SVC IH passes control to R/TM, which begins termination of those requesters. The caller will be abnormally terminated with an ABEND code of X'0F8'.		
3 The SVC IH, after determining that the requester can issue SVCs, saves the interruption code, and saves the instruction length.		
3.1 If the Extended SVC Router (ESR) has been invoked (that is, either SVC 109, 116, or 122 has been issued), the caller's ESR code in register 15 is used to obtain the appropriate ESR table entry. The table entry provides the proper entry environment (locking, type, APF authorization, etc.) for the invoked service routine.		
4 Since some SVCs can only be issued by users with APF authorization, the SVC IH determines whether the SVC requires authorization. The SVC IH refers to the SVC table, using the value in the FLCSVCN field of the PSA as an index value into the SVC table. If TESTAUTH returns a non-zero return code, this indicates that the user does not have authorization. Then, the SVC IH gives control to R/TM to abnormally terminate the user with an ABEND code of X'047'.		

Diagram 19-8. SVC Interruption Handler (IEAVESVC) (Part 3 of 10)



4.1 Check for non-preemptable SVC. If it is, make task non-preemptable.

5 Check SVC type.
- Type 1 (Step 6).
- Type 2, 3, or 4 (Step 10).
- Type 6 (Step 15).

Type 1 SVC Processing

6 Move the registers from the LCCA to the TCB, and obtain Local Lock.

- If not obtained immediately, alter PSW to point to the SVC instruction.

  To Caller who will be re-dispatched, and enter the SVC IH at Step 1.

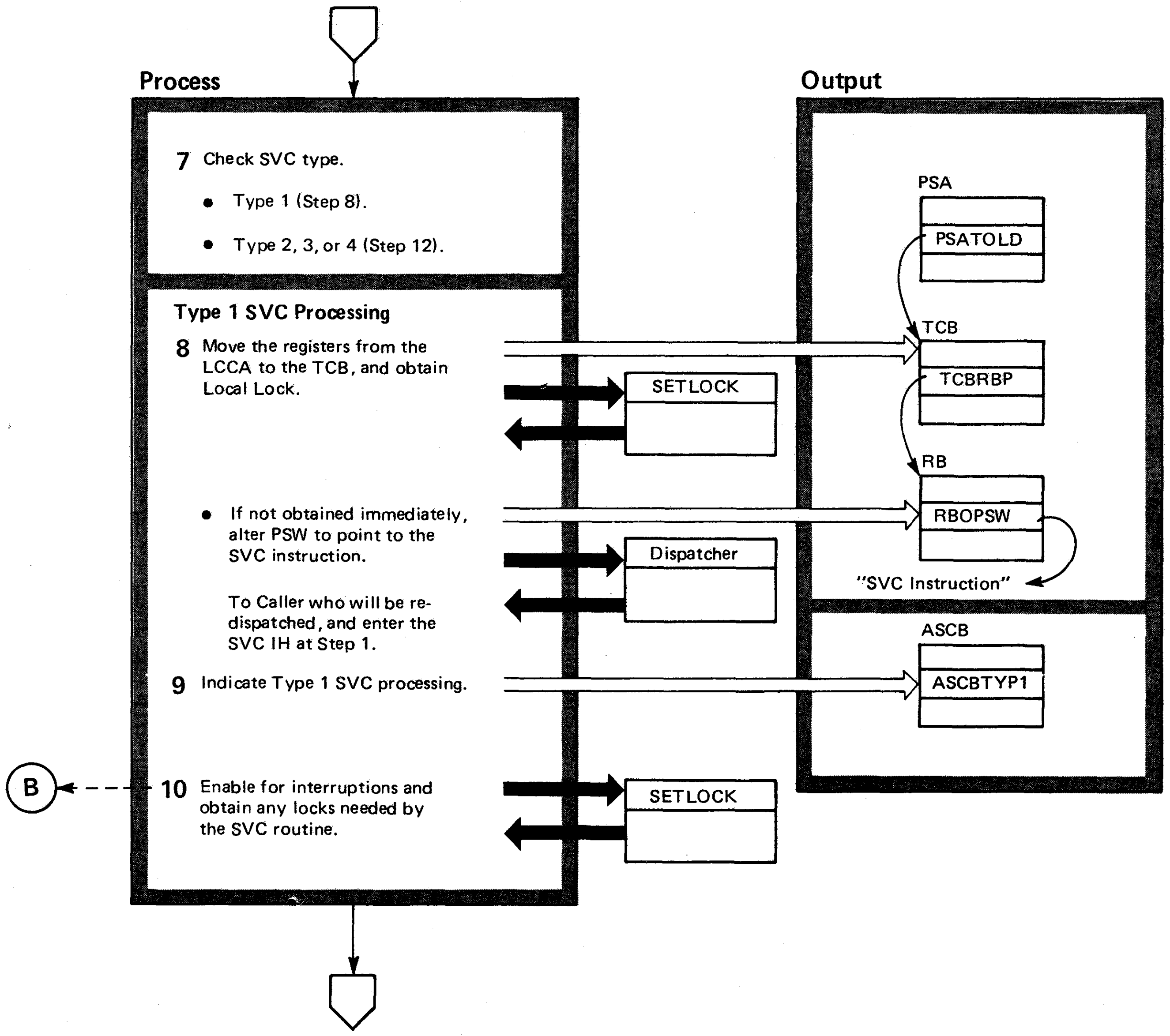
7 Indicate Type 1 SVC processing.

8 Enable for interruptions and obtain any locks needed by the SVC routine.

Diagram 19-8. SVC Interruption Handler (IEAVESVC) (Part 4 of 10)

Extended Description	Module	Label
6 Since some SVCs can only be issued by users with APF authorization, the SVC IH determines whether the SVC requires authorization. The SVC IH refers to the SVC table, using the value in the FLCSVCN field of the PSA as an index value into the SVC table. If TESTAUTH returns a non-zero return code, this indicates that the user does not have authorization. Then, the SVC IH gives control to R/TM to abnormally terminate the user with an ABEND code of X'047'.		

Diagram 19-8. SVC Interruption Handler (IEAVESVC) (Page 5 of 10)



Process
Output
7 Check SVC type.
• Type 1 (Step 8).
• Type 2, 3, or 4 (Step 12).
Type 1 SVC Processing
8 Move the registers from the LCCA to the TCB, and obtain Local Lock.
SETLOCK
• If not obtained immediately, alter PSW to point to the SVC instruction.
Dispatcher
To Caller who will be re-dispatched, and enter the SVC IH at Step 1.
9 Indicate Type 1 SVC processing.
B
10 Enable for interruptions and obtain any locks needed by the SVC routine.
SETLOCK
PSA
PSATOLD
TCB
TCBRBP
RB
RBOPSW
"SVC Instruction"
ASCB
ASCBTYP1

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

4.1 If the SVC is non-preemptable (SVCNP=1), then the task is made non-preemptable (TCBNONPR=1).

5 Based on the SVC type, this step branches to the appropriate processing routine. Note that steps 8 and 9 show processing common to SVC types 1, 2, 3, and 4.

SVC type	*Steps*
1	6-9
2, 3, 4	10-12
6	15-16

Type 1 SVC Processing

6 To process Type 1 SVCs, the SVC IH must move the caller's registers in the TCB and obtain the local lock. A request is made conditionally, since the SVC IH cannot be suspended (see the SETLOCK routine). Operation continues, at step 7, if the local lock is obtained. Otherwise, the SVC IH changes the RBOPSW in the requester's RB to indicate that it will be redispatched to reissue the SVC instruction, and gives the dispatcher control. The requester will eventually be redispatched.

7 The SVC IH indicates Type 1 processing in the ASCBFLG1 field, bit ASCBTYP1.

8 Interruptions can now be processed, with the status of any interrupted programs being saved in the IHSA (interruption handler save area). The operating environment for the requested SVC routine can now be set by the SVC IH. As the first step, the SVC IH obtains any locks that the SVC routine needs, as indicated by the SVC table.

Diagram 19-8. SVC Interruption Handler (IEAVESVC) (Part 5 of 10)

Process

Type 2, 3, or 4 Processing

10 Acquire an SVRB. The SVRB pool is managed by the SVC FLIH. If an SVRB pool element is available, perform step b. If not, do step a and continue with b.

a) Initialize SVRB pool.

- Get pool.
- Build SVRB pool.
- Chain SVRB pool.

b) Normal Request.

- Get an SVRB element.
- Initialize SVRB.
- Chain SVRB to TCB/RB queue.
- Move the Registers from the LCCA save area to the SVRB.

Output

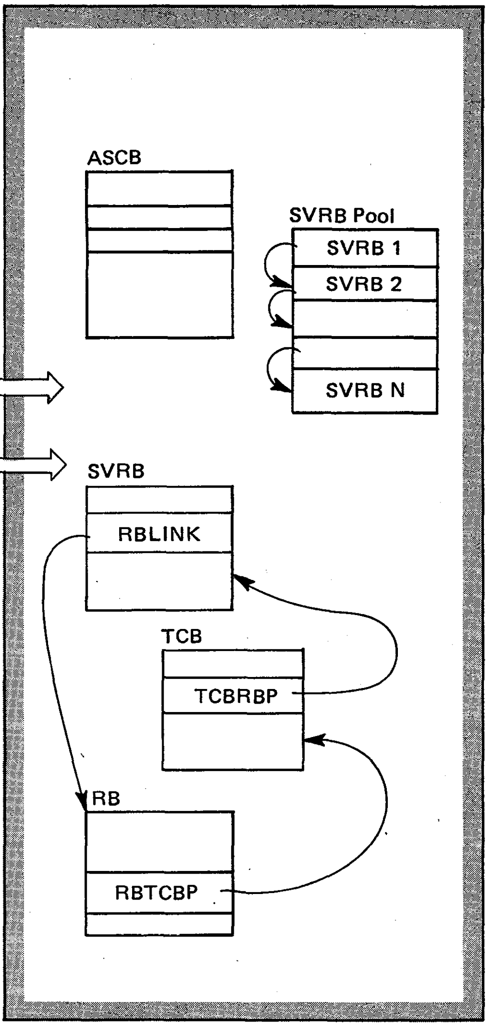


Diagram 19-8. SVC Interruption Handler (IEAVESVC) (Part 6 of 10)

Extended Description	Module	Label
7 Steps 8-11 show Type 1 SVC processing, while steps 12-14 show Type 2, 3, and 4 SVC processing. Note that steps 10 and 11 show processing common to all SVC routines.		
8 To process Type 1 SVCs, the SVC IH must move the caller's registers in the TCB and obtain the local lock. A request is made conditionally, since the SVC IH cannot be suspended (see the SETLOCK routine). Operation continues, at step 9, if the local lock is obtained. Otherwise, the SVC IH changes the RBOPSW in the requester's RB to indicate that it will be redispatched to reissue the SVC instruction, and gives the dispatcher control. The requester will eventually be redispatched.		
9 The SVC IH indicates Type 1 processing in the ASCBFLG1 field, bit ASCBTYP1.		
10 Interruptions can now be processed, with the status of any interrupted programs being saved in the IHSA (interruption handler save area). The operating environment for the requested SVC routine can now be set by the SVC IH. As the first step, the SVC IH obtains any locks that the SVC routine needs, as indicated by the SVC table.		

Diagram 19-8. SVC Interruption Handler (IEAVESVC) (Part 7 of 10)

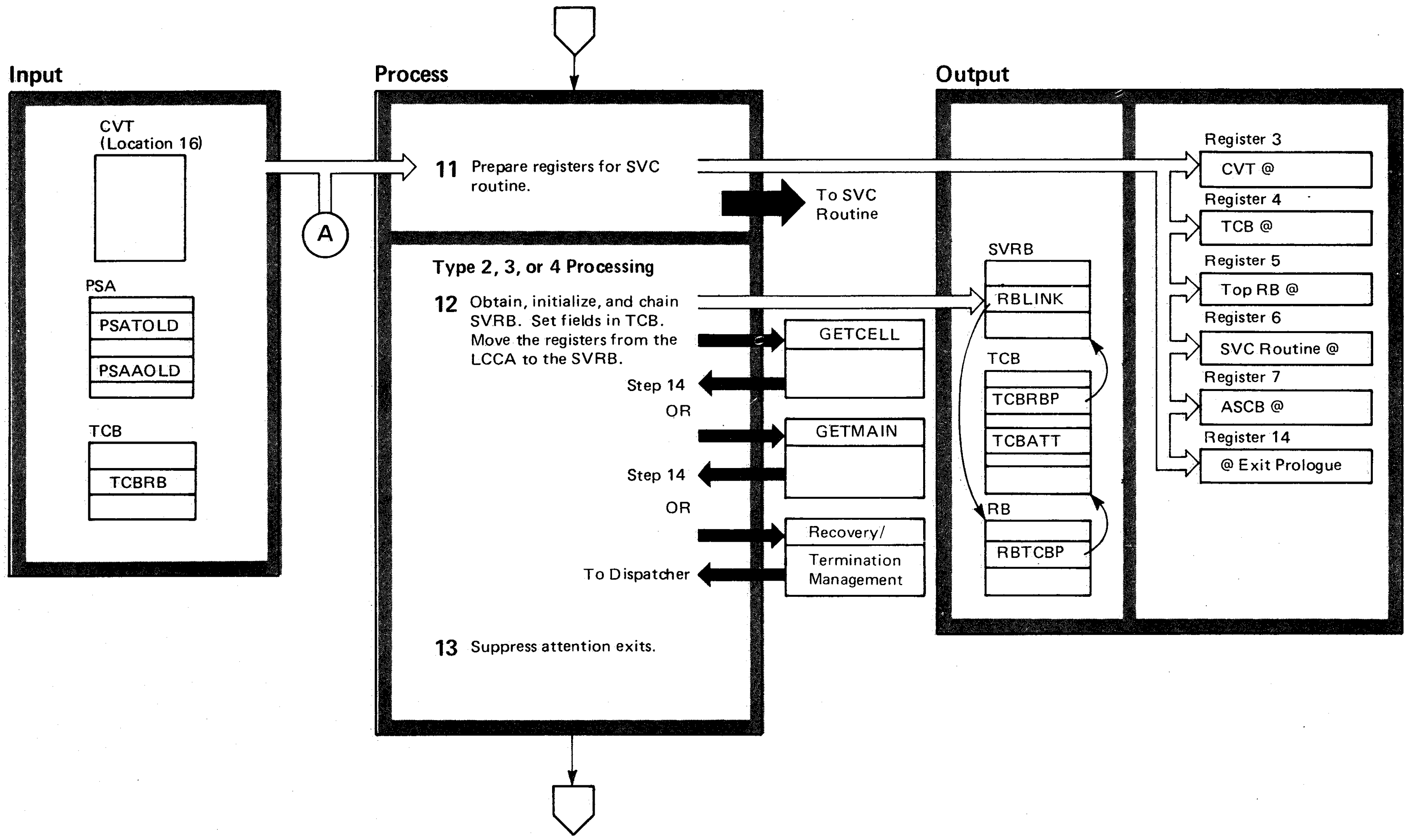


Diagram 19-8. SVC Interruption Handler (IEAVESVC) (Part 6 of 10)

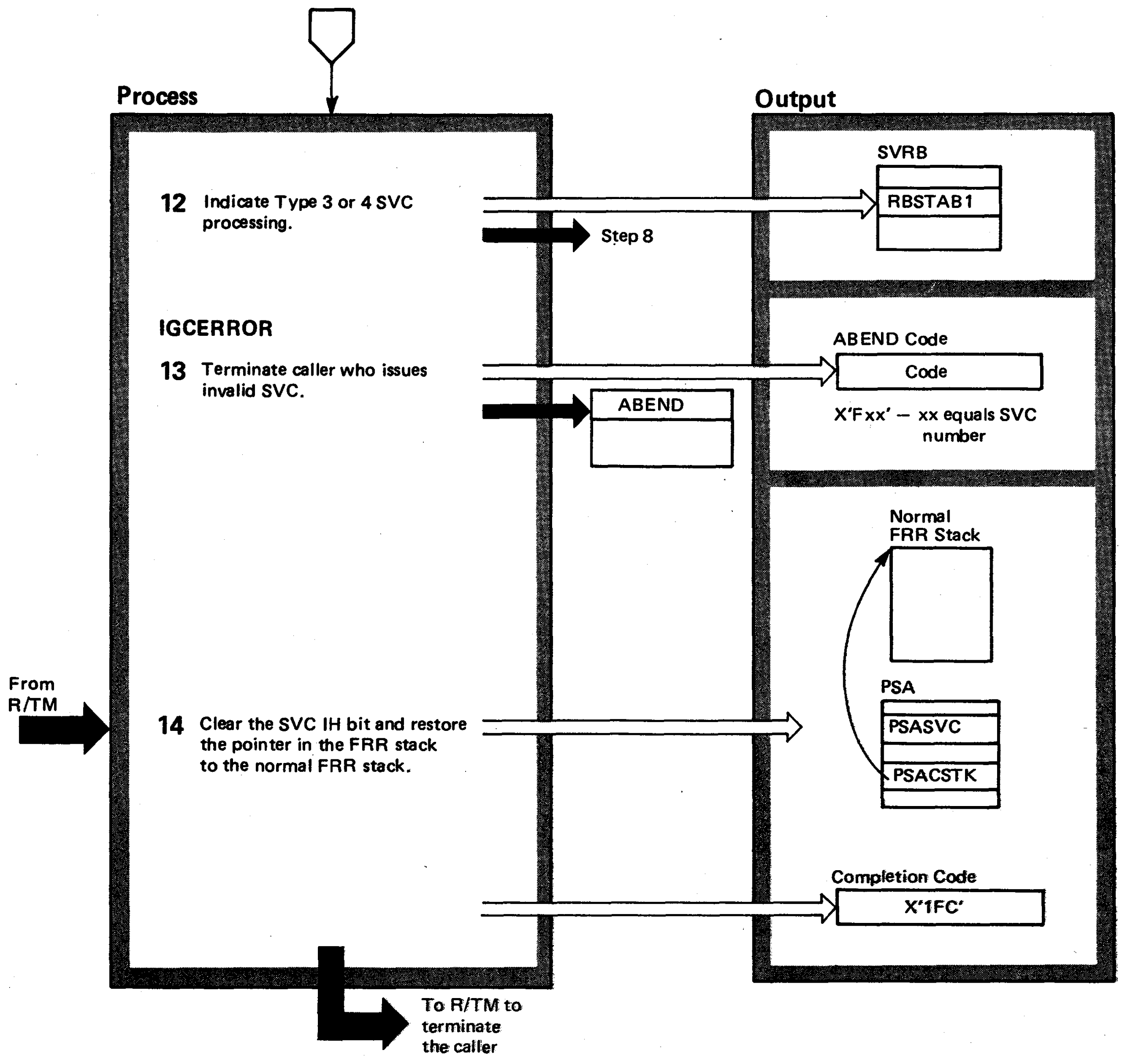
Extended Description **Module** **Label**

Types 2, 3, and 4 SVC Processing

10 Type 2, 3, or 4 SVCs need SVRBs (supervisor request blocks) built. The SVC interruption handler obtains the storage for an SVRB, moves the registers from LCCA, and initializes the SVRB. The SVC IH (interrupt handler) obtains the storage for an SVRB in the following manner:

- Attempts to directly obtain an SVRB pool chained off the ASCB.
- If the ASCBSVRB pointer is zero (no SVRBs available), the SVC IH determines whether an ABEND or ABTERM is in process. If so, the SVC IH uses GETMAIN to acquire a single SVRB. If this GETMAIN fails, the address space will be terminated. If no ABEND or ABTERM is in progress, the SVC IH will issue a GETMAIN to expand the pool. If the pool cannot be expanded, R/TM will be called with a X'0F9' ABEND code. If the pool is obtained, it will be initialized and chained to the ASCBSVRB pool queue.

Diagram 19-8. SVC Interruption Handler (IEAVESVC) (Part 7 of 10)


Process
Output
12 Indicate Type 3 or 4 SVC processing.
Step 8
SVRB
RBSTAB1
IGCERROR
13 Terminate caller who issues invalid SVC.
ABEND
ABEND Code
Code
X'Fxx' – xx equals SVC number
Normal FRR Stack
PSA
PSASVC
PSACSTK
From R/TM
14 Clear the SVC IH bit and restore the pointer in the FRR stack to the normal FRR stack.
Completion Code
X'1FC'
To R/TM to terminate the caller

Extended Description	Module	Label
11 As the last step, the SVC IH sets the proper values in input registers used by the SVC routine, and gives the SVC routine control using the address in the SVC table. Registers 0, 1, 13 and 15 contain the same value as when the requester issued the SVC.		
12 Type 2, 3, or 4 SVCs need SVRBs (supervisor request blocks) built. The SVC IH obtains the storage for an SVRB, moves the registers from the LCCA and initializes it. The SVC IH obtains the storage for the SVRB in the manner: ● Attempts to use the GETCELL routine to obtain the necessary storage for an SVRB. ● If the GETCELL fails, the SVC IH determines whether an ABEND or ABTERM is in process. If so, the SVC IH uses the GETMAIN routine to obtain the storage for a single SVRB. The address space will be terminated if this GETMAIN fails. If no ABEND is in process, the SVC IH will try to expand the SVRB cell pool via a GETMAIN. If the SVRB cell pool cannot be expanded, the SVC IH gives control to R/TM to abnormally terminate the SVC requester with an ABEND code of X'0F9'. The SVC IH will obtain a single cell for the SVRB from the expanded SVRB cell pool if the attempt succeeded. ● After obtaining an SVRB, the SVC IH indicates whether the SVRB was obtained by GETMAIN or GETCELL. When the SVC routine completes, EXIT or Exit Prologue frees the storage with either FREEMAIN or FREECELL.		
13 The SVC IH suppresses attention exits from processing. The TCBATT bits indicate this.		

Diagram 19-8. SVC Interruption Handler (IEAVESVC) (Part 9 of 10)

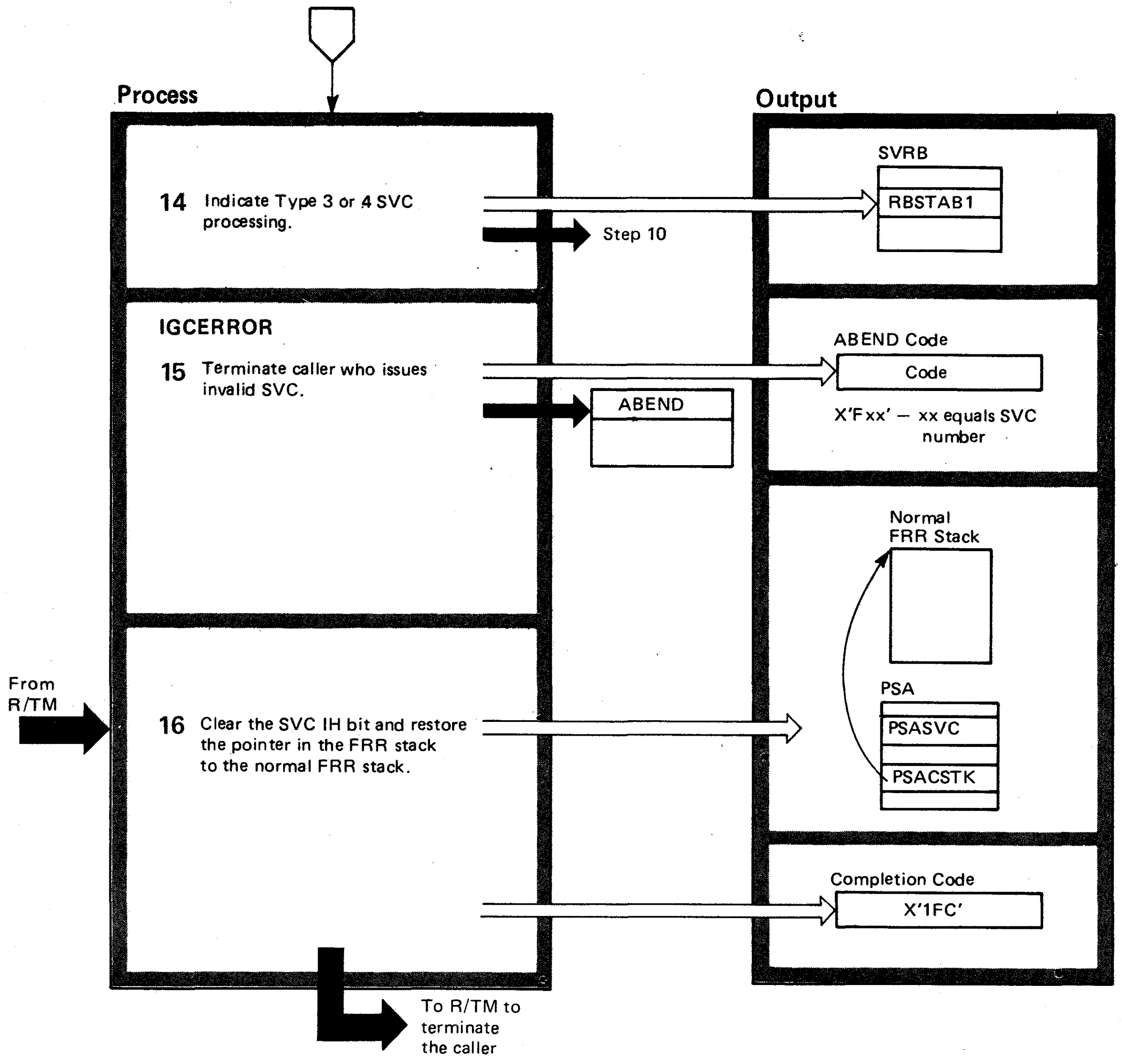
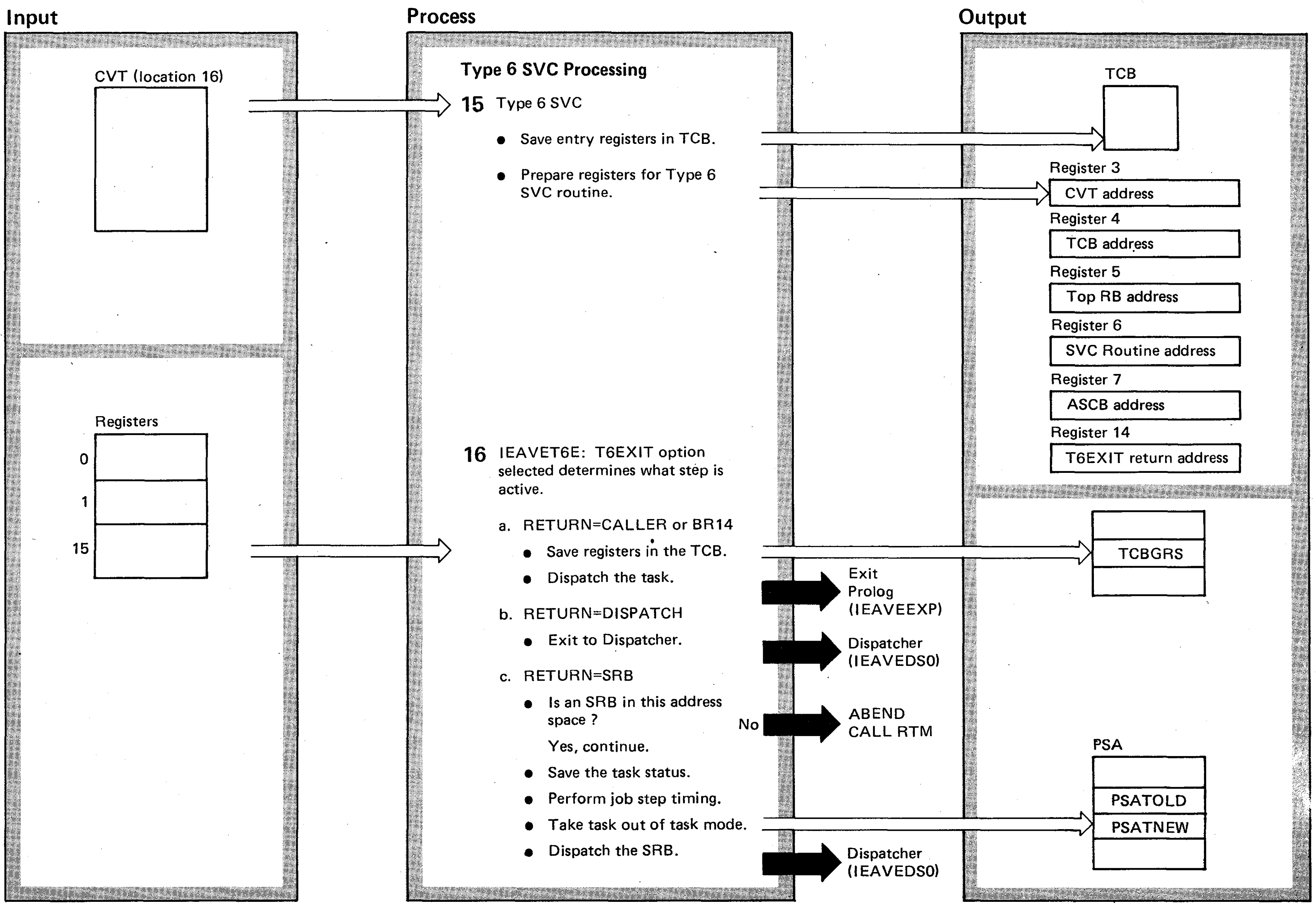

Process
Output
14 Indicate Type 3 or 4 SVC processing.
Step 10
SVRB
RBSTAB1
IGCERROR
15 Terminate caller who issues invalid SVC.
ABEND
ABEND Code
Code
X'Fxx' — xx equals SVC number
From R/TM
16 Clear the SVC IH bit and restore the pointer in the FRR stack to the normal FRR stack.
Normal FRR Stack
PSA
PSASVC
PSACSTK
Completion Code
X'1FC'
To R/TM to terminate the caller

Diagram 19-8. SVC Interruption Handler (IEAVESVC) (Part 8 of 8)

Extended Description	Module	Label
12 The SVC IH indicates in the requester's RB that the SVC is either Type 3 or Type 4.		
13 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 equals the number of the invalid SVC.		
14 The SVC IH FRR (functional recovery routine) clears the SVC indicator in the PSA, sets the FRR stack pointer to the normal stack, and terminates the caller with a X'1FC' completion code.	IEAVESVR	

Diagram 19-8. SVC Interruption Handler (IEAVESVC) (Part 9 of 10)



Type 6 SVC Processing

15 Type 6 SVC

- Save entry registers in TCB.
- Prepare registers for Type 6 SVC routine.

16 IEAVET6E: T6EXIT option selected determines what step is active.

a. RETURN=CALLER or BR14
- Save registers in the TCB.
- Dispatch the task.

b. RETURN=DISPATCH
- Exit to Dispatcher.

c. RETURN=SRB
- Is an SRB in this address space ?

  Yes, continue.
- Save the task status.
- Perform job step timing.
- Take task out of task mode.
- Dispatch the SRB.

Diagram 19-8. SVC Interruption Handler (IEAVESVC) (Part 10 of 10)

Extended Description	Module	Label
14 The SVC IH indicates in the requester's RB that the SVC is either Type 3 or Type 4.		
15 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 equals the number of the invalid SVC.		
16 The SVC IH FRR (functional recovery routine) clears the SVC indicator in the PSA, sets the FRR stack pointer to the normal stack, and terminates the caller with a X'1FC' completion code.	IEAVESVR	

Diagram 19-9. I/O Interruption Handler (IEAVEIO) (Part 1 of 4)

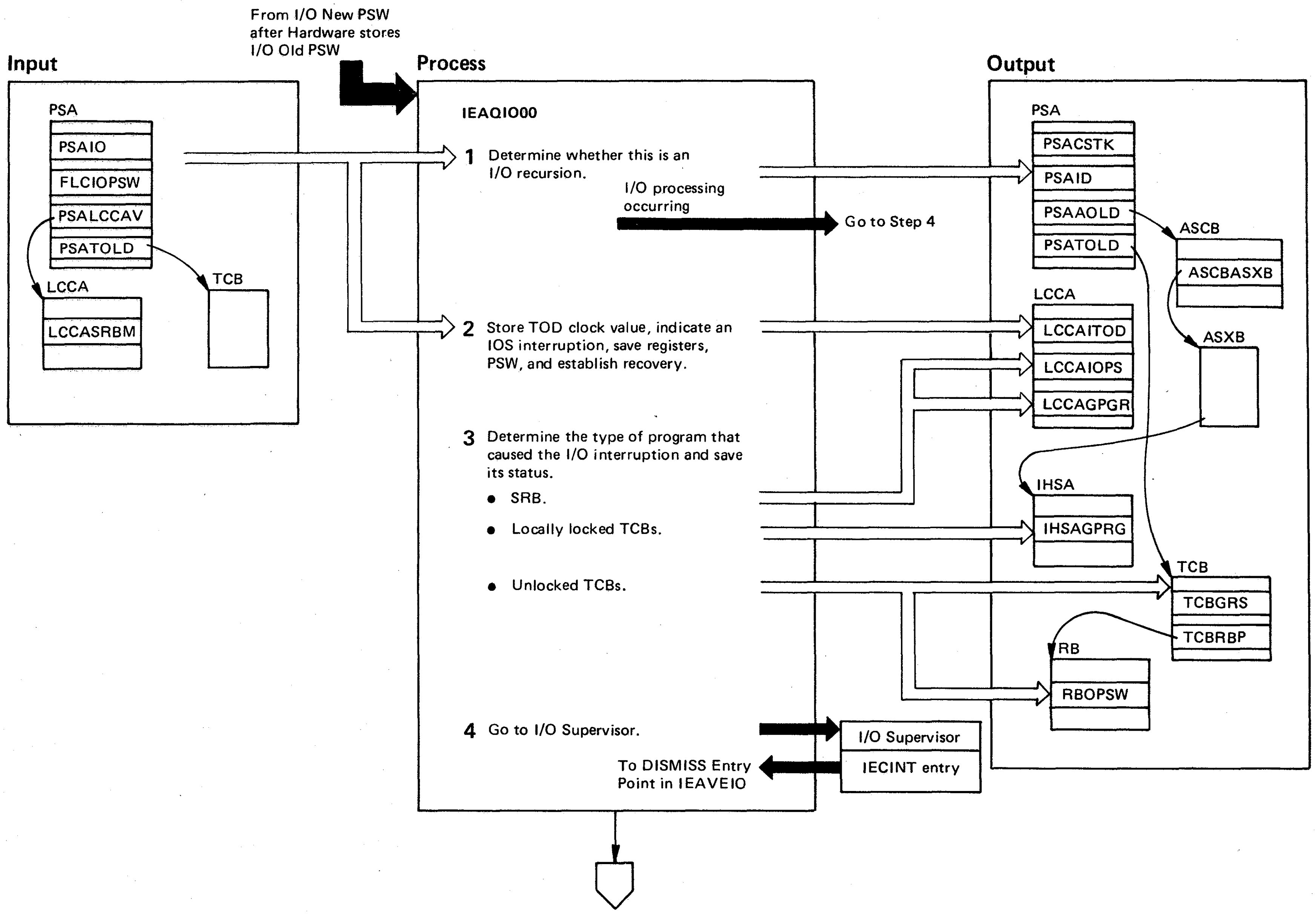
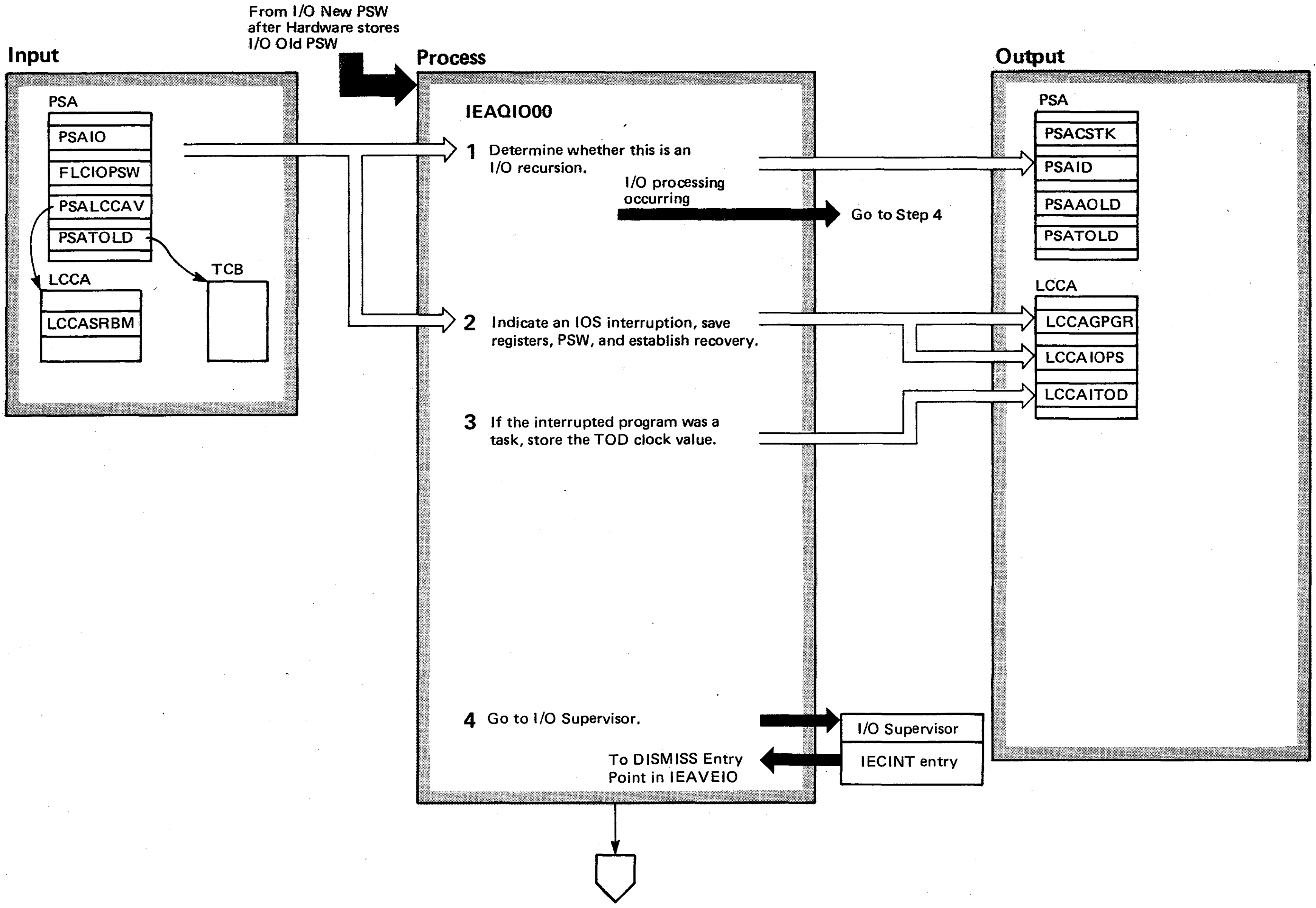

From I/O New PSW after Hardware stores I/O Old PSW
Input
PSA
PSAIO
FLCIOPSW
PSALCCAV
PSATOLD
LCCA
LCCASRBM
TCB
Process
IEAQIO00
1 Determine whether this is an I/O recursion.
I/O processing occurring
Go to Step 4
2 Store TOD clock value, indicate an IOS interruption, save registers, PSW, and establish recovery.
3 Determine the type of program that caused the I/O interruption and save its status.
• SRB.
• Locally locked TCBs.
• Unlocked TCBs.
4 Go to I/O Supervisor.
I/O Supervisor
IECINT entry
To DISMISS Entry Point in IEAVEIO
Output
PSA
PSACSTK
PSAID
PSAAOLD
PSATOLD
ASCB
ASCBASXB
ASXB
LCCA
LCCAITOD
LCCAIOPS
LCCAGPGR
IHSA
IHSAGPRG
TCB
TCBGRS
TCBRBP
RB
RBOPSW

Diagram 19-8. SVC Interruption Handler (IEAVESVC) (Part 10 of 10)

Extended Description	Module	Label
Type 6 SVC Processing		
15 The Type 6 SVC processor saves the registers stored in LCCA in the TCB and then sets up input registers for the Type 6 SVC routine.		TYPE6SVC
Type 6 Exit Processing		
16 When a Type 6 SVC exists, there are three options:		IEAVET6E
a. RETURN=CALLER or BR14 results in registers 0, 1, and 15 being saved in the TCB and an exit made to exit prolog to directly re-dispatch the task.		
b. RETURN=DISPATCH results in a direct entry into the dispatcher.		
c. RETURN=SRB results in a check of the SRB being scheduled for this address space. If there is no SRB, an ABEND is issued. If there is a SRB, the SVC IH saves the task status (floating point registers and timing data), calls the dispatcher job step timing routine (DSJSTCSR), decrements the ASCBTCBS count, sets PSATOLD to zero to take the task out of task mode, and calls the global SRB dispatcher routine to directly dispatch the specified SRB.		

Diagram 19-9. I/O Interruption Handler (IEAVEIO) (Part 1 of 4)



From I/O New PSW after Hardware stores I/O Old PSW
Input
PSA
PSAIO
FLCIOPSW
PSALCCAV
PSATOLD
LCCA
LCCASRBM
TCB
Process
IEAQIO00
1 Determine whether this is an I/O recursion.
I/O processing occurring
Go to Step 4
2 Indicate an IOS interruption, save registers, PSW, and establish recovery.
3 If the interrupted program was a task, store the TOD clock value.
4 Go to I/O Supervisor.
To DISMISS Entry Point in IEAVEIO
I/O Supervisor
IECINT entry
Output
PSA
PSACSTK
PSAID
PSAAOLD
PSATOLD
LCCA
LCCAGPGR
LCCAIOPS
LCCAITOD

Diagram 19-9. I/O Interruption Handler (IEAVEIO) (Part 2 of 4)

Extended Description	Module	Label
The I/O interruption handler (IH) saves the requester's status prior to giving the I/O supervisor (IOS) control. Furthermore, the I/O IH routes recursive I/O interruptions directly to the I/O supervisor.		
1 The I/O IH looks at the recursive bit (PSAIO) in the PSA to check for a recursive entry. IOS immediately receives control for recursive conditions. (I/O recursions will occur only if IOS enables for I/O interruptions.) Otherwise, normal processing occurs at step 2.	IEAVEIO	
2 The I/O IH sets the recursion bit in the PSA, PSAIO, to indicate that it is currently processing an I/O request. It then saves the registers and PSW, and sets the FRR stack pointer to the I/O stack.		
3 If the interrupted process was a task, the I/O interrupt handler stores the TOD clock value for job step timing.		
4 IOS receives control to process the I/O request. IOS reenters the I/O IH at the entry point DISMISS.		

Diagram 19-9. I/O Interruption Handler (IEAVEIO) (Part 3 of 4)

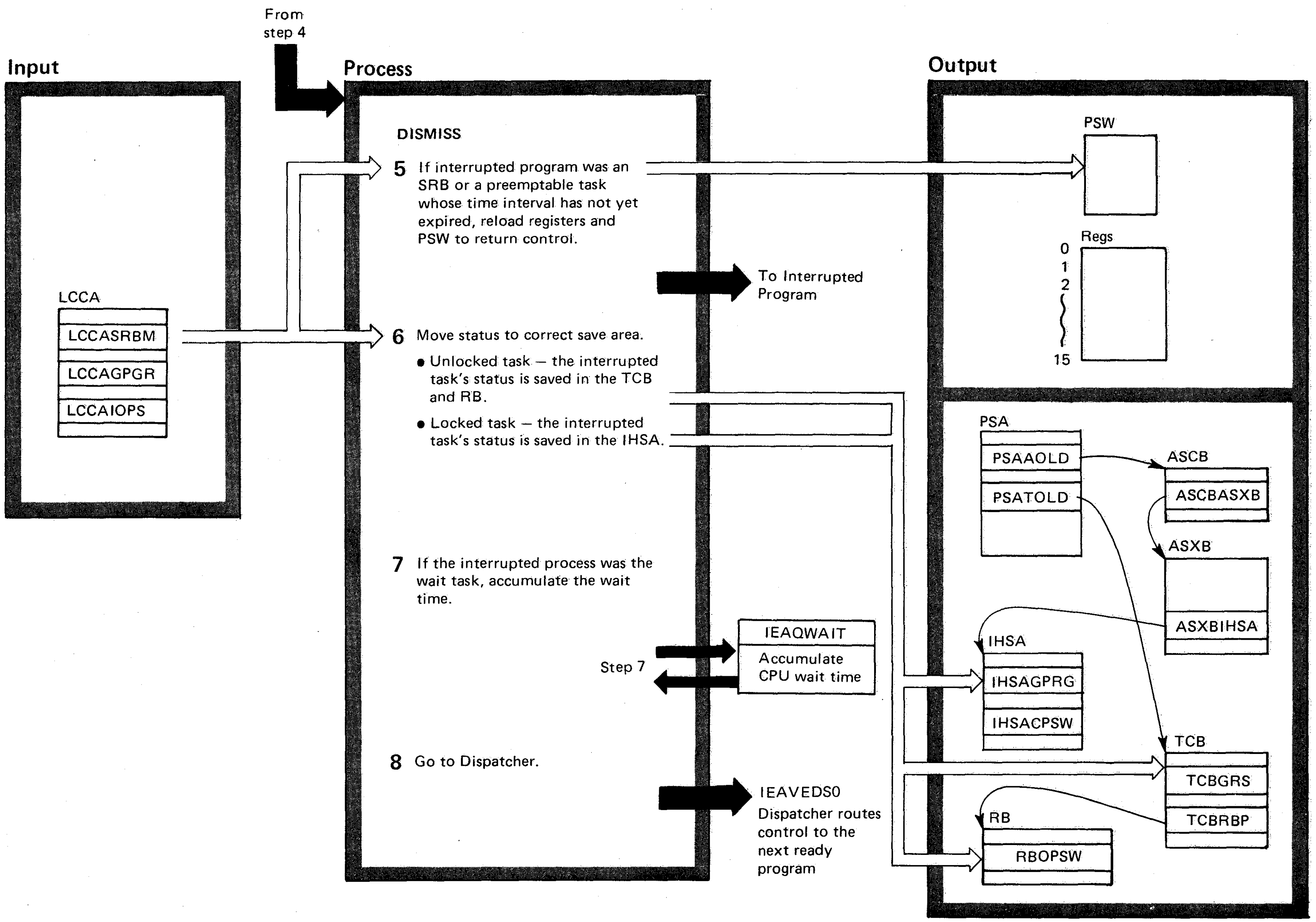
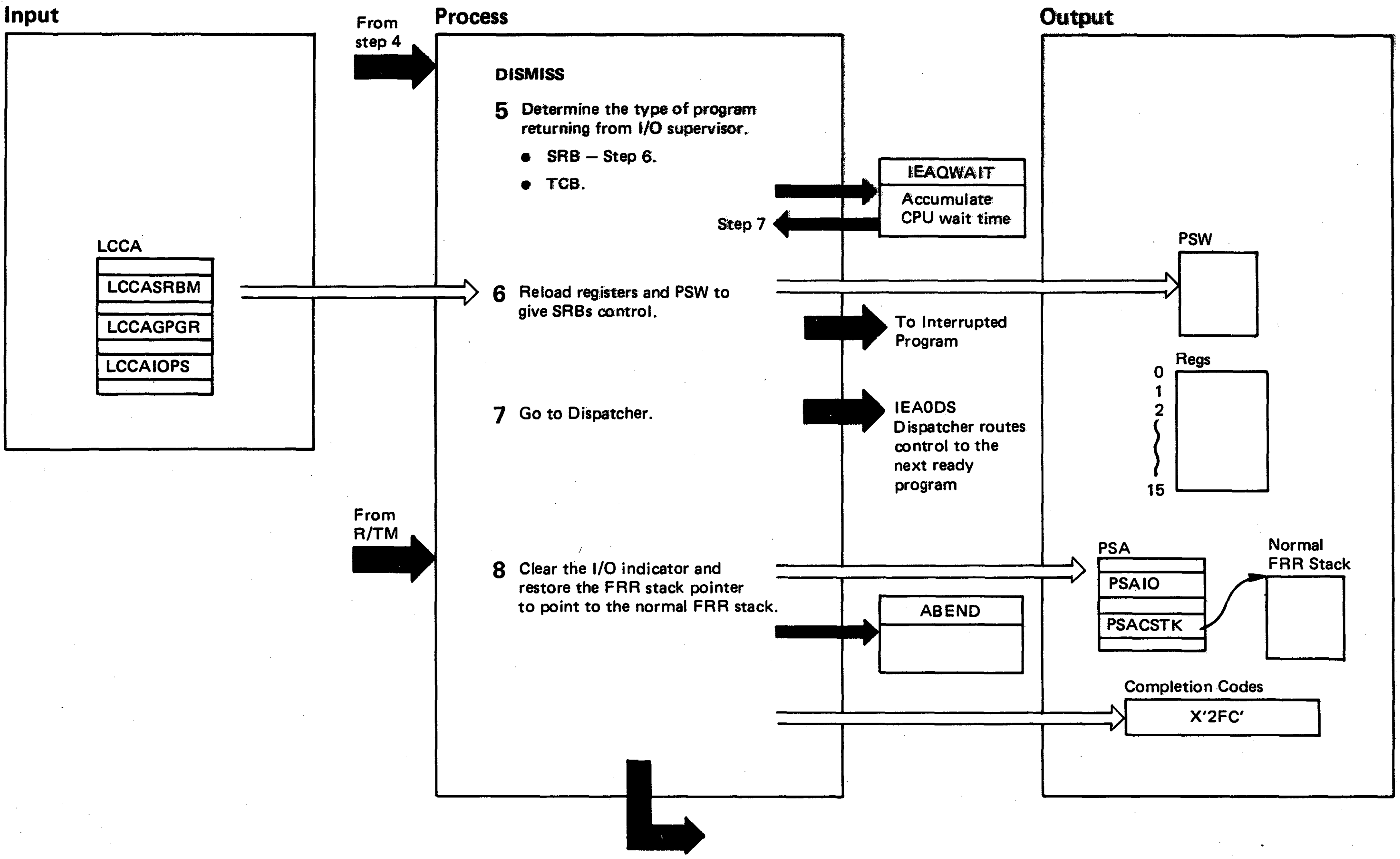

From step 4
Input
LCCA
LCCASRBM
LCCAGPGR
LCCAIOPS
Process
DISMISS
5 If interrupted program was an SRB or a preemptable task whose time interval has not yet expired, reload registers and PSW to return control.
To Interrupted Program
6 Move status to correct save area.
• Unlocked task — the interrupted task's status is saved in the TCB and RB.
• Locked task — the interrupted task's status is saved in the IHSA.
7 If the interrupted process was the wait task, accumulate the wait time.
IEAQWAIT
Accumulate CPU wait time
Step 7
8 Go to Dispatcher.
IEAVEDS0
Dispatcher routes control to the next ready program
Output
PSW
Regs
0
1
2
15
PSA
PSAAOLD
PSATOLD
ASCB
ASCBASXB
ASXB
ASXBIHSA
IHSA
IHSAGPRG
IHSACPSW
TCB
TCBGRS
TCBRBP
RB
RBOPSW

Diagram 19-9. I/O Interruption Handler (IEAVEIO) (Part 2 of 4)

Extended Description	Module	Label
The I/O interruption handler (IH) saves the requester's status prior to giving the I/O supervisor (IOS) control. Furthermore, the I/O IH routes recursive I/O interruptions directly to the I/O supervisor.		
1 The I/O IH looks at the recursive bit (PSAIO) in the PSA to check for a recursive entry. IOS immediately receives control for recursive conditions. (I/O recursions will occur only if IOS enables for I/O interruptions.) Otherwise, normal processing occurs at step 2.	IEAVEIO	
2 The I/O IH stores the TOD clock value for CPU wait time calculations. The I/O IH sets the recursion bit in the PSA, PSAIO, to indicate that it is currently processing an I/O request. It then saves the registers and PSW, and sets the FRR stack pointer to the I/O stack.		
3 The I/O IH handles the processing for SRBs, locked TCBs, and unlocked TCBs. The processing differs, as follows: • SRBs — The requester's status is saved in the LCCA. • Locked TCBs — The I/O IH saves the requester's status in the IHSA. • Unlocked TCBs — The requester's status is saved in TCB and RB.		
4 IOS receives control to process the I/O request. IOS reenters the I/O IH at the entry point DISMISS.		

Diagram 19-9. I/O Interruption Handler (IEAVEIO) (Part 3 of 4)



DISMISS

5 Determine the type of program returning from I/O supervisor.
- SRB – Step 6.
- TCB.

6 Reload registers and PSW to give SRBs control.

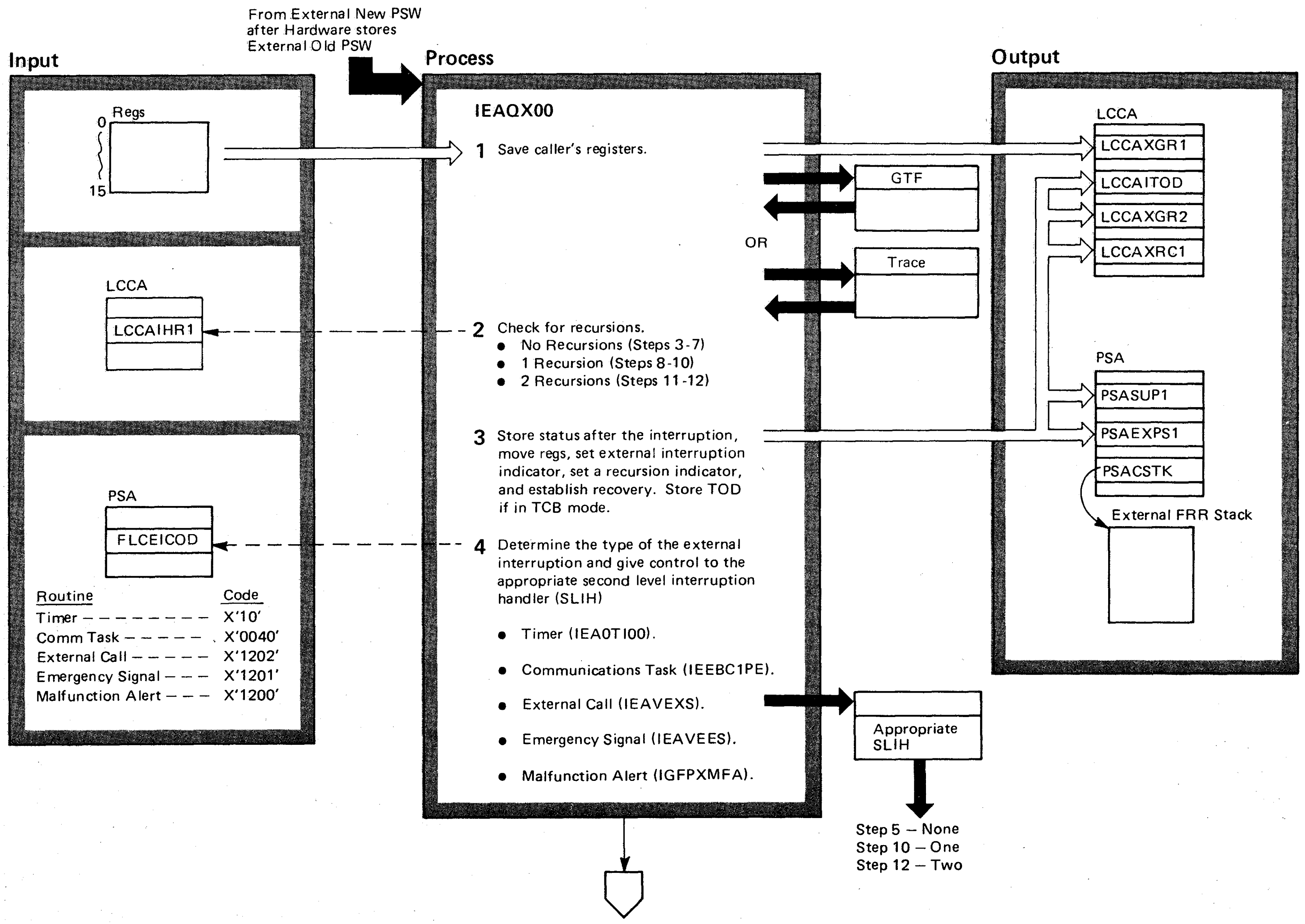
7 Go to Dispatcher.

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

Diagram 19-9. I/O Interruption Handler (IEAVEIO) (Part 4 of 4)

Extended Description	Module	Label
5 SRBs do not have CPU wait time calculations done. SRBs have their status restored by loading the PSW and registers. The I/O IH resets the PSAIO bit, and restores the FRR stack pointer as it was before the I/O interruption occurred. If a task has not executed for a specific interval, the interrupt processing time is deducted from this execution time. Control is returned to the interrupted routine after its status is restored.		
6 Status is stored in a different area depending on the interrupted process.		
7 The wait task has CPU wait time calculations done by the IEAQWAIT routine.		
8 The I/O IH routes control to the dispatcher. The I/O IH saves the registers and PSW and resets the PSAIO bit.		

Diagram 19-10. External First Level Interruption Handler (IEAVEEXT) (Part 1 of 6)



Routine	Code
Timer	X'10'
Comm Task	X'0040'
External Call	X'1202'
Emergency Signal	X'1201'
Malfunction Alert	X'1200'

1 Save caller's registers.

2 Check for recursions.
- No Recursions (Steps 3-7)
- 1 Recursion (Steps 8-10)
- 2 Recursions (Steps 11-12)

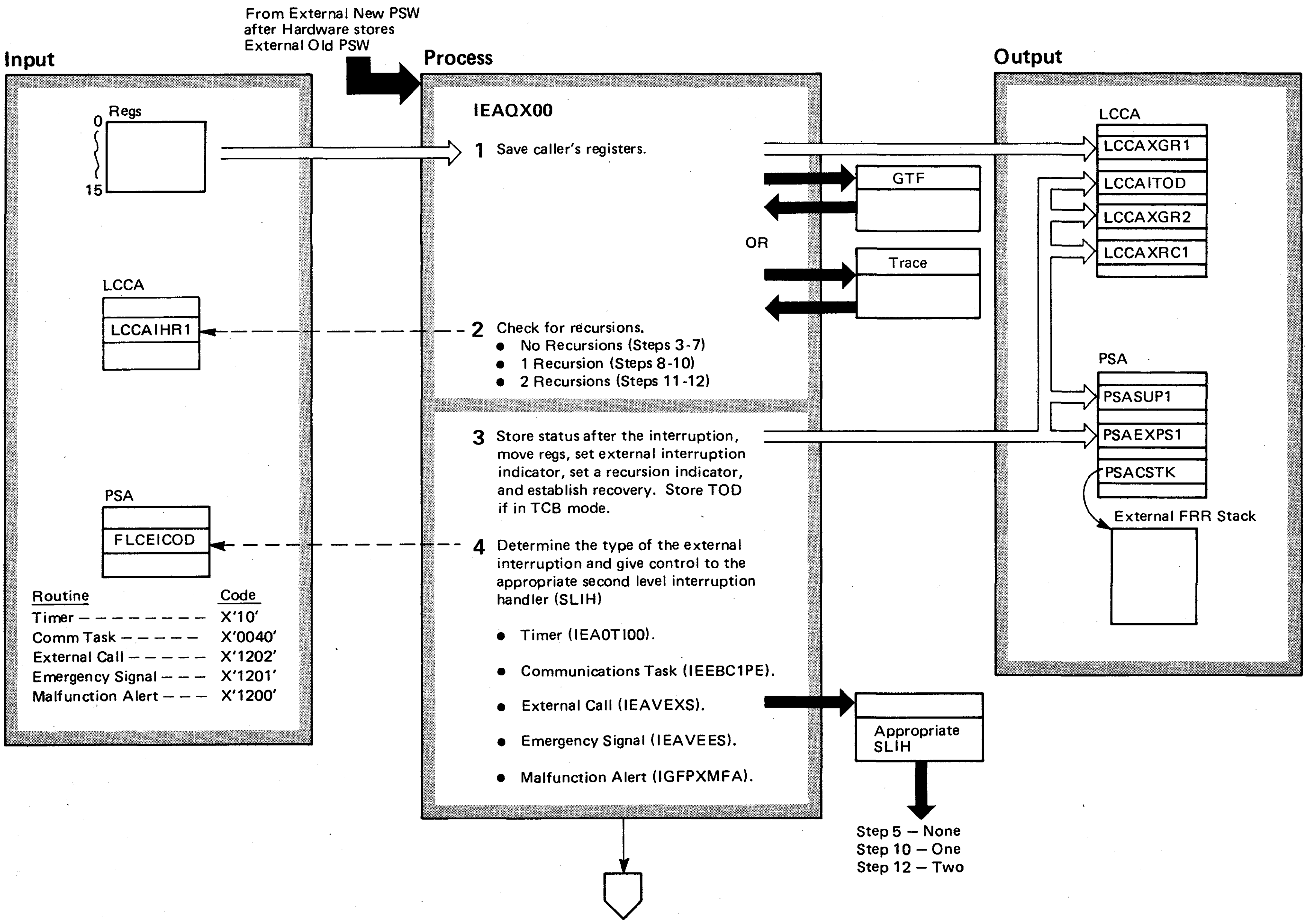
3 Store status after the interruption, move regs, set external interruption indicator, set a recursion indicator, and establish recovery. Store TOD if in TCB mode.

4 Determine the type of the external interruption and give control to the appropriate second level interruption handler (SLIH)
- Timer (IEA0TI00).
- Communications Task (IEEBC1PE).
- External Call (IEAVEXS).
- Emergency Signal (IEAVEES).
- Malfunction Alert (IGFPXMFA).

Diagram 19-9. I/O Interruption Handler (IEAVEIO) (Part 4 of 4)

Extended Description	Module	Label
5 SRBs do not have CPU wait time calculations done. The wait task has CPU wait time calculations done by the IEAQWAIT routine. SRBs and non-preemptable tasks have their status restored by loading the PSW and registers.		
6 The I/O IH resets the PSAIO bit, and restores the FRR stack pointer as it was before the I/O interruption occurred.		
7 The I/O IH routes control to the dispatcher. The I/O IH restores the registers and PSW and resets the PSAIO bit.		
8 The I/O IH FRR (functional recovery routine) clears the I/O interruption indicator, and points the FRR stack pointer in the PSA to the normal FRR stack. It also terminates the interrupted program, with a completion code of X'2FC'.		

Diagram 19-10. External First Level Interruption Handler (IEAVEEXT) (Part 1 of 6)



Routine	Code
Timer	X'10'
Comm Task	X'0040'
External Call	X'1202'
Emergency Signal	X'1201'
Malfunction Alert	X'1200'

1 Save caller's registers.

2 Check for recursions.
- No Recursions (Steps 3-7)
- 1 Recursion (Steps 8-10)
- 2 Recursions (Steps 11-12)

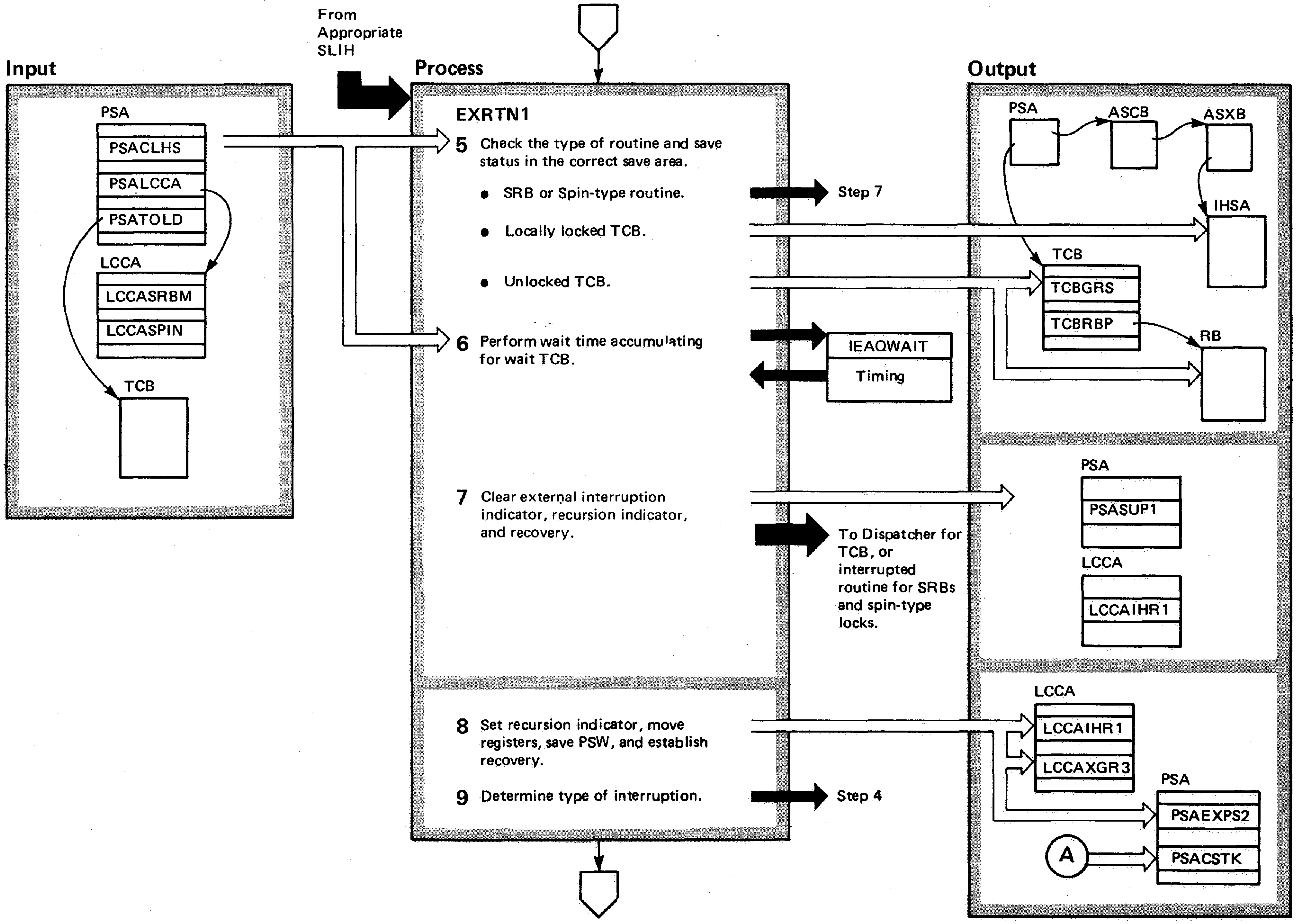
3 Store status after the interruption, move regs, set external interruption indicator, set a recursion indicator, and establish recovery. Store TOD if in TCB mode.

4 Determine the type of the external interruption and give control to the appropriate second level interruption handler (SLIH)
- Timer (IEAOTI00).
- Communications Task (IEEBC1PE).
- External Call (IEAVEXS).
- Emergency Signal (IEAVEES).
- Malfunction Alert (IGFPXMFA).

Diagram 19-10. External First Level Interruption Handler (IEAVEEXT) (Part 2 of 6)

Extended Description	Module	Label
The external first level interruption handler (FLIH) routes control to the appropriate second level interruption handler routine after an external interruption. The external FLIH saves the status of the program operating at the time of the interruption. The external FLIH can handle recursions, when external interruptions – either EMS or MFA – occur in an external second level interruption handler. Two levels of recursions can be processed by the external FLIH.		
1 The external FLIH initially saves the status of the program currently operating in a temporary location in the LCCA. The status will be moved later. The external FLIH then tracks the interruption with GTF or trace.	IEAVEEXT	IEAQEX00
2 The external FLIH can process two levels of recursions. Steps 3 - 7 show processing for no recursions; steps 8 - 10 show processing for one recursion; and steps 11 - 12 show processing for two recursions. Note that all levels of recursions use the function in step 4.		
3 The external FLIH: • Saves the TOD (time-of-day) value in the LCCAITOD field if in TCB mode. • Stores the PSW in the PSAEXPS1 field. • Moves the register from the LCCAXGR1 field to the LCCAXGR2 field (to prevent overlaying the LCCAXGR1 field in the event of a recursion). • Sets the external interruption bit, LCCAXRC1, in field LCCAIHR1, to indicate one level of recursion. • Sets the recovery indicator in field PSASUP1. • Sets the current FRR stack pointer to the external FLIH FRR stack.		
4 The external FLIH determines which one of the five types of external interruption occurred. These interruptions, and how they occur, follow: • Timer. Occurs when a selected timer interval expires. • Comm Task. Occurs when the operator presses the external interruption key on the operator's console. • External call. Occurs after a user issues a SIGP (signal processor) via an RPSGNL request. • Emergency signal. Occurs after a user issues a SIGP. • Malfunction alert. Occurs if another CPU fails.		
The external FLIH routes control to the appropriate second level interruption handler (SLIH). Control returns to the external FLIH from the SLIHs at: • For no recursions – entry point EXRTN1 (step 5). • For one recursion – entry point EXRTN2 (step 10). • For two recursions – entry point REC2RTN (step 12).		PROCESS

Diagram 19-10. External First Level Interruption Handler (IEAVEEXT) (Part 3 of 6)



5 Check the type of routine and save status in the correct save area.

- SRB or Spin-type routine.
- Locally locked TCB.
- Unlocked TCB.

6 Perform wait time accumulating for wait TCB.

7 Clear external interruption indicator, recursion indicator, and recovery.

8 Set recursion indicator, move registers, save PSW, and establish recovery.

9 Determine type of interruption.

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

5 After receiving control from the appropriate SLIH, the external FLIH performs these actions:

- For SRBs or spinning routines, the external FLIH clears the recovery bit PSAEXT and the external interruption indicator LCCAXRC1, and returns to the interrupted program (via an LPSW instruction).
- For locally locked TCBs, the external FLIH moves the registers from LCCAXGR2 and the PSW from PSAEXPS1 into the IHSA.
- For unlocked TCBs, the registers are moved into the TCB and the PSW is moved into the RB.

6 If the interrupted routine was the WAIT TCB, the external FLIH passes control to the Wait routine (IEAQWAIT) to perform wait time accumulation.

7 The external FLIH clears the PSAEXT bit and the LCCAXRC1 field for SRBs or spinning routines. Control goes to the dispatcher for TCBs.

8 For one recursion, the external FLIH sets a recursion indicator (a bit) in LCCAIHR1. The routine also moves the registers and PSW to LCCAXGR3, to prevent overlaying them in case of another recursion. The PSW goes to PSAEXPS2.

9 The external FLIH determines the type of interruption, and gives the appropriate SLIH control (step 4). Control returns from the SLIH to EXRTN2 (step 10).

Diagram 19-10. External First Level Interruption Handler (IEAVEEXT) (Part 5 of 6)

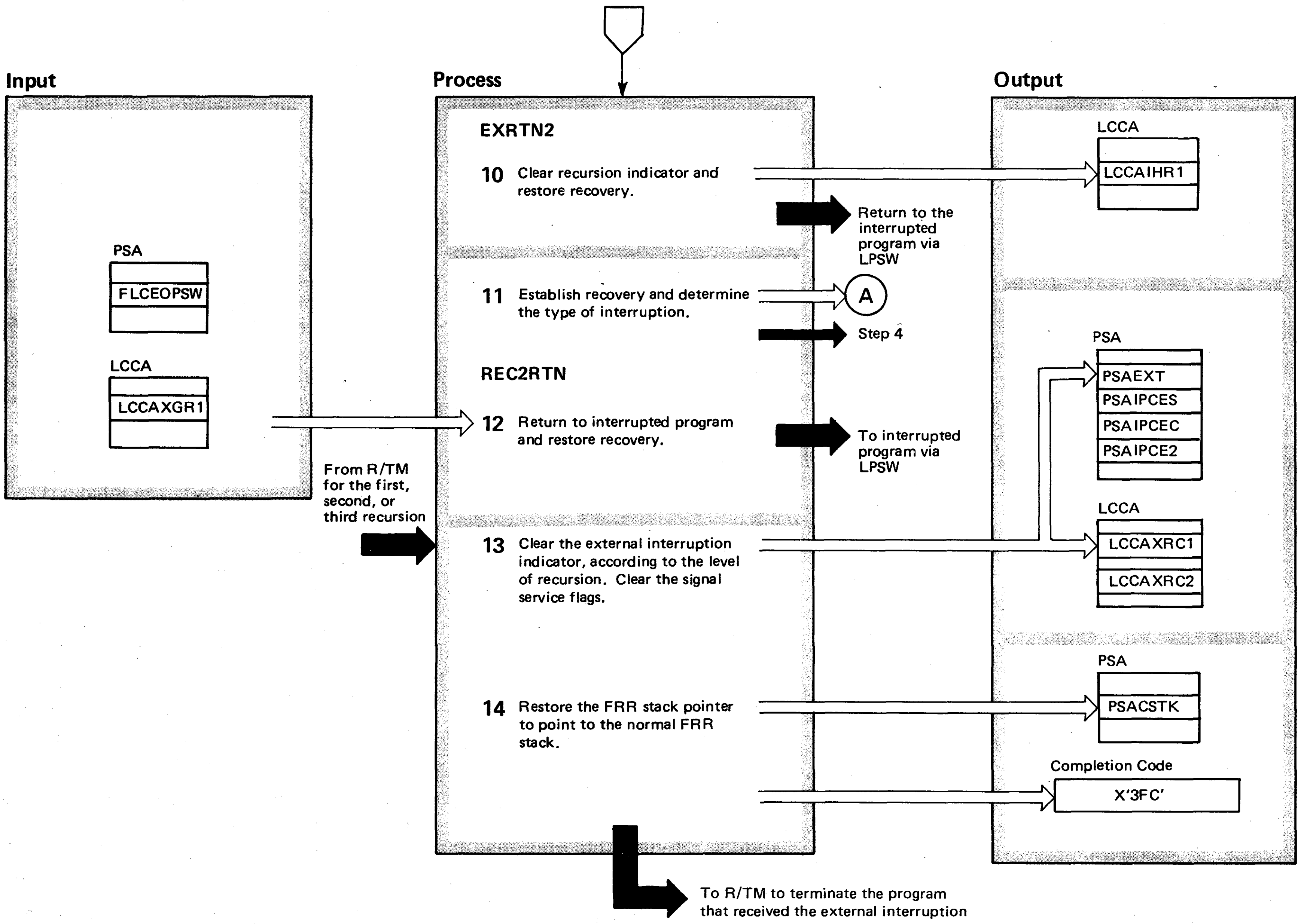
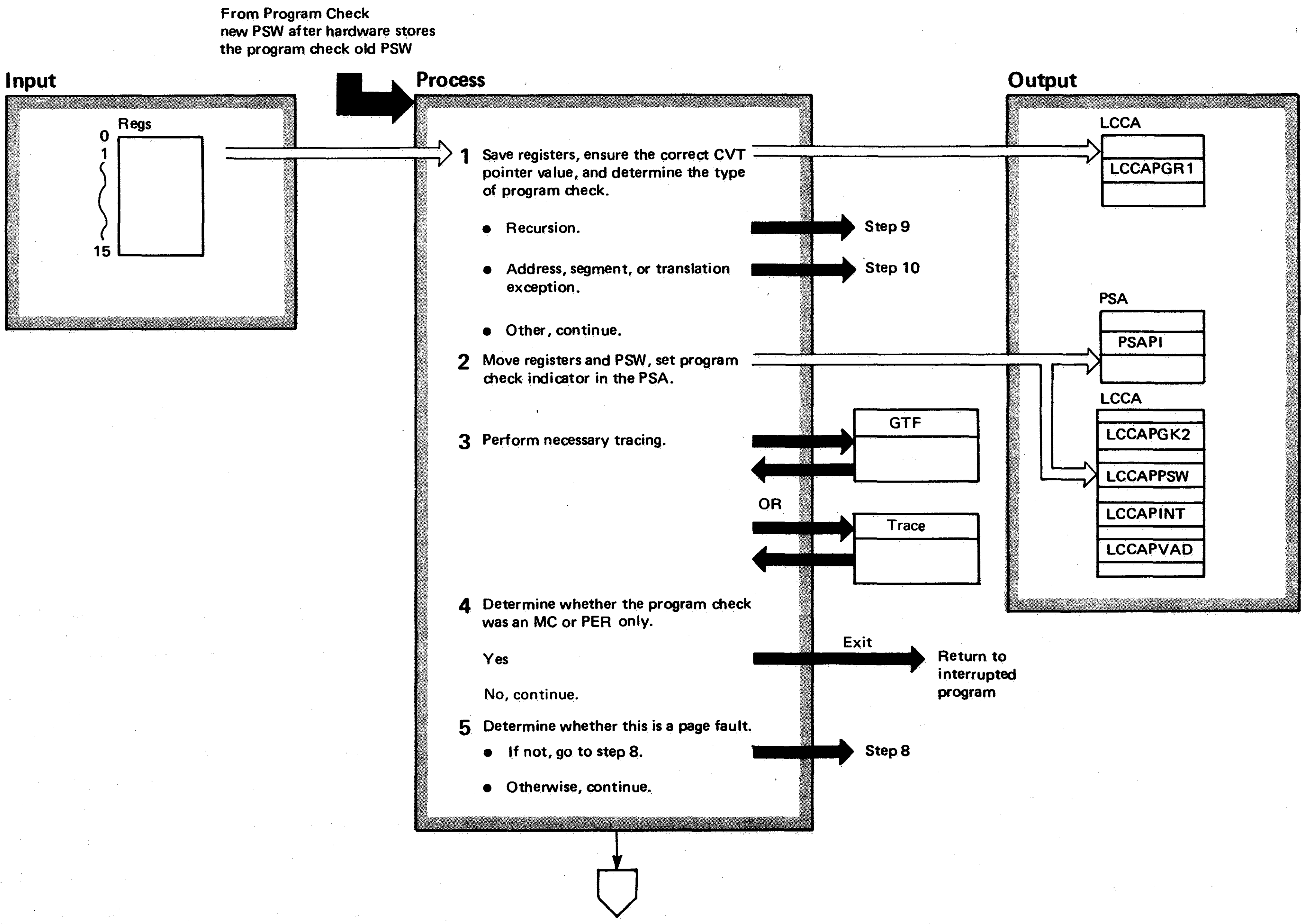

Input
PSA
FLCEOPSW
LCCA
LCCAXGR1
Process
EXRTN2
10 Clear recursion indicator and restore recovery.
Return to the interrupted program via LPSW
11 Establish recovery and determine the type of interruption.
A
Step 4
REC2RTN
12 Return to interrupted program and restore recovery.
To interrupted program via LPSW
From R/TM for the first, second, or third recursion
13 Clear the external interruption indicator, according to the level of recursion. Clear the signal service flags.
14 Restore the FRR stack pointer to point to the normal FRR stack.
To R/TM to terminate the program that received the external interruption
Output
LCCA
LCCAIHR1
PSA
PSAEXT
PSAIPCES
PSAIPCEC
PSAIPCE2
LCCA
LCCAXRC1
LCCAXRC2
PSA
PSACSTK
Completion Code
X'3FC'

Diagram 19-10. External First Level Interruption Handler (IEAVEEXT) (Part 6 of 6)

Extended Description	Module	Label
10 The external FLIH clears the recursion indicator, and returns to the interrupted program, via an LPSW instruction. Note that the interrupted program will be an appropriate external SLIH.		EXRTN2
11 For second level recursions, the external FLIH determines the type of interruption, and gives control to the appropriate SLIH (step 4). Control returns at entry point REC2RTN (step 12).		
12 The external FLIH second level recursion FRR stack has been set at entry to step 11; it is reset at step 12. The external FLIH returns to the interrupted program.		REC2RTN
13 The external IH has three FRRs (functional recovery routines), one for each level of recursion. They all clear various indicators, restore the FRR stack pointer to point to the current FRR stack, and terminate the program that received the interruption.		
For no recursions, the first FRR clears the external interruption indicator, PSAEXT, clears the recursion indicator, LCCAXRC1 in the LCCA, and clears any signal service indicators in the PSA fields PSAIPCES and PSAIPCEC.	IEAVEE1R	
For one recursion, the second FRR clears the recursion indicator in the LCCA, LCCAXRC2, and the emergency signal service routine recursion indicator, PSAIPCE2. Note that the emergency signal primary indicator, PSAIPCES, will not be cleared if it is not an emergency signal (EMS) recursion. The external FLIH restores the FRR stack pointer.	IEAVEE2R	
For the second recursion, the last FRR clears the EMS recursion indicator, PSAIPCE2.	IEAVEE3R	
14 All the FRRs point the PSACSTK field to the normal FRR stack, and terminate the program that received the external interruptions with a X'3FC' completion code.	IEAVEE1R IEAVEE2R IEAVEE3R	

Diagram 19-11. Program Check Interruption Handler (PC IH) (IEAVEPC) (Part 1 of 12)



1 Save registers, ensure the correct CVT pointer value, and determine the type of program check.

- Recursion.
- Address, segment, or translation exception.
- Other, continue.

2 Move registers and PSW, set program check indicator in the PSA.

3 Perform necessary tracing.

4 Determine whether the program check was an MC or PER only.

Yes

No, continue.

5 Determine whether this is a page fault.

- If not, go to step 8.
- Otherwise, continue.

Diagram 19-11. Program Check Interruption Handler (PC IH) (IEAVEPC) (Part 2 of 12)

Extended Description	Module	Label
The program check IH (interruption handler) receives control from the program check NPSW after a program check occurs, traces the program check via GTF or the trace facility, and routes control to the appropriate routine. The program check IH processes page faults by giving control to real storage management, processes MC (monitor call instructions) and PER (program event recording) interruptions by noting their occurrence and returning to the interrupted program, and processes the remaining types of program checks by routing control to R/TM.		
1 The program check IH saves the registers in LCCAPGR1 and ensures that the CVT pointer points to the CVT. Step 9 shows how the program check IH processes recursions, and step 10 shows processing for address, segment, and translation exceptions.	IEAVEPC	IEAQPK00
2 The program check moves the registers from LCCAPGR1 to LCCAPGR2, and the PSW from FLCPOPSW to LCCAPPSW, saves the interruptions code in LCCAPINT, and the translation address in LCCAPVAD to prevent losing this information if a recursion occurs. The recursion indicator in the PSA is also set at this time.		
3 The program check IH gives control to GTF or, optionally, the trace facility, to record the occurrence of the interruption.		
4 The program check IH returns control to the interrupted program if either an MC or PER interruption alone occurred. (A PER condition can occur with any other program check.)		
5 The program check IH determines whether a page fault caused the program check. For page faults, control continues at step 6. If the program check was not caused by a page fault, control goes to step 8, to continue processing.		

Diagram 19-11. Program Check Interruption Handler (PC IH) (IEAVEPC) (Part 3 of 12)

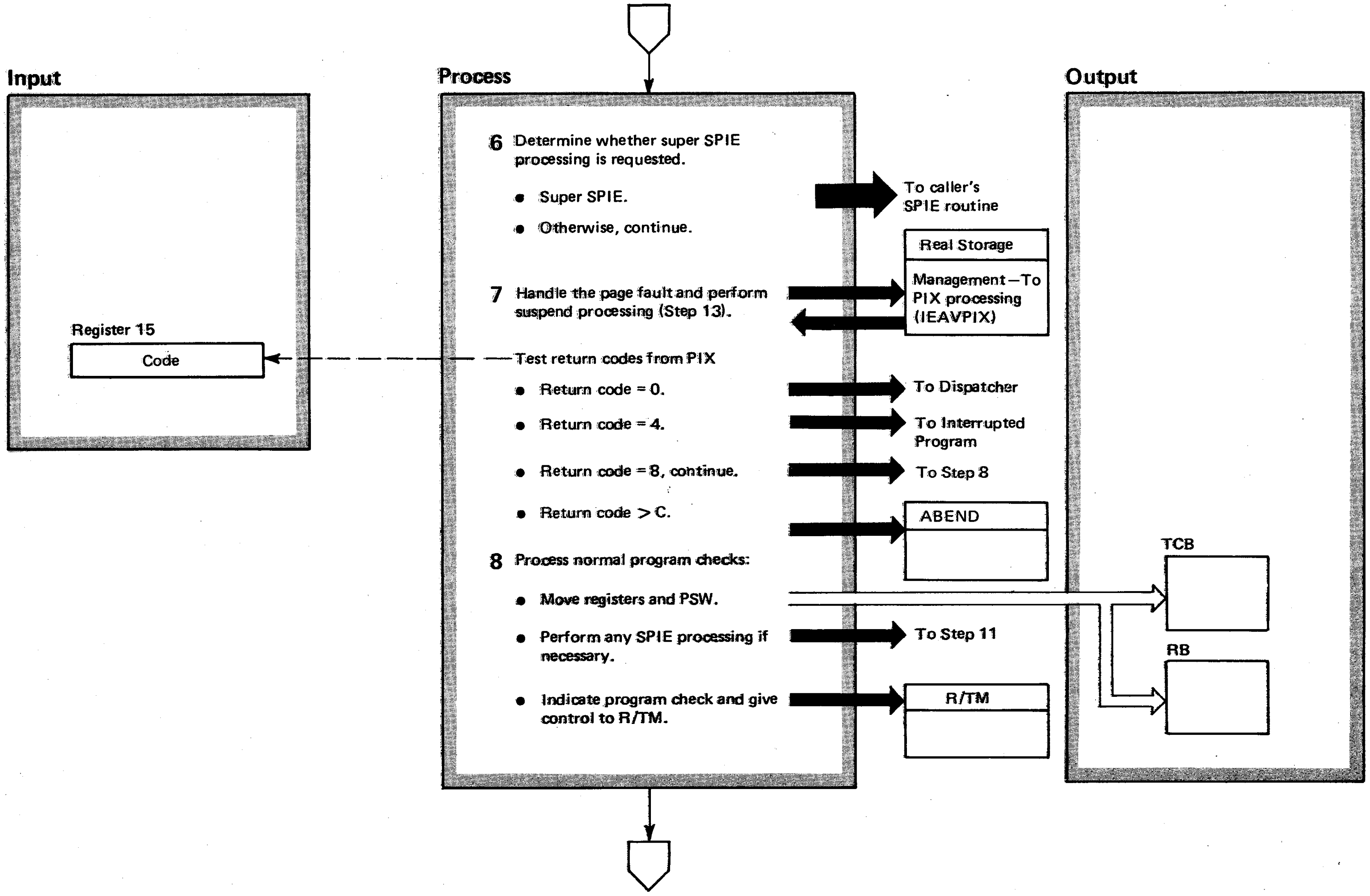

Input
Register 15
Code
Process
6 Determine whether super SPIE processing is requested.
• Super SPIE.
• Otherwise, continue.
7 Handle the page fault and perform suspend processing (Step 13).
Test return codes from PIX
• Return code = 0.
• Return code = 4.
• Return code = 8, continue.
• Return code > C.
8 Process normal program checks:
• Move registers and PSW.
• Perform any SPIE processing if necessary.
• Indicate program check and give control to R/TM.
To caller's SPIE routine
Real Storage Management—To PIX processing (IEAVPIX)
To Dispatcher
To Interrupted Program
To Step 8
ABEND
To Step 11
R/TM
Output
TCB
RB

Diagram 19-11. Program Check Interruption Handler (PC IH) (IEAVEPC) (Part 4 of 12)

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

6 The program check IH will determine whether control should be routed to the caller's super SPIE routine.
If the caller's super SPIE routine should receive control the program check IH:

- Sets up the PIE and PICA.
- Sets up the TCB and RB to enter the SPIE exit.
- Route control to the caller's super SPIE routine via an LPSW instruction.

If the caller does not have a super SPIE routine, processing continues at step 7.

7 Control goes to the PIX routine, part of Real Storage Management, to perform the actual paging. PIX interacts with the program check IH's suspend routine (IEAVSUSP) to logically suspend the program that received the page fault if this is a valid page fault and paging I/O is required.

PIX passes one of four return codes to the program check IH in register 15. These codes and the actions taken by the program check IH follow:

0 — The program was suspended. Control goes to the dispatcher, to dispatch the next ready unit of work.

4 — Either the real storage frame containing the page was reclaimed or a valid page has been referenced for the first time — no paging I/O was necessary. Control goes back to the program that received the page fault.

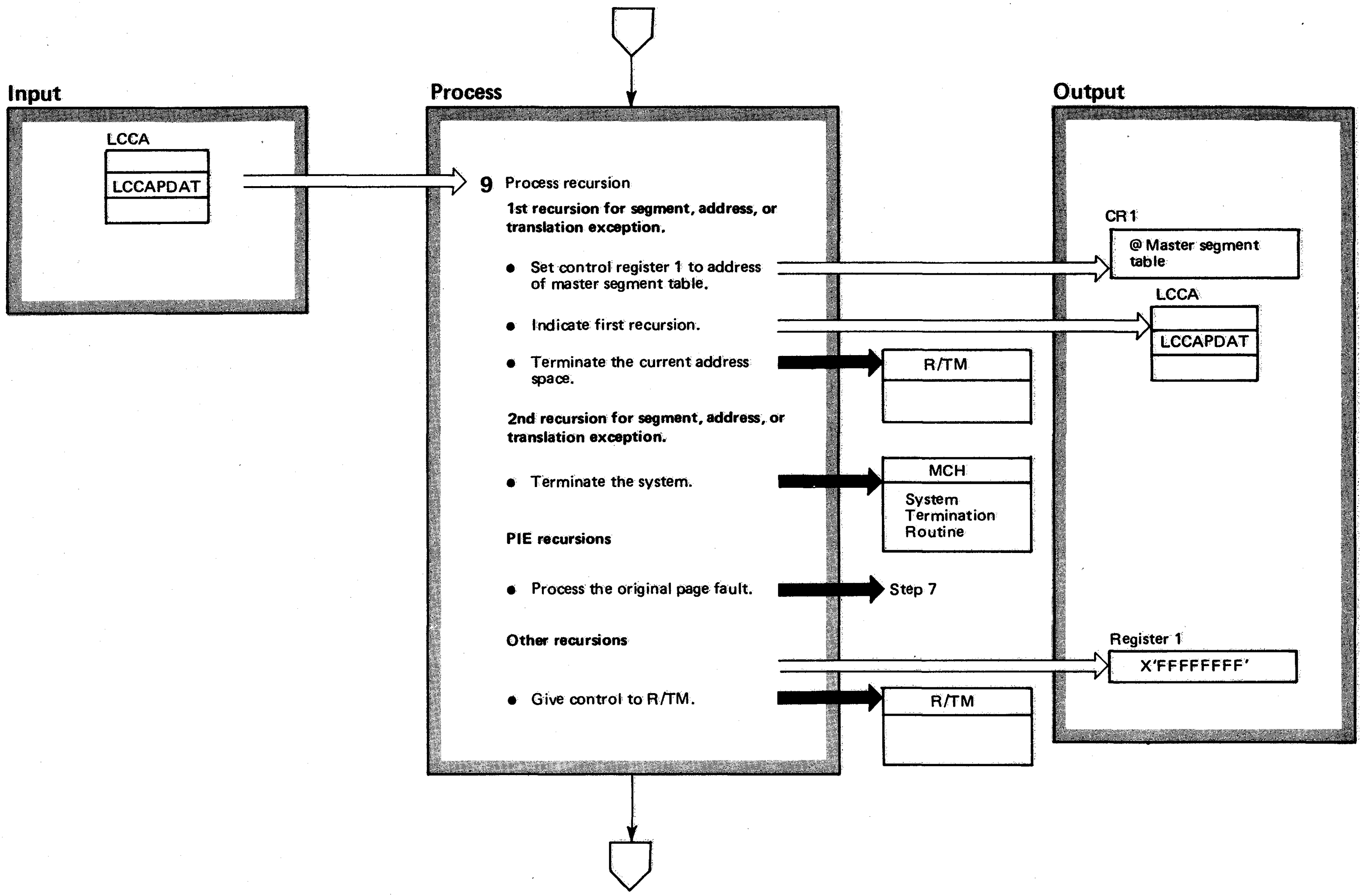
8 — The page was not valid. This will be treated as an X'0C4' abend. Control goes to the next series of operations in the program check IH.

C or greater — An internal error occurred in PIX. The task will be abnormally terminated with a X'028' code.

8 The program check IH performs processing for non-DAT type program checks:

- For unlocked TCBs, it stores the status of the interrupted program in the TCB and RB.
- Any SPIE processing will be performed (step 11), if requested by the caller.
- For all other cases, control goes to R/TM.

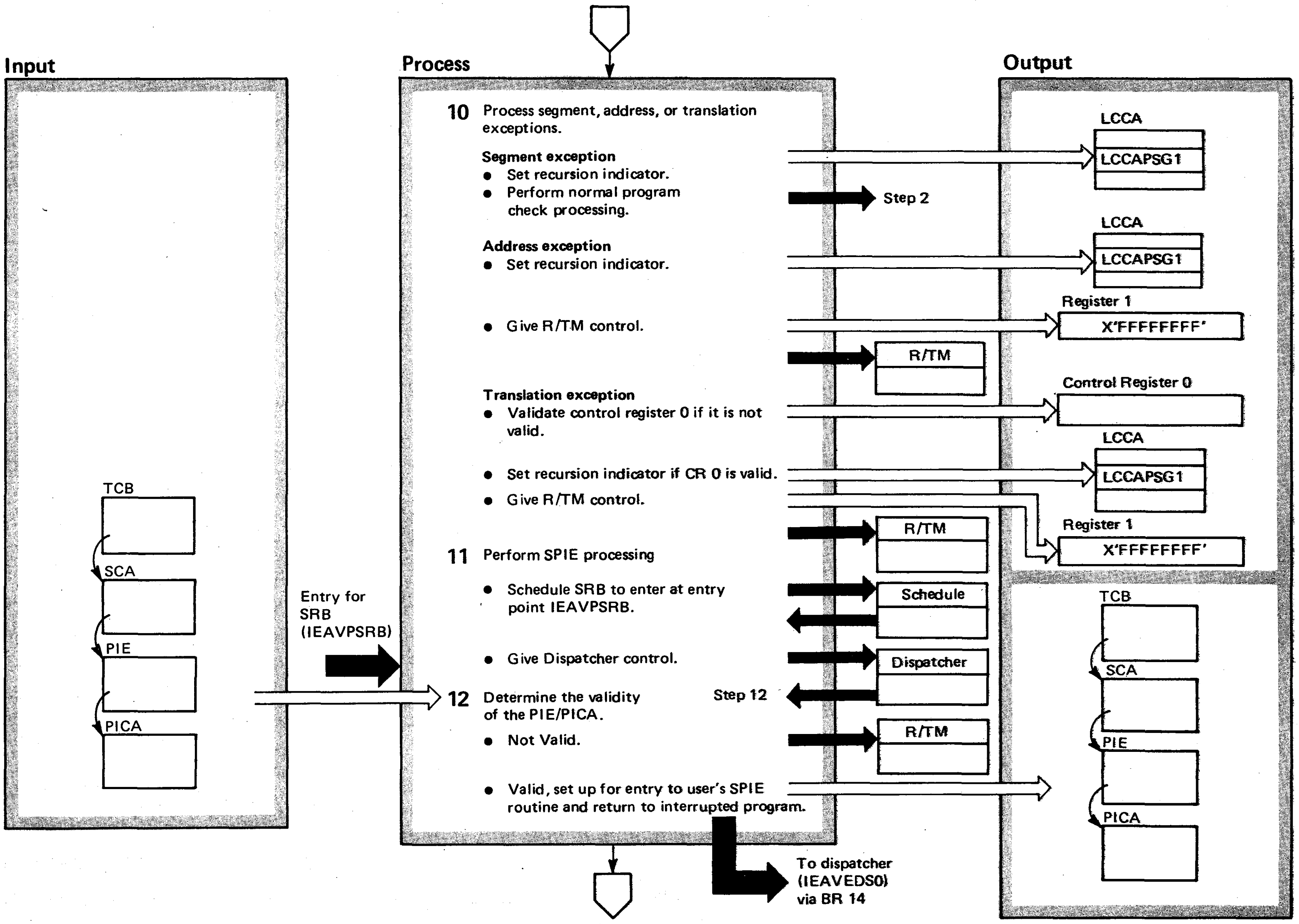
Diagram 19-11. Program Check Interruption Handler (PC IH) (IEAVEPC) (Part 5 of 12)



Input
LCCA
LCCAPDAT
Process
9 Process recursion
1st recursion for segment, address, or translation exception.
● Set control register 1 to address of master segment table.
● Indicate first recursion.
● Terminate the current address space.
2nd recursion for segment, address, or translation exception.
● Terminate the system.
PIE recursions
● Process the original page fault.
Other recursions
● Give control to R/TM.
R/TM
MCH
System Termination Routine
Step 7
R/TM
Output
CR1
@ Master segment table
LCCA
LCCAPDAT
Register 1
X'FFFFFFFF'

Extended Description	Module	Label
9 Recursions in the program check IH for address, segment, and translation exceptions imply that the interruption handler may be unable to access critical data. Therefore, the first recursion will terminate the address space while the second recursion will terminate the system.		
A PIE recursion means that a page fault has occurred while trying to perform super SPIE processing. The program check IH discontinues super SPIE processing, and process the original page fault.		
R/TM handles other program check recursions.		

Diagram 19-11. Program Check Interruption Handler (PC IH) (IEAVEPC) (Part 7 of 12)



10 Process segment, address, or translation exceptions.

Segment exception
- Set recursion indicator.
- Perform normal program check processing.

Address exception
- Set recursion indicator.

- Give R/TM control.

Translation exception
- Validate control register 0 if it is not valid.

- Set recursion indicator if CR 0 is valid.
- Give R/TM control.

11 Perform SPIE processing

- Schedule SRB to enter at entry point IEAVPSRB.

- Give Dispatcher control.

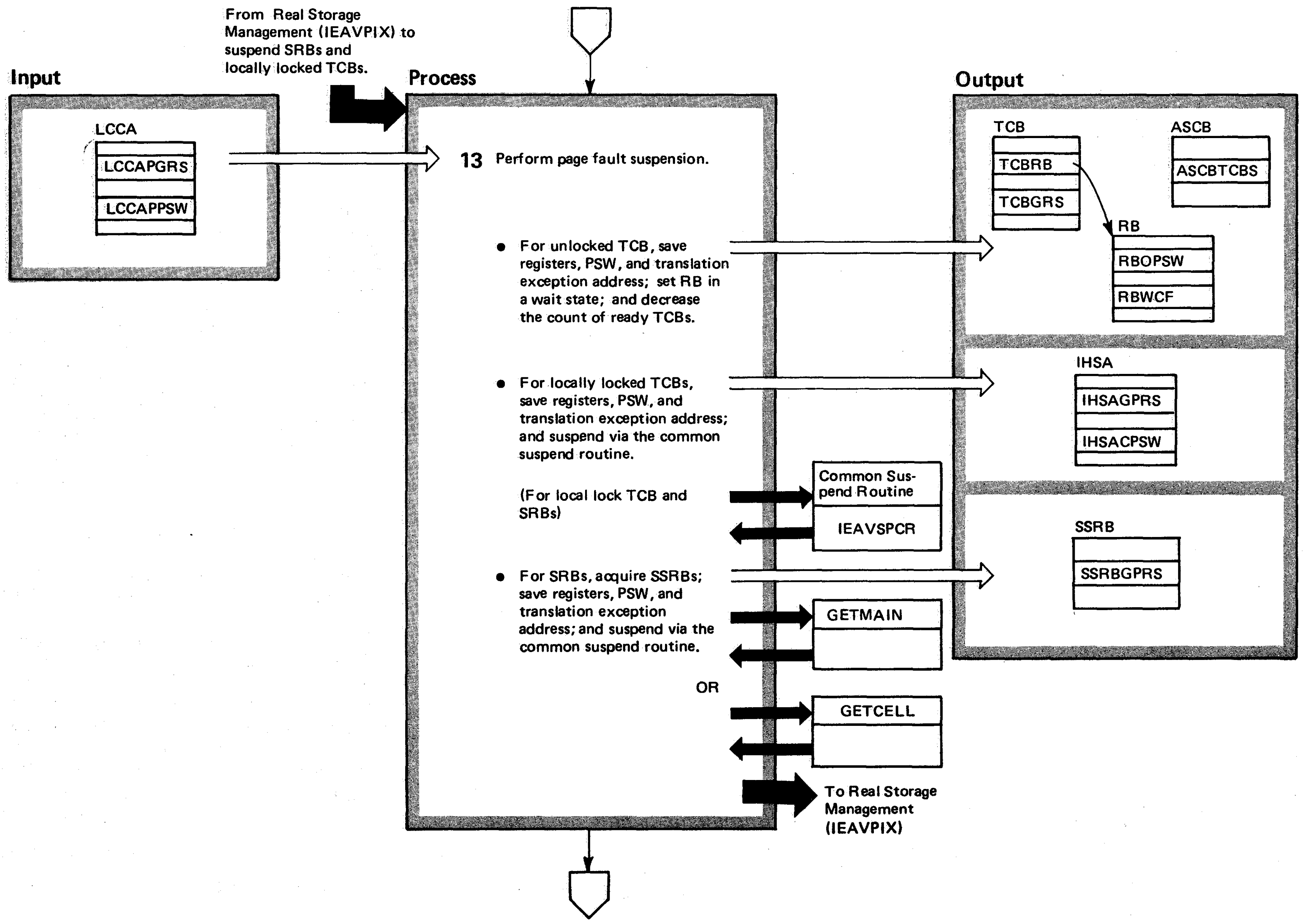
12 Determine the validity of the PIE/PICA.
- Not Valid.

- Valid, set up for entry to user's SPIE routine and return to interrupted program.

Diagram 19-11. Program Check Interruption Handler (PC IH) (IEAVEPC) (Part 8 of 12)

Extended Description	Module	Label
10 The program check IH handles segment, address, or translation exceptions in the following manner: • Segment exception. First, it sets the recursion indicator (to indicate any recursion conditions) in the LCCA. It then performs normal program check handling at step 2. • Address exception. Like segment exceptions, the program check IH sets a recursion indicator in the LCCA. Then, it sets a X'FFFFFFFF' value in register 1, to indicate to R/TM that the interrupted program's status remains in the LCCA. Control then goes to R/TM. • Tanslation exception. It validates control register 0 with the default values if necessary. Then it sets the recursion indicator, and gives control to R/TM with a x'FFFFFFFF' value in register 1.		
11 The program check IH readies the caller's SPIE routine, as follows. First, it schedules an SRB. The SRB will enter the program check IH SPIE subroutine at entry point IEAVPSRB. (The task is set non-dispatchable until the SRB routine completes.) After scheduling the SRB, control goes to the dispatcher to dispatch the SRB at a later time.		
12 The SRB enters at entry point IEAVPSRB. Here, the program checks the validity of the PIE/PICA. Control goes to R/TM to terminate the task if the PIE/PICA is not valid. If the PIE/PICA is valid, the program check IH sets the proper values in the TCB, RB, PIE, and PICA to give control to the user's SPIE routine. Control then goes to the dispatcher, which dispatches the task to the caller's SPIE routine.		IEAVPSRB

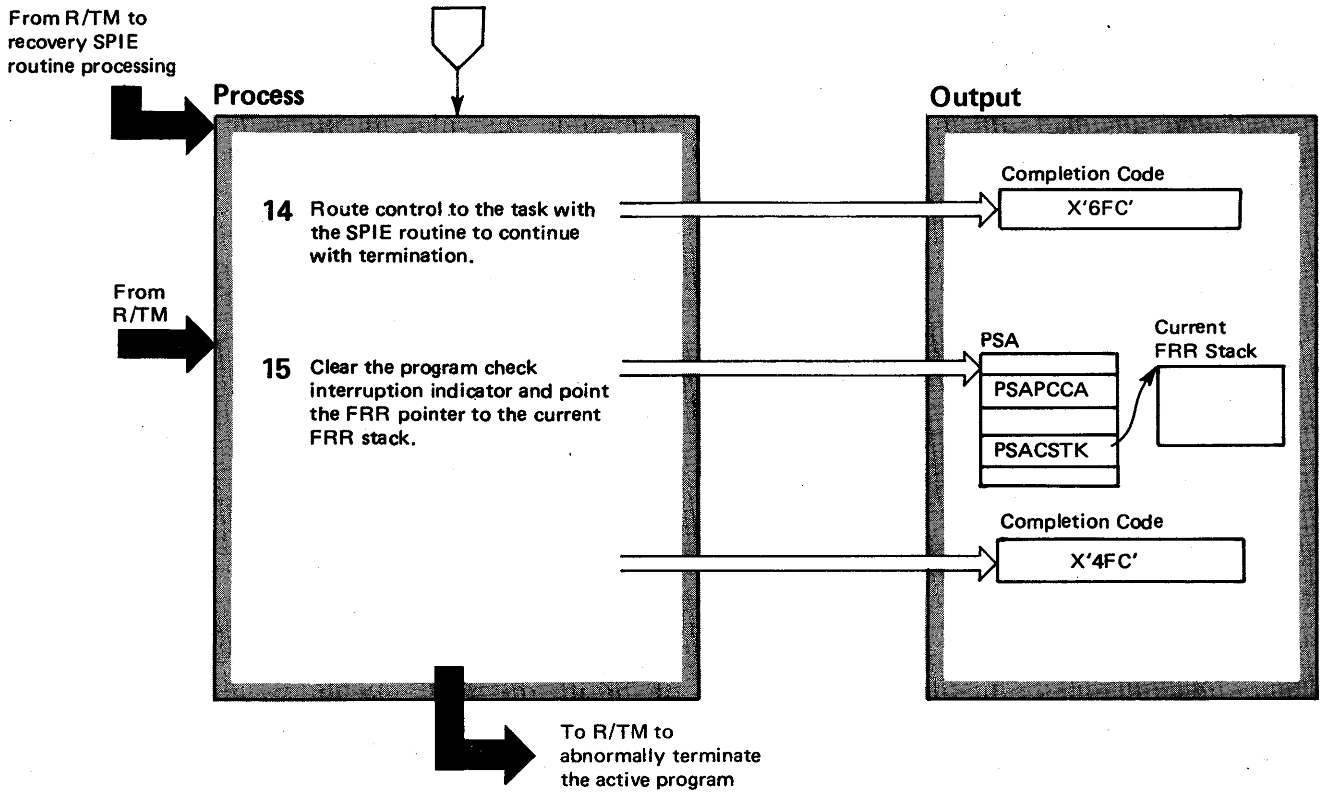
Diagram 19-11. Program Check Interruption Handler (PC IH) (IEAVEPC) (Part 9 of 12)



From Real Storage Management (IEAVPIX) to suspend SRBs and locally locked TCBs.
Input
LCCA
LCCAPGRS
LCCAPPSW
Process
13 Perform page fault suspension.
• For unlocked TCB, save registers, PSW, and translation exception address; set RB in a wait state; and decrease the count of ready TCBs.
• For locally locked TCBs, save registers, PSW, and translation exception address; and suspend via the common suspend routine.
(For local lock TCB and SRBs)
Common Suspend Routine
IEAVSPCR
• For SRBs, acquire SSRBs; save registers, PSW, and translation exception address; and suspend via the common suspend routine.
GETMAIN
OR
GETCELL
To Real Storage Management (IEAVPIX)
Output
TCB
TCBRB
TCBGRS
ASCB
ASCBTCBS
RB
RBOPSW
RBWCF
IHSA
IHSAGPRS
IHSACPSW
SSRB
SSRBGPRS

Extended Description	Module	Label
13 The program check IH suspends routines with a page fault in the following manner:		IEAVSUSP
• For unlocked TCBs, suspend processing saves the registers and PSW in the TCB and RB, and saves the translation exception address in the RB. Suspend processing sets the RB in the wait state by adding a 1 in the RBWCF field. Finally, suspend processing decreases the count of ready tasks in the ASCBTCBS field of the ASCB by 1.		
• For locally locked TCBs, suspend processing saves the registers, PSW, and translation exception address. Then, suspend processing gives control to the common suspend routine to complete suspension.		
• For SRBs, suspend processing first obtains an SSRB. Either GETMAIN or GETCELL will be used to acquire the storage for the SSRB, depending on various conditions. Suspend processing saves the registers, PSW, and translation exception address. The common suspend routine finishes suspend processing.		

Diagram 19-11. Program Check Interruption Handler (PC IH) (IEAVEPC) (Part 11 of 12)

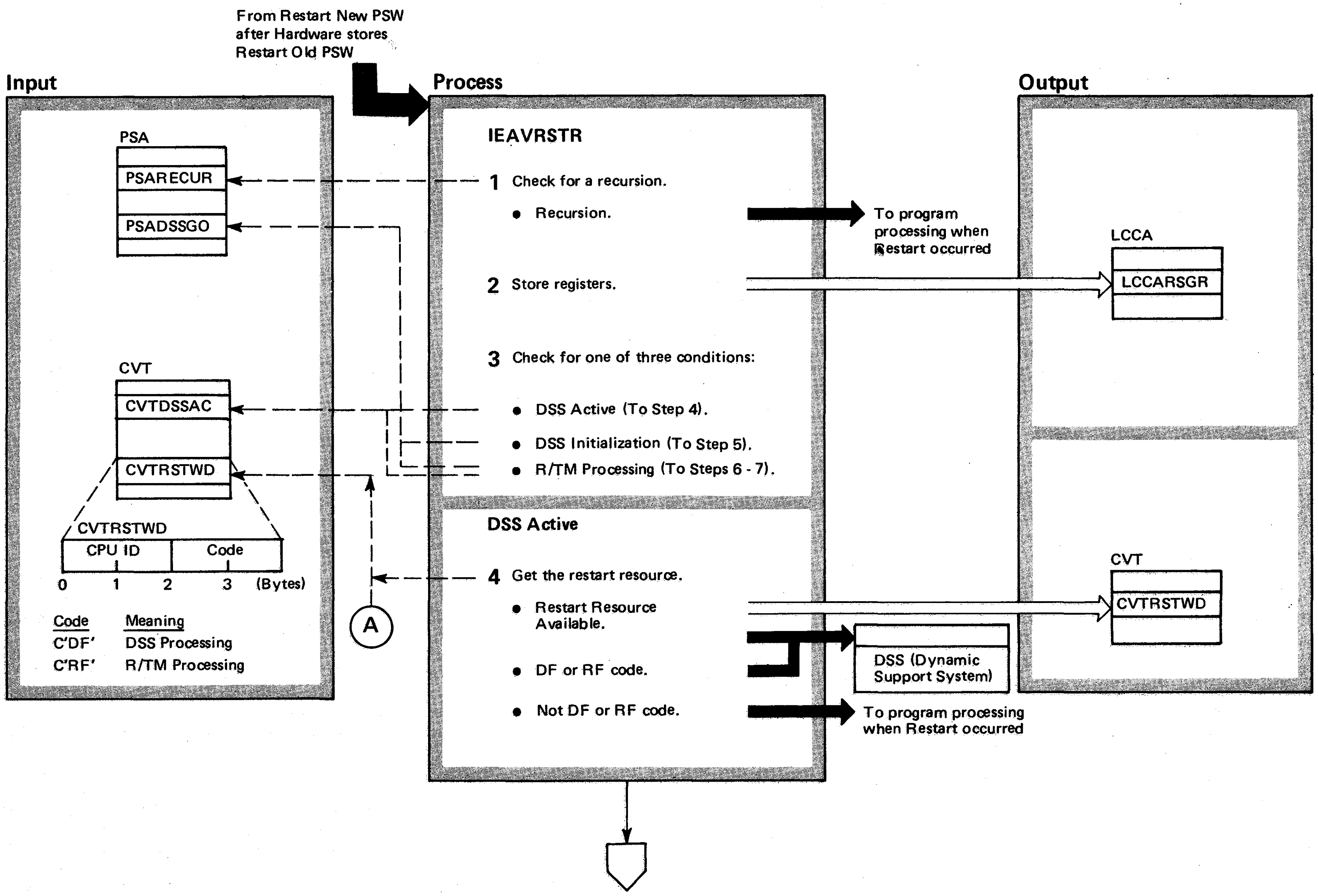


From R/TM to recovery SPIE routine processing
Process
Output
14 Route control to the task with the SPIE routine to continue with termination.
Completion Code
X'6FC'
From R/TM
15 Clear the program check interruption indicator and point the FRR pointer to the current FRR stack.
PSA
PSAPCCA
PSACSTK
Current FRR Stack
Completion Code
X'4FC'
To R/TM to abnormally terminate the active program

Diagram 19-11. Program Check Interruption Handler (PC IH) (IEAVEPC) (Part 12 of 12)

Extended Description	Module	Label
14 The program check IH will recover errors that occur during the SRB portion of SPIE processing. Control goes to the task with the SPIE routine to continue with termination. The recovery routine passes an X'6FC' completion code.		
15 The program check IH FRR (IEAVEPCR) clears the program check interruption indicator and points the FRR stack pointer, PSACSTK, to the current FRR stack. The FRR abnormally terminates the active program with a completion code of X'4FC'.	IEAVEPCR	IEAVEPCR
The program check IH RMTR frees the SSRB.		IEAVEFRE

Diagram 19-12. Restart Interruption Handler (IEAVERES) (Part 1 of 4)



From Restart New PSW after Hardware stores Restart Old PSW
Input
PSA
PSARECUR
PSADSSGO
CVT
CVTDSSAC
CVTRSTWD
CVTRSTWD
CPU ID
Code
0
1
2
3
(Bytes)
Code
Meaning
C'DF'
DSS Processing
C'RF'
R/TM Processing
A
Process
IEAVRSTR
1 Check for a recursion.
• Recursion.
2 Store registers.
3 Check for one of three conditions:
• DSS Active (To Step 4).
• DSS Initialization (To Step 5).
• R/TM Processing (To Steps 6 - 7).
DSS Active
4 Get the restart resource.
• Restart Resource Available.
• DF or RF code.
• Not DF or RF code.
To program processing when Restart occurred
DSS (Dynamic Support System)
To program processing when Restart occurred
Output
LCCA
LCCARSGR
CVT
CVTRSTWD

Diagram 19-12. Restart Interruption Handler (IEAVERES) (Part 2 of 4)

Extended Description	Module	Label
The restart interruption handler (IH) routes control to Recovery/Termination or to DSS (Dynamic Support System) after the operator hits the restart key on the console or a routine issues a Restart SIGP instruction.		
1 The restart IH ignores recursive entries by giving control back to the program executing when the restart interruption occurred. Otherwise, normal processing continues.	IEAVRSTR	IEAVRSTR
2 The restart IH saves current status in the LCCA.		
3 The restart IH handles any of three separate conditions; DSS active; DSS initialization; or R/TM processing.		
4 The restart IH will try to obtain the restart lock after determining that DSS is active by placing the CPUID and DF code into the lockword. If the lockword equals 0 or if the lockword already contains either the DSS code (DF) or R/TM code (RF), control goes to DSS at entry point IQARIH00. The DSS routine may return control to the restart IH at step 6 if it determines that R/TM should receive control. The restart IH will return control to the interrupted program if the restart lock could not be obtained.		

Diagram 19-12. Restart Interruption Handler (IEAVERES) (Part 3 of 4)

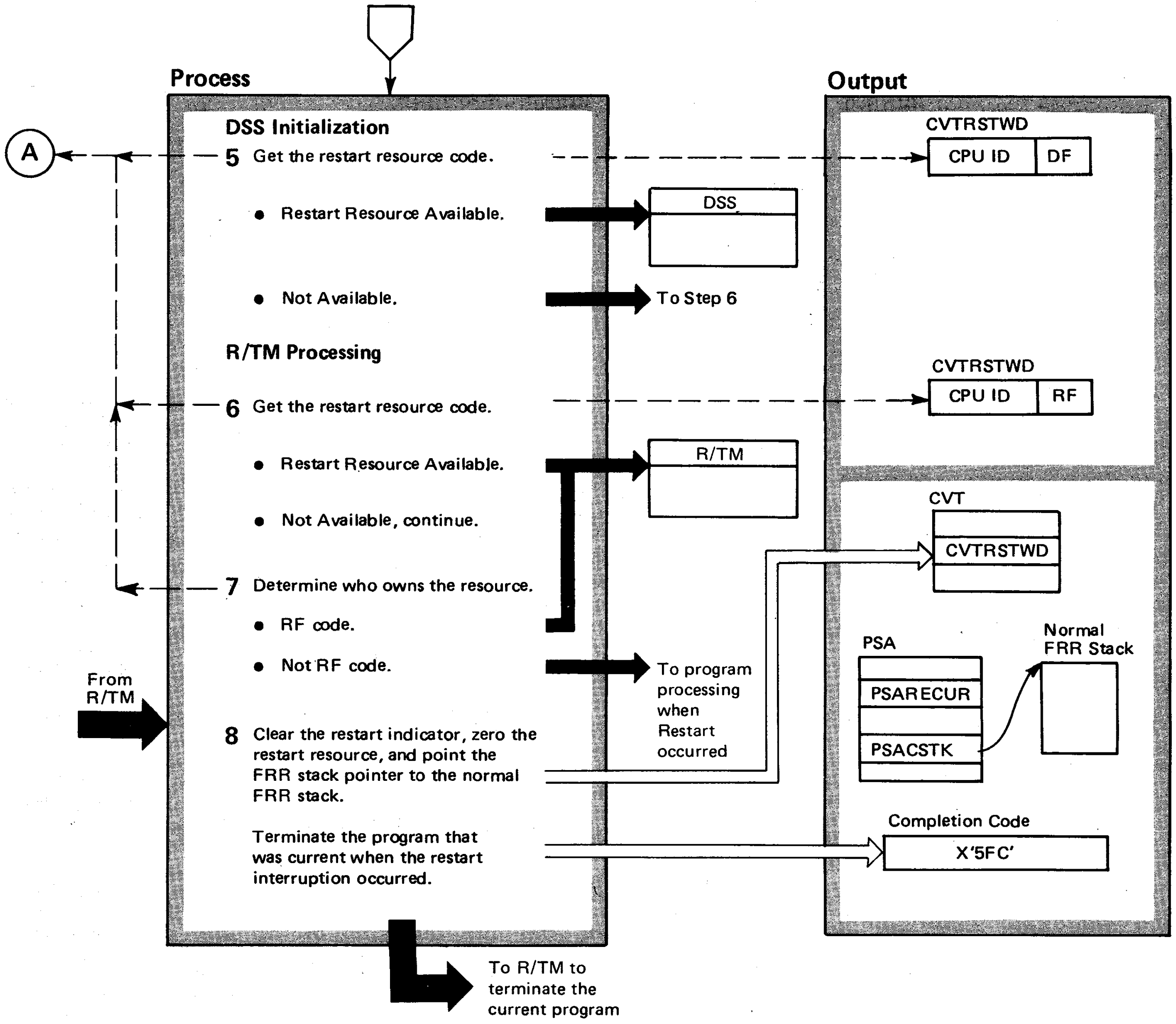

Process
DSS Initialization
5 Get the restart resource code.
● Restart Resource Available.
● Not Available.
R/TM Processing
6 Get the restart resource code.
● Restart Resource Available.
● Not Available, continue.
7 Determine who owns the resource.
● RF code.
● Not RF code.
From R/TM
8 Clear the restart indicator, zero the restart resource, and point the FRR stack pointer to the normal FRR stack.
Terminate the program that was current when the restart interruption occurred.
A
DSS
To Step 6
R/TM
To program processing when Restart occurred
To R/TM to terminate the current program
Output
CVTRSTWD
CPU ID
DF
CVTRSTWD
CPU ID
RF
CVT
CVTRSTWD
PSA
PSARECUR
PSACSTK
Normal FRR Stack
Completion Code
X'5FC'

Diagram 19-12. Restart Interruption Handler (IEAVERES) (Part 4 of 4)

Extended Description	Module	Label
5 If the operator (by setting manually the bit PSADSSGO) has requested that DSS be initialized, then the restart IH tries to obtain the restart lock, using the DF code. The restart IH passes control to DSS only if the lock initially equals 0. When the lock cannot be used, proceed at step 6.		
6 R/TM uses the restart interruption as a method of breaking program loops. The restart IH passes control to R/TM if the restart lock (CVTRSTWD) equals 0 or is already owned by R/TM.		
7 Control goes to R/TM if the resource code indicates current R/TM processing. Otherwise, control goes to the program executing at the time the restart interruption occurred.		
8 The restart FRR (functional recovery routine) clears the restart interruption indicator in PSARECUR, zeros the restart resource in the CVTRSTWD field of the CVT, 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 19-13. Signal Service Routines – IPC (IEAVERI) (Part 1 of 6)

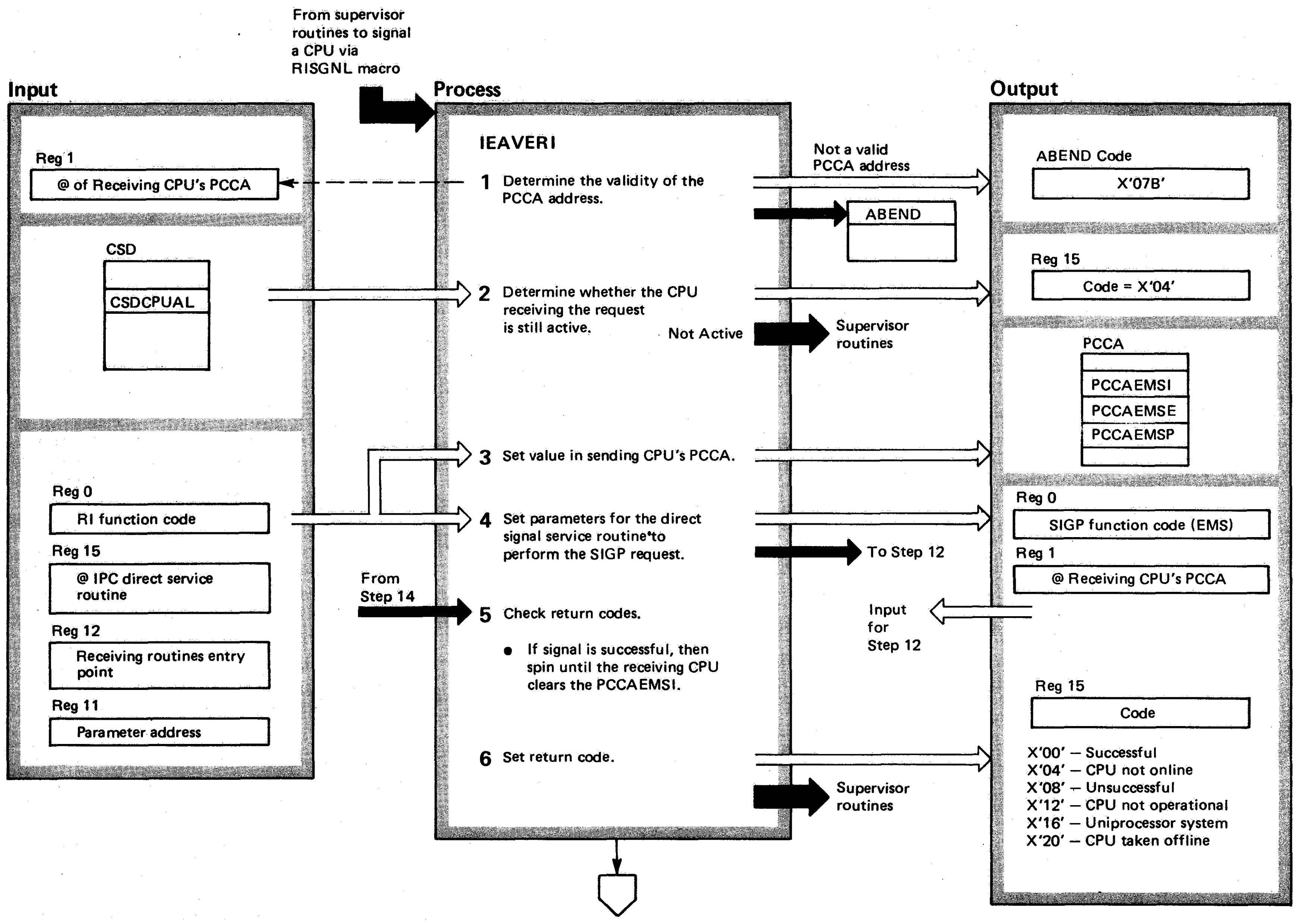

From supervisor routines to signal a CPU via RISGNL macro
Input
Process
Output
Reg 1
@ of Receiving CPU's PCCA
CSD
CSDCPUAL
Reg 0
RI function code
Reg 15
@ IPC direct service routine
Reg 12
Receiving routines entry point
Reg 11
Parameter address
IEAVERI
1 Determine the validity of the PCCA address.
2 Determine whether the CPU receiving the request is still active.
Not Active
3 Set value in sending CPU's PCCA.
4 Set parameters for the direct signal service routine to perform the SIGP request.
From Step 14
5 Check return codes.
• If signal is successful, then spin until the receiving CPU clears the PCCAEMSI.
6 Set return code.
Not a valid PCCA address
ABEND
Supervisor routines
To Step 12
Input for Step 12
Supervisor routines
ABEND Code
X'07B'
Reg 15
Code = X'04'
PCCA
PCCAEMSI
PCCAEMSE
PCCAEMSP
Reg 0
SIGP function code (EMS)
Reg 1
@ Receiving CPU's PCCA
Reg 15
Code
X'00' – Successful
X'04' – CPU not online
X'08' – Unsuccessful
X'12' – CPU not operational
X'16' – Uniprocessor system
X'20' – CPU taken offline

Diagram 19-13. Signal Service Routines – IPC (IEAVERI) (Part 2 of 6)

Extended Description	Module	Label
Signal service routines causes communication between the CPUs in a multiprocessor system. The signal service routines, combined with the emergency signal second level interruption handler (SLIH) and the external call SLIH, produce the new IPC (interprocessor communication) feature. The signal service routines provide the signal-issuing capability, while the two SLIHs provide the signal receiving and routing capability. The signal service routines consist of three functionally related modules:		

- IEAVERI – which performs the remote immediate signal (using emergency signal)
- IEAVERP – which performs the remote pendable signal (using external call)
- IEAVEDR – which performs the direct signal, and issues the SIGP (signal processor) instruction for IEAVERI and IEAVERP.

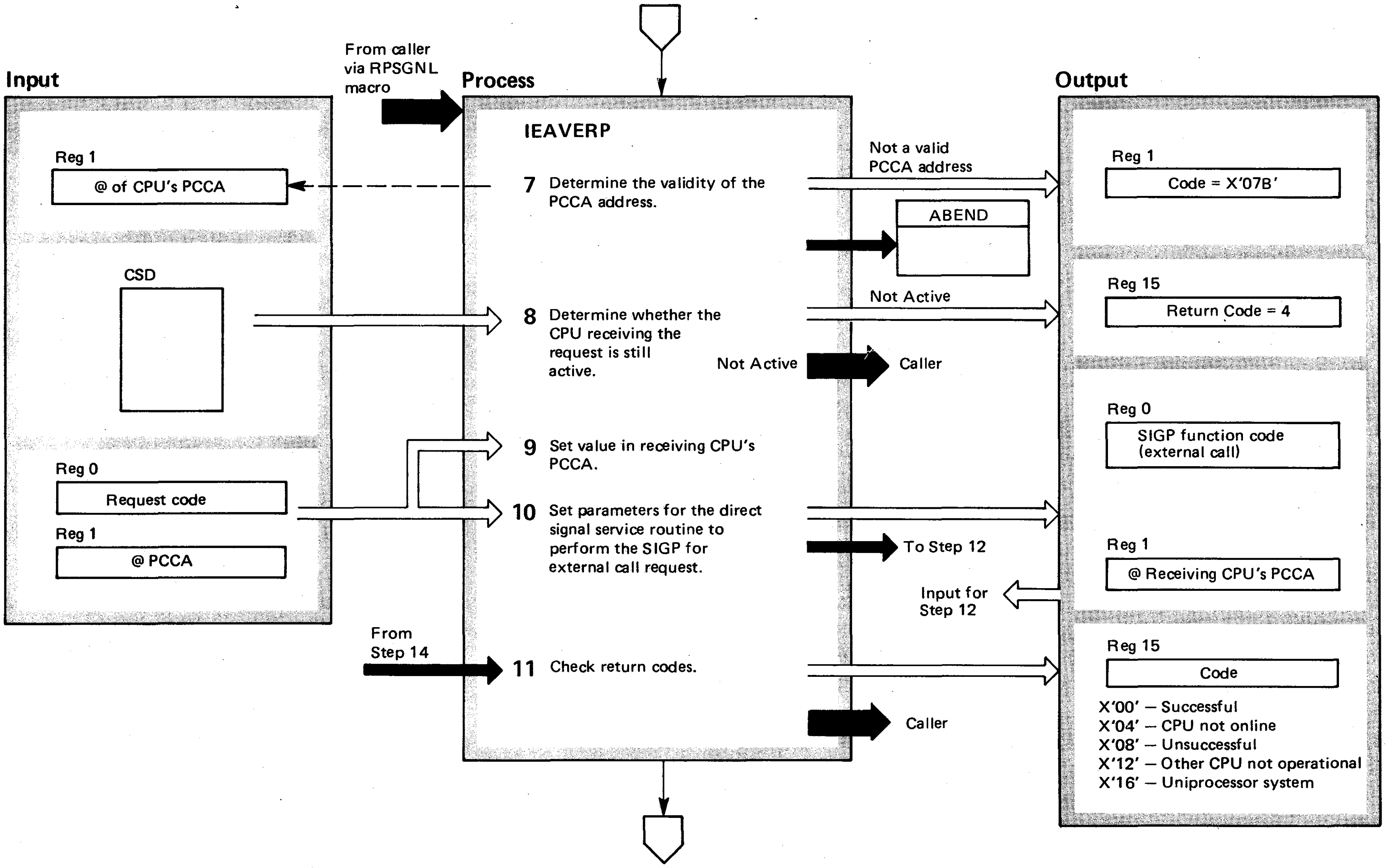
The SIGP instruction, issued by the direct signal routine, contains 12 functions:

- Start
- Stop
- Sense
- Program reset
- Initial program reset
- Stop and store status
- Initial microprogram load
- Initial CPU reset
- CPU Reset
- Restart
- Emergency signal
- External call

The publication *OS/VS2 System Programming Library: Supervisor,* GC28-0628, explains RISGNL, RPSGNL, and DSGNL instructions in detail. The publication *IBM System/370: Principles of Operation,* GA22-7000, expains the hardware signals explained above.

Extended Description	Module	Label
1 The remote immediate signal routine (part of the remote signal routines) performs the functions described in steps 1-6. First, it determines the validity of the PCCA (physical configuration communication area) address. This routine gives control to ABEND if it finds the PCCA address invalid. The caller receives a X'07B' ABEND code. Otherwise, normal processing follows.	IEAVERI	IEAVERI
2 Control returns to the caller if the receiving CPU is not online, with a return code of 4 in register 15.		
3 Next, the remote immediate routine sets the function code, entry point address, and parameter address in the PCCAEMSB field of the sending CPU's PCCA.		
4 The remote immediate routine sets the input values for the direct signal routine, which actually issues the SIGP instruction.		
5 The direct signal routine (steps 12-14) sets a return code (see step 14) and passes this code back to the caller. The remote immediate routine checks this code. For serial or parallel requests, if the signal was successful, the remote immediate routine spins until the receiving CPU clears the PCCAEMSI field.		
6 The caller receives a return code, indicating the status of the request, from the remote immediate routine.		

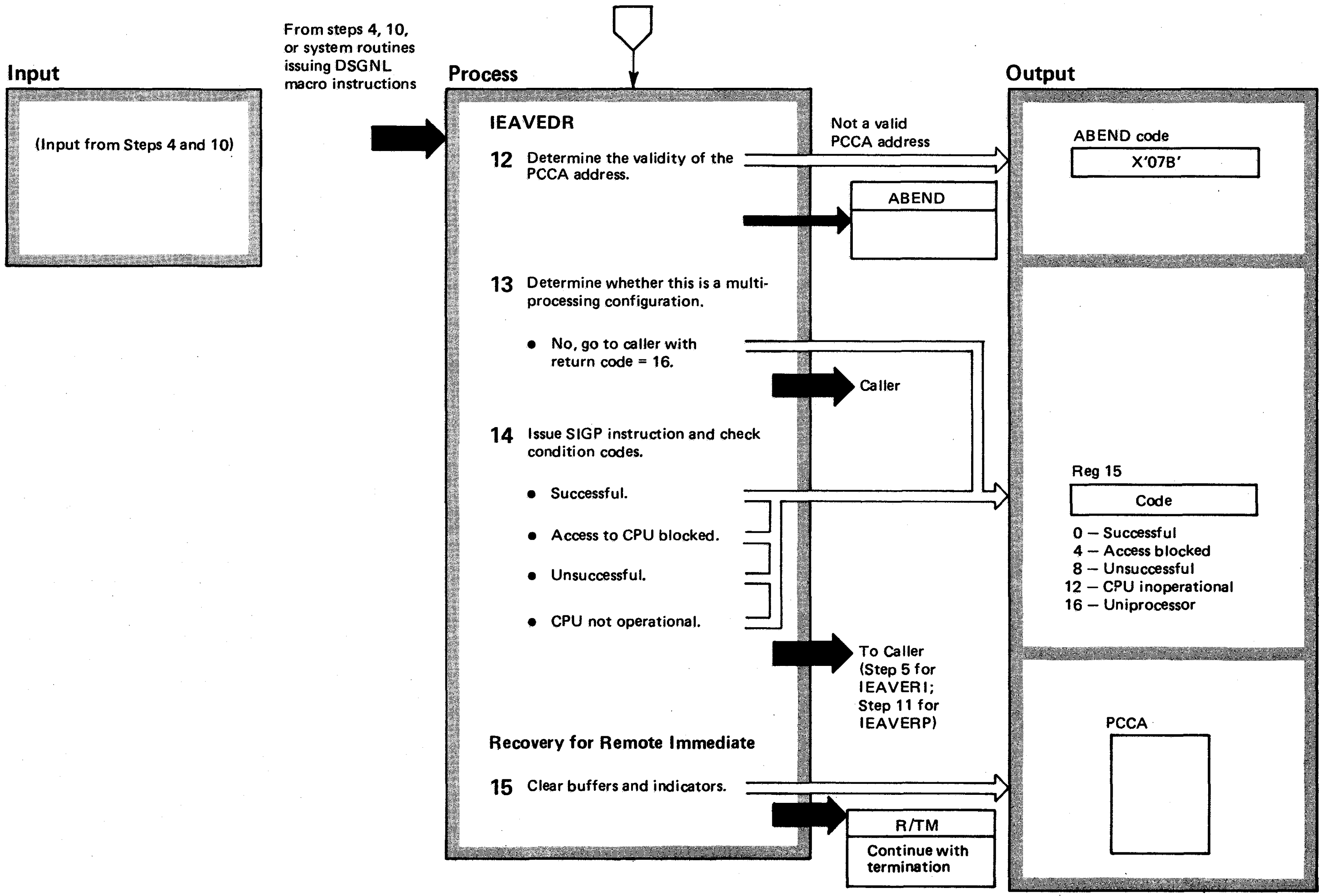
Diagram 19-13. Signal Service Routines – IPC (IEAVERP) (Part 3 of 6)



Input
Process
Output
From caller via RPSGNL macro
IEAVERP
Reg 1
@ of CPU's PCCA
7 Determine the validity of the PCCA address.
Not a valid PCCA address
Reg 1
Code = X'07B'
ABEND
CSD
8 Determine whether the CPU receiving the request is still active.
Not Active
Reg 15
Return Code = 4
Not Active
Caller
Reg 0
SIGP function code (external call)
9 Set value in receiving CPU's PCCA.
Reg 0
Request code
Reg 1
@ PCCA
10 Set parameters for the direct signal service routine to perform the SIGP for external call request.
To Step 12
Reg 1
@ Receiving CPU's PCCA
Input for Step 12
From Step 14
11 Check return codes.
Reg 15
Code
Caller
X'00' – Successful
X'04' – CPU not online
X'08' – Unsuccessful
X'12' – Other CPU not operational
X'16' – Uniprocessor system

Extended Description	Module	Label
7 The remote pendable signal routine (part of the remote signal routines) performs the functions described in steps 7-11. First, it determines the validity of the PCCA address. The remote pendable routine gives control to ABEND if it finds the PCCA address invalid. The caller receives a X'07B' ABEND code. Otherwise, normal processing follows.	IEAVERP	IEAVERP
8 The remote pendable routine determines whether the CPU receiving the request is still active, since it could have stopped processing. Control goes back to the caller, with a return code of 4 in register 15, if the receiving CPU is not active. Otherwise, normal processing continues.		
9 Next, the remote pendable routine sets the function code in the PCCARPB field of the receiving CPU's PCCA.		
10 The remote signal routines set the input values for the direct signal routine, which actually issues the SIGP instruction for an external call request.		
11 The remote signal routine checks the return codes, and returns to the caller with a code in register 15 indicating status of the request.		

Diagram 19-13. Signal Service Routines – IPC (IEAVEDR) (Part 5 of 6)



IEAVEDR

12 Determine the validity of the PCCA address.

13 Determine whether this is a multiprocessing configuration.

- No, go to caller with return code = 16.

14 Issue SIGP instruction and check condition codes.

- Successful.
- Access to CPU blocked.
- Unsuccessful.
- CPU not operational.

Recovery for Remote Immediate

15 Clear buffers and indicators.

Diagram 19-13. Signal Service Routines – IPC (IEAVEDR) (Part 6 of 6)

Extended Description	Module	Label
12 The direct signal routine checks the validity of the PCCA address, and gives control to ABEND for an invalid address.	IEAVEDR	
13 The direct signal routine checks to see if this is a multiprocessing configuration. If it is not, control goes back to the caller, with a return code of 16 in register 15.		
14 The direct signal routine issues the SIGP instruction and receives a condition code. Control then returns to the caller.		
15 The signal services FRR (functional recovery routine) handles errors occurring during the RISGNL sending processing (module IEAVERI). The signal services FRR ensures that recovery occurs on the same CPU that the error occurred. The signal services FRR then clears the EMS buffer in the PCCA, clears the super bit, PSAIPCRI, and clears the spin bit, LCCASIGP. Control returns to R/TM, which subsequently gives control to the caller's error recovery routine.	IEAVEIPR	

Diagram 19-14. External Call Second Level Interruption Handler (IEAVEXS) (Part 1 of 2)

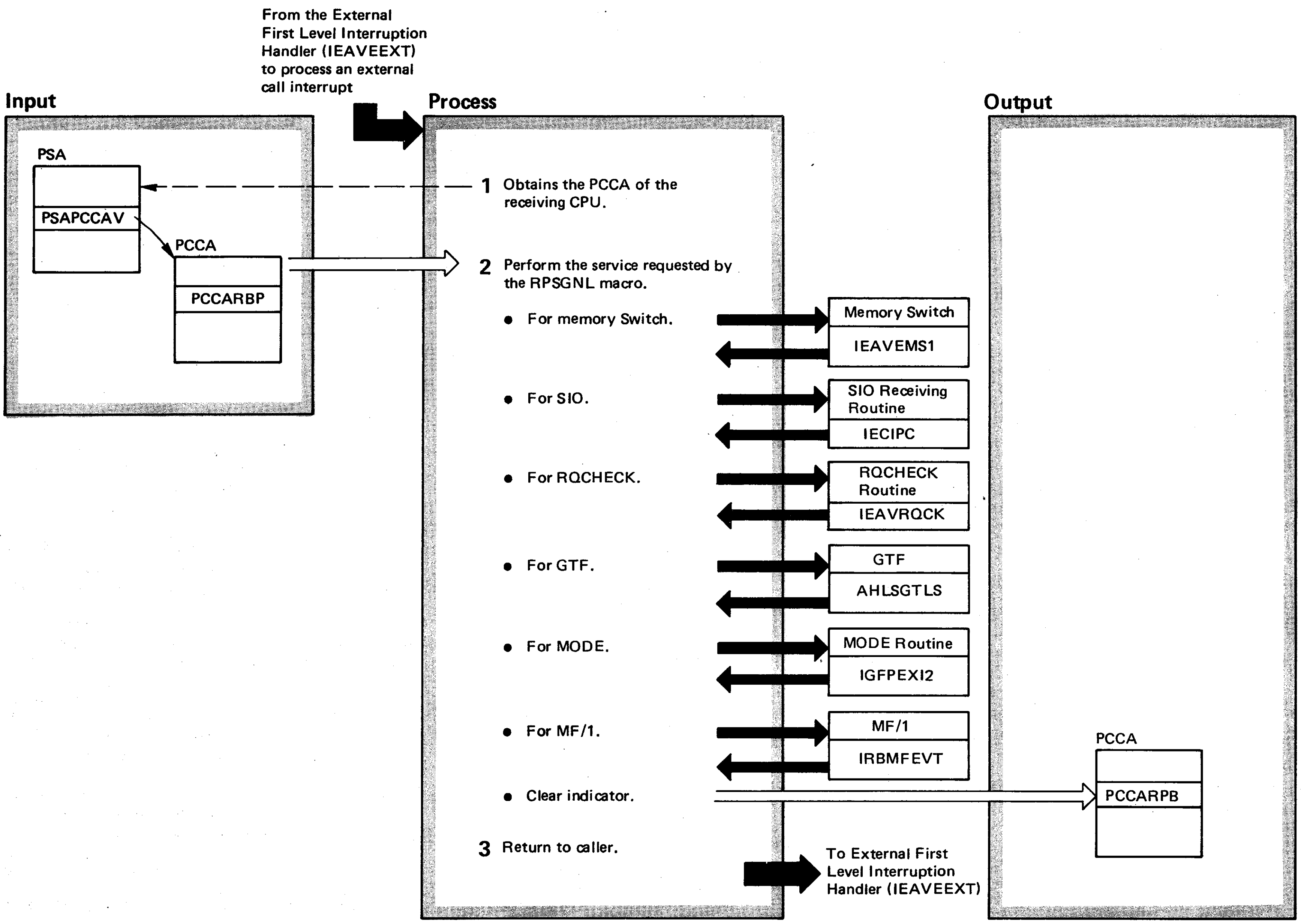


Diagram 19-14. External Call Second Level Interruption Handler (IEAVEXS) (Part 2 of 2)

Extended Description	Module	Label
The external call second level interruption handler (SLIH) routes control to any of six service routines requested by the RPSGNL (remote pendable) function. The external call SLIH can pass control to these routines: ● Memory switch routine. ● Start I/O receiving routine. ● RQCHECK routine. ● Generalized Trace Facility routine (GTF). ● MODE routine. ● Measurement Facility (MF/1). Control returns to the external call SLIH from these routines; the external call SLIH returns control to the external call FLIH.		

Extended Description	Module	Label
1 The external call SLIH locates the PCCA (physical control communications area) of the CPU executing by referring to the PSA (prefixed storage area).	IEAVEXS	IEAVEXS
2 The PCCA contains an indicator, in the remote pendable buffer (PCCARPB), of the service requested in the RPSGNL function. The external call SLIH checks the PCCARPB field, sequentially for each possible condition, to determine which services should receive control. In each case, the external SLIH: ● Determines the actions requested in the RPSGNL function. ● Turns the indicator in the PCCARPB off. ● Branches to the appropriate service routine. ● Double-checks the PCCARPB to ensure that no new requests have occurred during the previous processing.		
3 Control returns to the external call FLIH.		

Diagram 19-15. Emergency Signal Second Level Interruption Handler (IEAVEES) (Part 1 of 2)

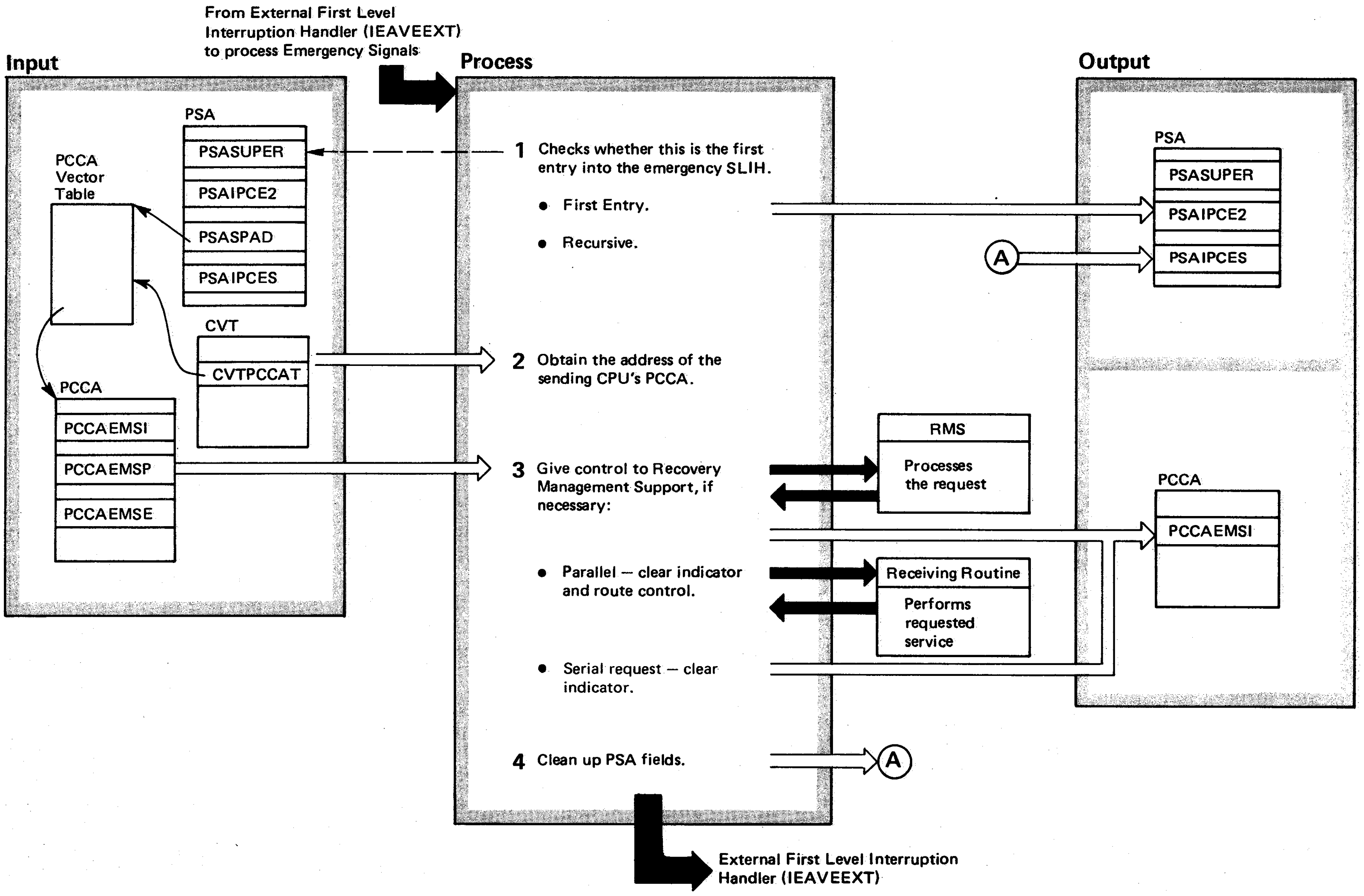

From External First Level Interruption Handler (IEAVEEXT) to process Emergency Signals
Input
Process
Output
PSA
PSASUPER
PSAIPCE2
PSASPAD
PSAIPCES
PCCA Vector Table
CVT
CVTPCCAT
PCCA
PCCAEMSI
PCCAEMSP
PCCAEMSE
1 Checks whether this is the first entry into the emergency SLIH.
• First Entry.
• Recursive.
2 Obtain the address of the sending CPU's PCCA.
3 Give control to Recovery Management Support, if necessary:
• Parallel — clear indicator and route control.
• Serial request — clear indicator.
4 Clean up PSA fields.
A
RMS
Processes the request
Receiving Routine
Performs requested service
PSA
PSASUPER
PSAIPCE2
PSAIPCES
PCCA
PCCAEMSI
External First Level Interruption Handler (IEAVEEXT)

Diagram 19-15. Emergency Signal Second Level Interruption Handler (IEAVEES) (Part 2 of 2)

Extended Description	Module	Label
The emergency signal second level interruption handler (SLIH) receives control from the external FLIH and routes control to a specified receiving routine to process an emergency signal (EMS).		

The emergency signal SLIH handles these types of requests:

- RMS (recovery management support)
- Serial
- Parallel

For RMS requests, the emergency signal SLIH branches to the RMS service routine. For serial requests, the emergency signal SLIH turns off an indicator bit *after* receiving control back from the specified receiving routine; for parallel requests, the emergency signal SLIH turns off an indicator bit *before* it branches to the specified receiving routine. Control always returns to the emergency signal SLIH.

Extended Description	Module	Label
1 The emergency signal SLIH checks the PSA SUPER bit to determine whether this is a recursive entry, and then indicates the type — either first entry or recursive entry — in the same PSASUPER field.	IEAVEES	
2 The emergency signal SLIH indexes into the PCCA vector table, using the PSASPAD, to obtain the sending CPU's PCCA.		
3 At this point, the emergency signal SLIH processes RMS, serial, or parallel requests. Control goes to RMS, to process the request using the address in a VCON. To process serial or parallel requests, the emergency signal SLIH obtains the entry points for the specified receiving routine from the PCCAEMSE field of the sending CPU's PCCA. The emergency signal SLIH clears the PCCAEMSI indicator to allow the sending CPU to proceed.		
4 The emergency signal SLIH cleans up PSA fields, and returns to the external FLIH.		

Diagram 19-16. Stage 1 Exit Effector (IEAVEF00) (Part 1 of 2)

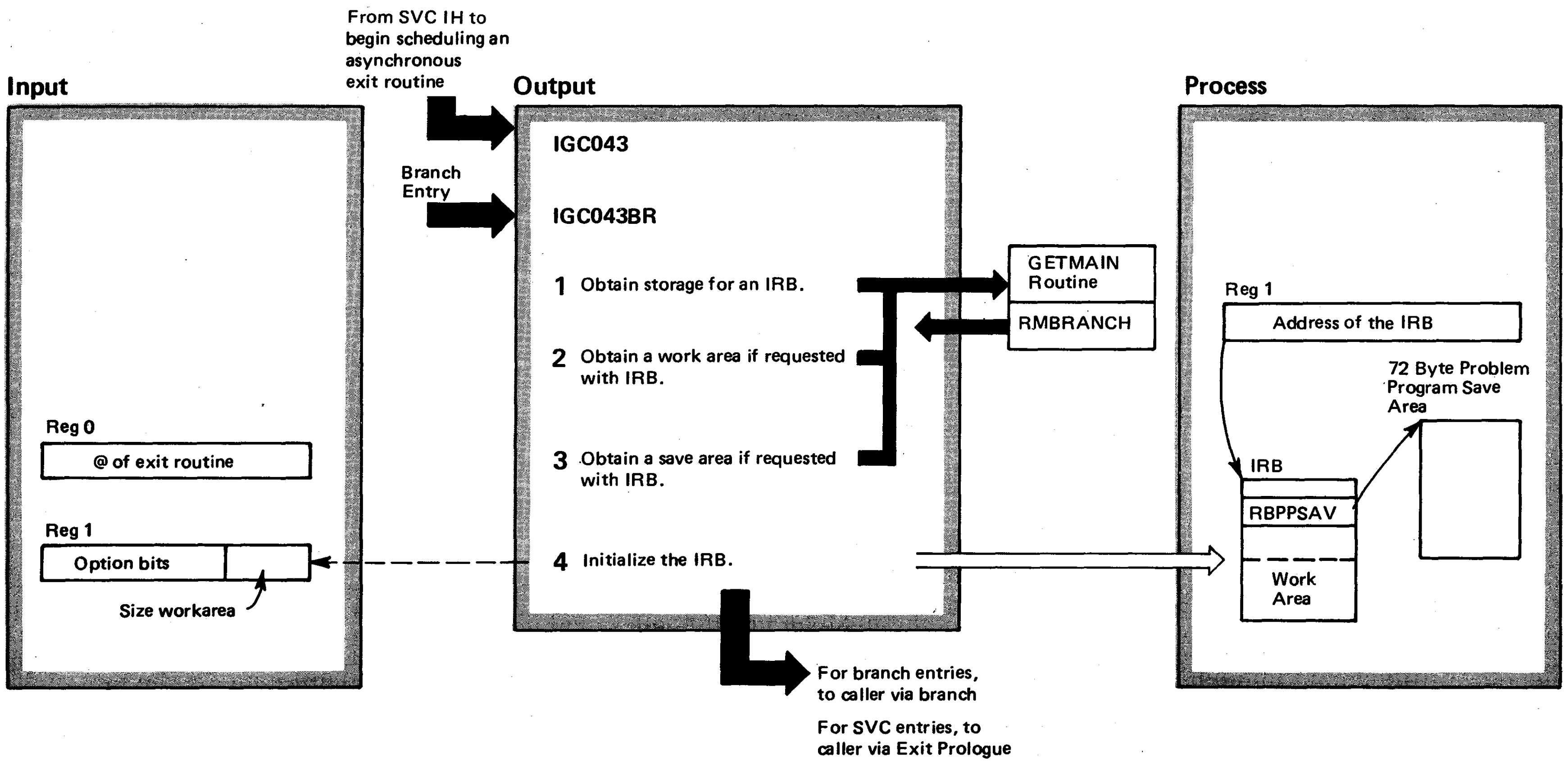
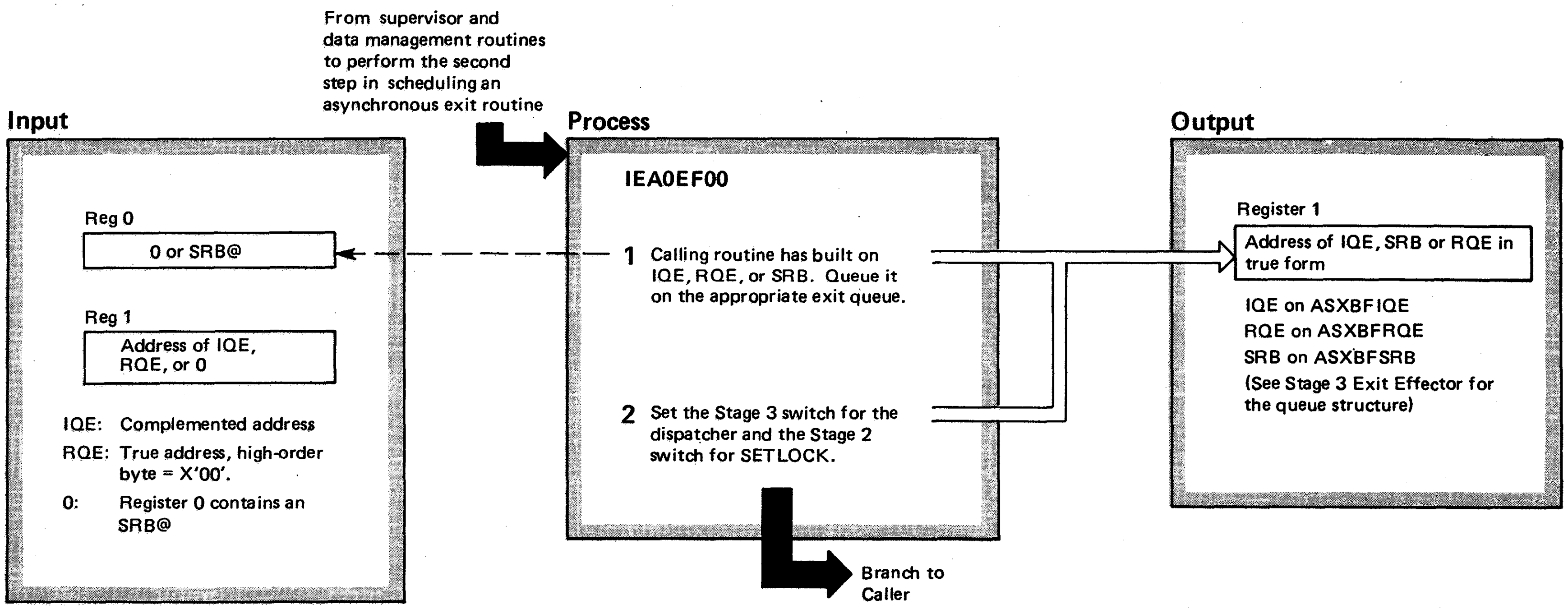

From SVC IH to begin scheduling an asynchronous exit routine
Input
Output
Process
IGC043
Branch Entry
IGC043BR
1 Obtain storage for an IRB.
2 Obtain a work area if requested with IRB.
3 Obtain a save area if requested with IRB.
4 Initialize the IRB.
GETMAIN Routine
RMBRANCH
Reg 0
@ of exit routine
Reg 1
Option bits
Size workarea
Reg 1
Address of the 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

Diagram 19-16. Stage 1 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 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 19-17. 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
Reg 0
0 or SRB@
Reg 1
Address of IQE, RQE, or 0
IQE: Complemented address
RQE: True address, high-order byte = X'00'.
0: Register 0 contains an SRB@
Process
IEA0EF00
1 Calling routine has built on 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.
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 19-17. Stage 2 Exit Effector (IEAVEEE2) (Part 2 of 2)

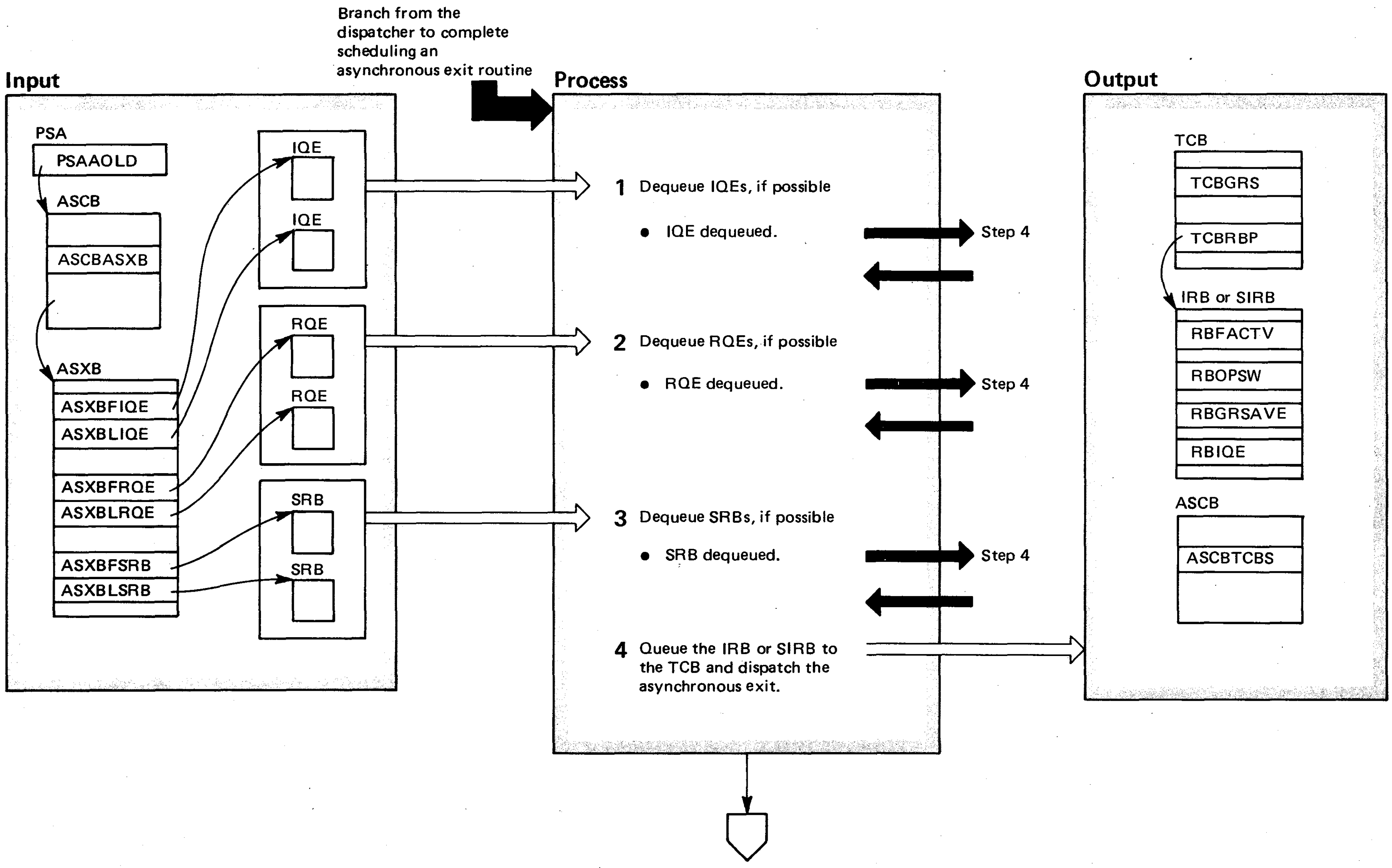
Extended Description	Module	Label
1 The exit queue on which the Stage 2 Exit Effector places the input queue element depends on whether the queue element is an IQE (interruption queue element), an RQE (request queue element), or an SRB (service request block).	IEAVEEE2	IEA0EF00

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 This indicates 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.

Diagram 19-18. Stage 3 Exit Effector (IEAVEEE0) (Part 1 of 4)



Extended Description	Module	Label
The stage 3 exit effector is the last routine used to schedule an asynchronous exit. The stage 3 exit effector dequeues IQEs (interruption queue elements), RQEs (request queue elements) or SRBs for asynchronous exit queues pointed to by the ASCB. 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.	IEAVEEE0	

For each IQE on the asynchronous exit queue, stage 3 exit effector does the following:

- It will first determine if the IQE can be dequeued at this time. An IQE will not 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 CPU.
  - D. The asynchronous exit is being scheduled to the error task and an error recovery procedure is in process on that task.
  - E. Asynchronous exits have been suppressed for the intended task (TCBFX=1).
  - F. This is an attention exit being scheduled and either all asynchronous exits or attention exits are suppressed (TCBFX=1 or TCBATT=1) for the intended task or any of the task's descendants in the task tree.
  - G. The transfer control function is in process for the TCB that stage 3 is checking (TCBS3A=1). If this flag is not on, it is turned on by stage 3.
- For all IQEs that can be dequeued, the IQE will be removed from the queue, and the IRB associated with the IQE will be enqueued to the specified TCB.

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 will be made to determine if it can be dequeued at this time. It will not be dequeued if:

A. Asynchronous exits are suppressed for the task (TCBFX=1).

B. The task it is being scheduled to is active on another CPU.

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.
For those RQEs that may be dequeued, the RQE will be removed from the queue, and the specified IRB will be enqueued to the TCB.

E. The transfer control function is in process for the TCB that stage 3 is checking (TCBS3A=1). If this flag is not on, it is turned on by stage 3.

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, and stage 3 exit effector will try to schedule this SIRB for only the top SRB on the queue. The SIRB will not be scheduled if the error task is already executing on another CPU, or if an error recovery procedure is in process in that address space. The transfer control function is in process for the TCB that stage 3 is checking (TCBS3A=1). If this flag is not on, it is turned on by stage 3. If the ERP can be scheduled at this time, the top SRB will be enqueued to the task specified as the error task in that address space.

4 In order to schedule the asynchronous routine, stage 3 exit effector must do the following processing:

A. The IRB must be placed on the RB chain of the specified task. The IRB becomes the current RB for that task.

B. The saved registers of the previously current routine are moved from the TCB to the IRB General Register save area.

C. The IRB is marked active (RBFACTV=1) so that any other requests for use of the same IRB will be deferred.

D. The address portion of the RBOPSW is set to the address specified in the RBEP field. This ensures that the dispatcher gives control to the asynchronous routine at the specified entry point.

E. The RBIQE is set to point to the queue element that scheduled the asynchronous routine (IQE, RQE, or SRB) area so that the asynchronous exit gets control with specific register contents.

F. If this task has been made ready and it previously was not, the count of ready TCBs (ASCBTCBS) is incremented by one.

G. Registers are initialized in the TCB to set up for entry to the asynchronous exit.

H. Stage 3/TCTL intersect flag is turned off (TCBS3A=0).

Diagram 19-18. Stage 3 Exit Effector (IEAVEEE0) (Part 3 of 4)

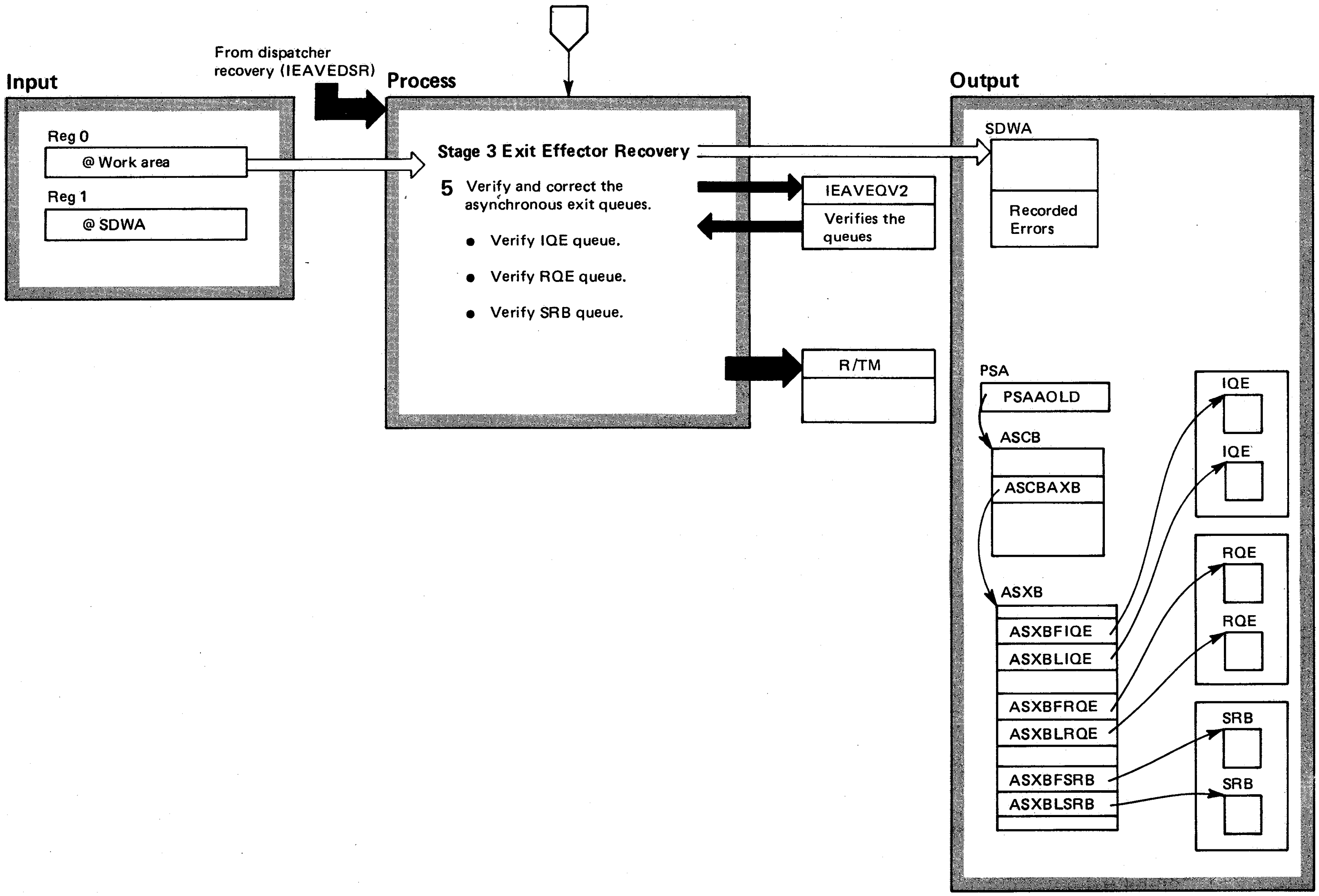
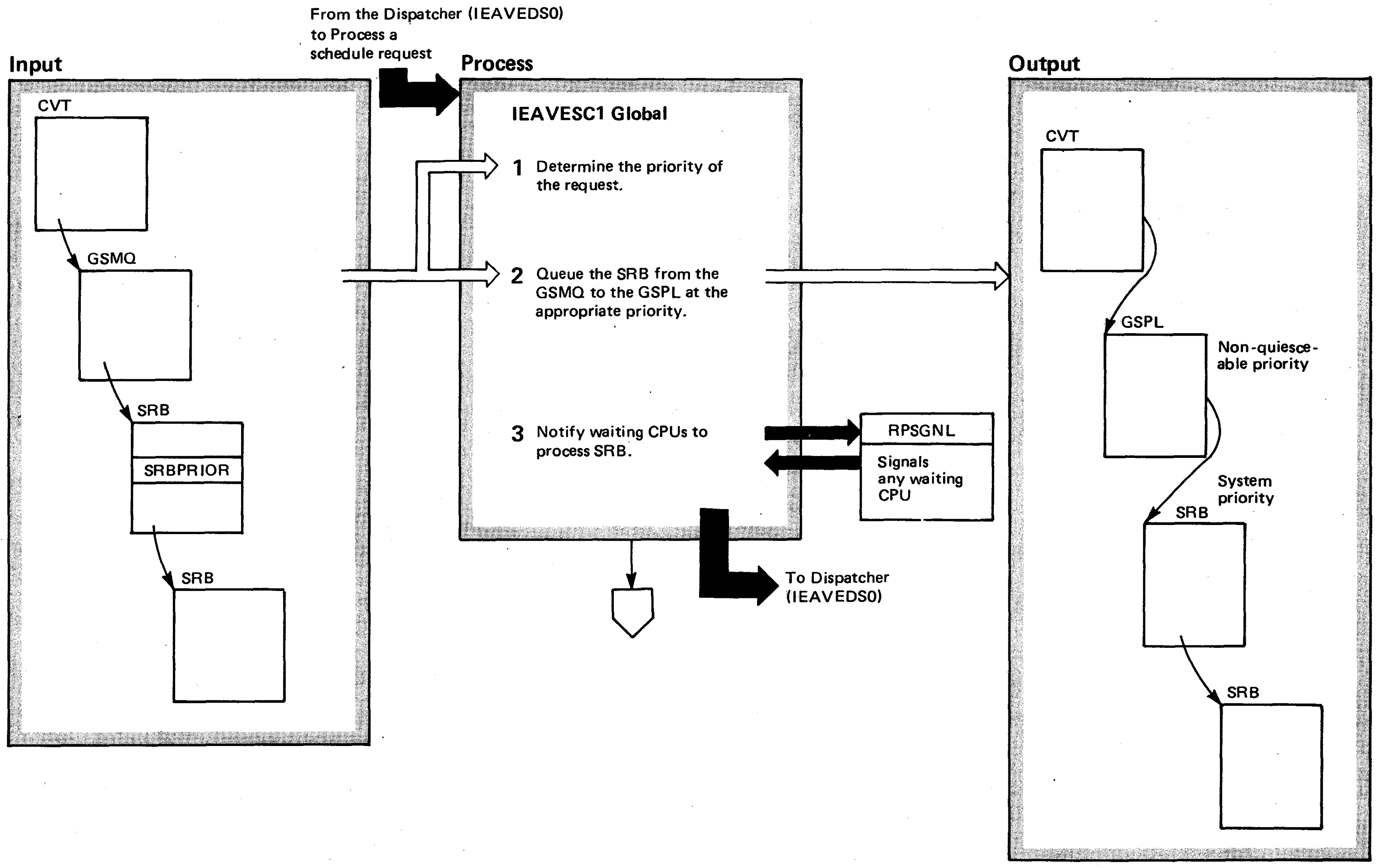


Diagram 19-18. Stage 3 Exit Effector (IEAVEEE0) (Part 4 of 4)

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, and RQE queue, and an SRB queue). It uses the Queue Verifier (IEAVEQV0) to perform this verification. It calls the routine three times, once for each queue. After each call, it will store a word of zeroes 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.	IEAVEEER	IEAVEEER

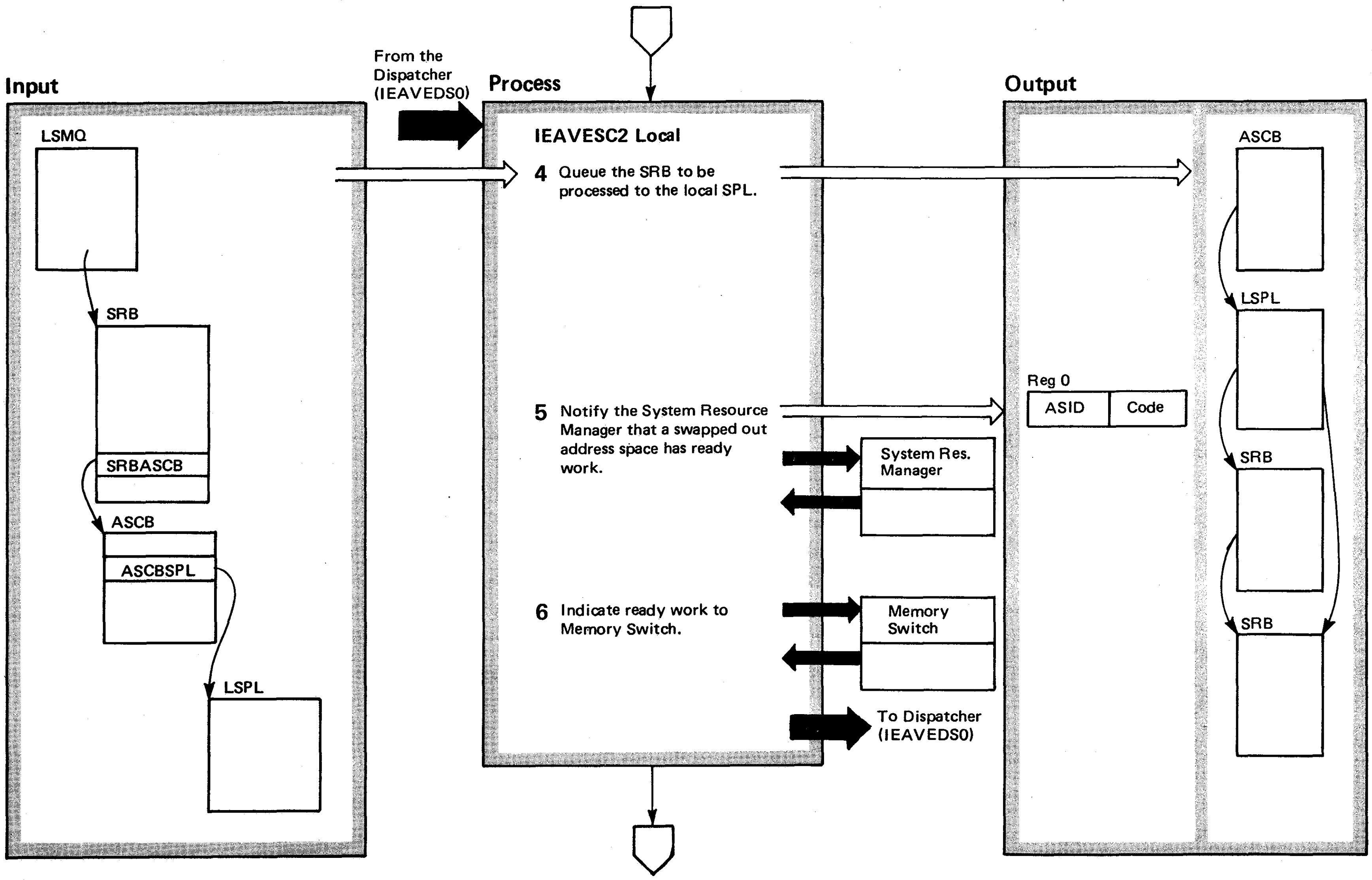
Diagram 19-19. SCHEDULE Processing (IEAVESC0) (Part 1 of 6)



From the Dispatcher (IEAVEDS0) to Process a schedule request
Input
CVT
GSMQ
SRB
SRBPRIOR
SRB
Process
IEAVESC1 Global
1 Determine the priority of the request.
2 Queue the SRB from the GSMQ to the GSPL at the appropriate priority.
3 Notify waiting CPUs to process SRB.
RPSGNL
Signals any waiting CPU
To Dispatcher (IEAVEDS0)
Output
CVT
GSPL
Non-quiesce-able priority
System priority
SRB
SRB

Extended Description	Module	Label
The Schedule service allows the requester to schedule system services. These system services can be scheduled to execute in any address space at either global or local priorities. System services scheduled at the global priority have a priority higher than that of the address space; those scheduled at a local priority have a priority higher than any task in the address space.		
The Schedule routine has two entry points – one for local priorities, one for global priorities.		
1 Schedule determines the address of the specific priority level. Schedule indexes by the value in SRBPRIOR into a table which contains the address of the specific level (Global Priority Index Table).	IEAVESC0	IEAVESC1
2 Schedule queues the SRB from the GSMQ (global service management queue) to the GSPL (global service priority list) in FIFO order.		
3 Schedule tests for CPUs dispatched to the wait task. Waiting CPUs will be activated to dispatch the SRBs on the GSPL. This will be tested by checking the count of CPUs dispatched to tasks, ASCBCPUS, in the wait ASCB. Schedule signals waiting CPUs (via RPSGNL, using the external call) forcing an entry to the dispatcher.		

Diagram 19-19. SCHEDULE Processing (IEAVESC0) (Part 3 of 6)



4 Queue the SRB to be processed to the local SPL.

5 Notify the System Resource Manager that a swapped out address space has ready work.

6 Indicate ready work to Memory Switch.

Diagram 19-19. SCHEDULE Processing (IEAVESCO) (Part 4 of 6)

Extended Description	Module	Label
4 Schedule locates the local SPL via ASCBSPL, from the ASCB indicated in SRBASCB. Schedule locates the priority level in the SPL by indexing by the value of SRBPRIOR into the Local Priority Index Table (LPIT) assembled in the schedule routine. Schedule queues the SRB to the requested priority level at the end of the queue.		IEAVESC2
5 Schedule notifies SRM (system resource manager) of work ready to be dispatched to an address space already swapped out. This will cause an eventual swap-in of that address space. Schedule also notifies the timer supervisor, by turning off ASCBTMLW, that the address space is no longer in a long wait.		
6 Schedule calls memory switch to determine whether the ready address space has a higher priority than the current address space. Schedule will indicate if the SRB has CPU affinity, if necessary, by sending a complemented value in register 1.		

Diagram 19-19. SCHEDULE Processing (IEAVESC0) (Part 5 of 6)

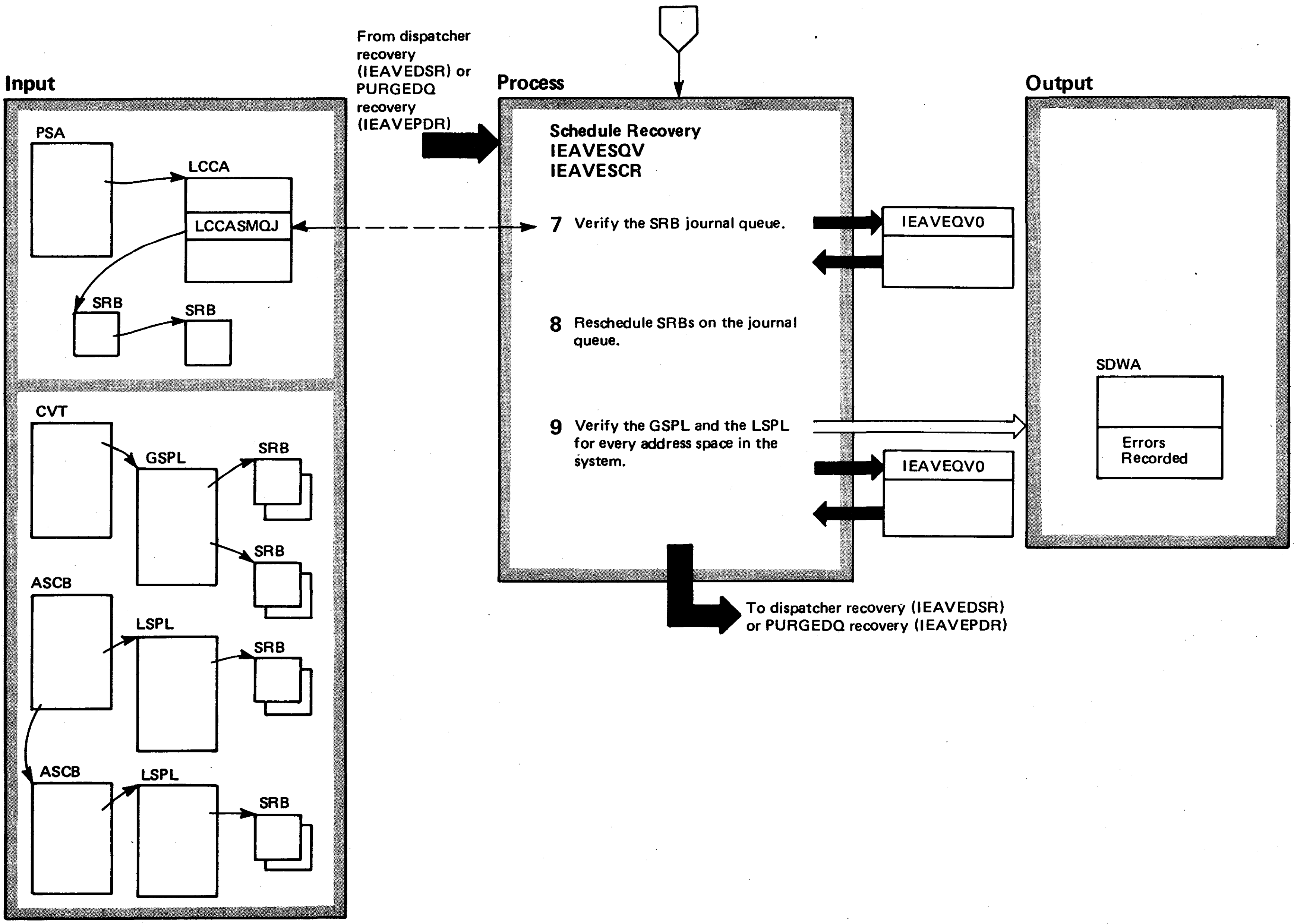
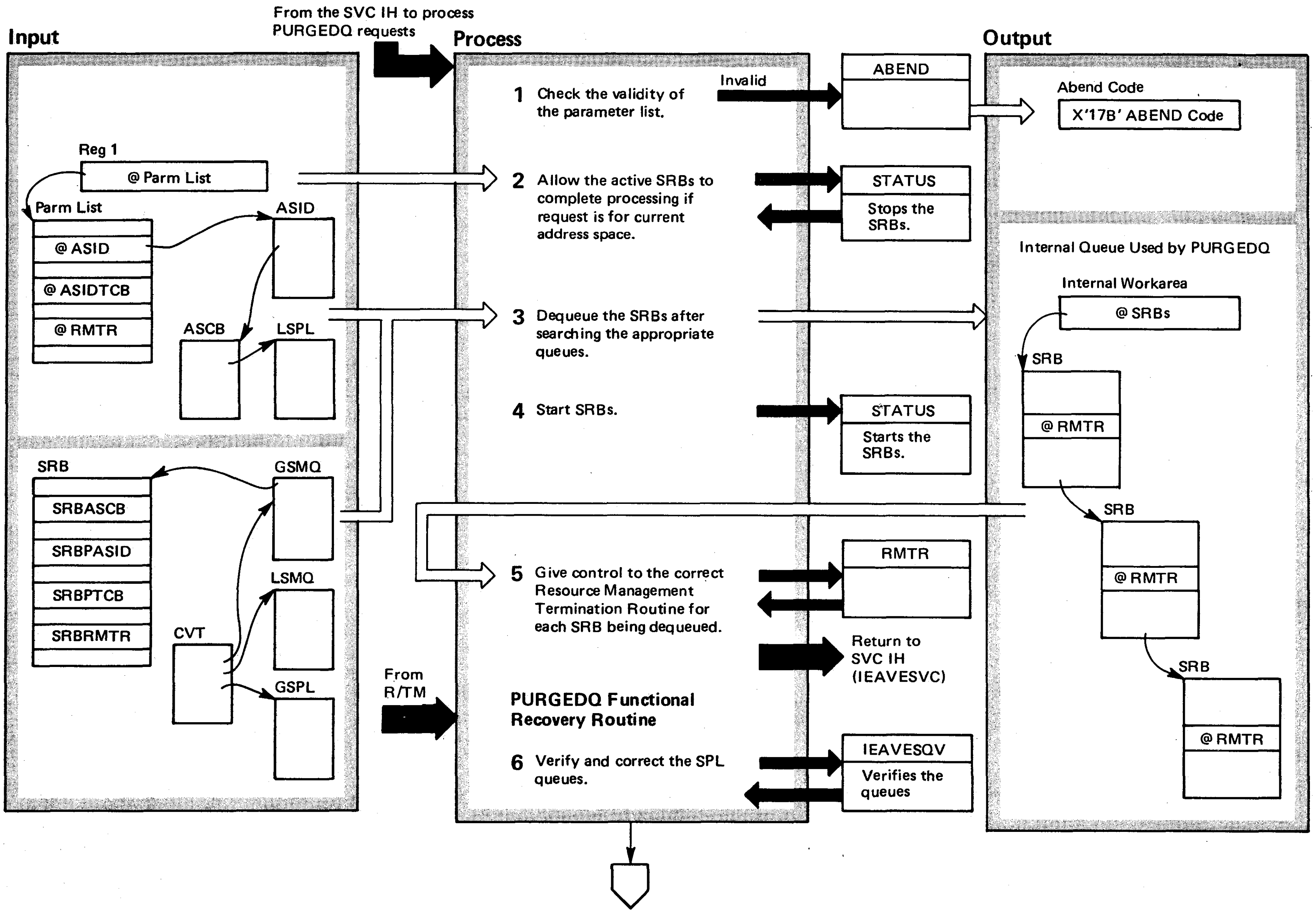

Input
PSA
LCCA
LCCASMQJ
SRB
SRB
CVT
GSPL
SRB
SRB
ASCB
LSPL
SRB
ASCB
LSPL
SRB
From dispatcher recovery (IEAVEDSR) or PURGEDQ recovery (IEAVEPDR)
Process
Schedule Recovery
IEAVESQV
IEAVESCR
7 Verify the SRB journal queue.
8 Reschedule SRBs on the journal queue.
9 Verify the GSPL and the LSPL for every address space in the system.
IEAVEQV0
IEAVEQV0
To dispatcher recovery (IEAVEDSR) or PURGEDQ recovery (IEAVEPDR)
Output
SDWA
Errors Recorded

Diagram 19-19. SCHEDULE Processing (IEAVESC0) (Part 6 of 6)

Extended Description	Module	Label
7 The Schedule FRR verifies the SRB journal queue, which is anchored out of LCCASMQJ field and removes SRBs with bad information. The journal queue is used by Schedule to prevent losing SRBs that are being processed.		
8 Schedule FRR then re-schedules any SRBs remaining on the journal queue.		
9 The Schedule FRR uses the Queue Verifier to verify SRB queues – the GSPL, 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 19-20. PURGEDQ Processing (IEAVEPD0) (Part 1 of 4)



From the SVC IH to process PURGEDQ requests
Input
Process
Output
Reg 1
@ Parm List
Parm List
@ ASID
@ ASIDTCB
@ RMTR
ASID
ASCB
LSPL
SRB
SRBASCB
SRBPASID
SRBPTCB
SRBRMTR
GSMQ
LSMQ
CVT
GSPL
1 Check the validity of the parameter list.
Invalid
ABEND
Abend Code
X'17B' ABEND Code
2 Allow the active SRBs to complete processing if request is for current address space.
STATUS
Stops the SRBs.
3 Dequeue the SRBs after searching the appropriate queues.
Internal Queue Used by PURGEDQ
Internal Workarea
@ SRBs
SRB
@ RMTR
4 Start SRBs.
STATUS
Starts the SRBs.
5 Give control to the correct Resource Management Termination Routine for each SRB being dequeued.
RMTR
Return to SVC IH (IEAVESVC)
From R/TM
PURGEDQ Functional Recovery Routine
6 Verify and correct the SPL queues.
IEAVESQV
Verifies the queues

Diagram 19-20. PURGEDQ Processing (IEAVEPD0) (Part 2 of 4)

Extended Description	Module	Label
Supervisor services use PURGEDQ to cancel SRBs that, for various reasons, should not be executed. The schedule routine queues SRBs on a queue; and these SRBs execute asynchronously to the schedule request. PURGEDQ cancels SRBs, when necessary.		
1 PURGEDQ terminates callers with 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) • Local Service Management Queue (IEALSMQ) • Global SPL (IEAGSPL) • The local SPL for the address space specified in the 'ASID' parameter. b. Scanning the queues searching for a match on the specified inputs. c. Dequeuing those SRBs that match the inputs.		
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 to the caller.		
6 PURGEDQ enters its FRR if an error occurred during the queue scanning or updating portion 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

Diagram 19-20. PURGEDQ Processing (IEAVEPD0) (Part 3 of 4)

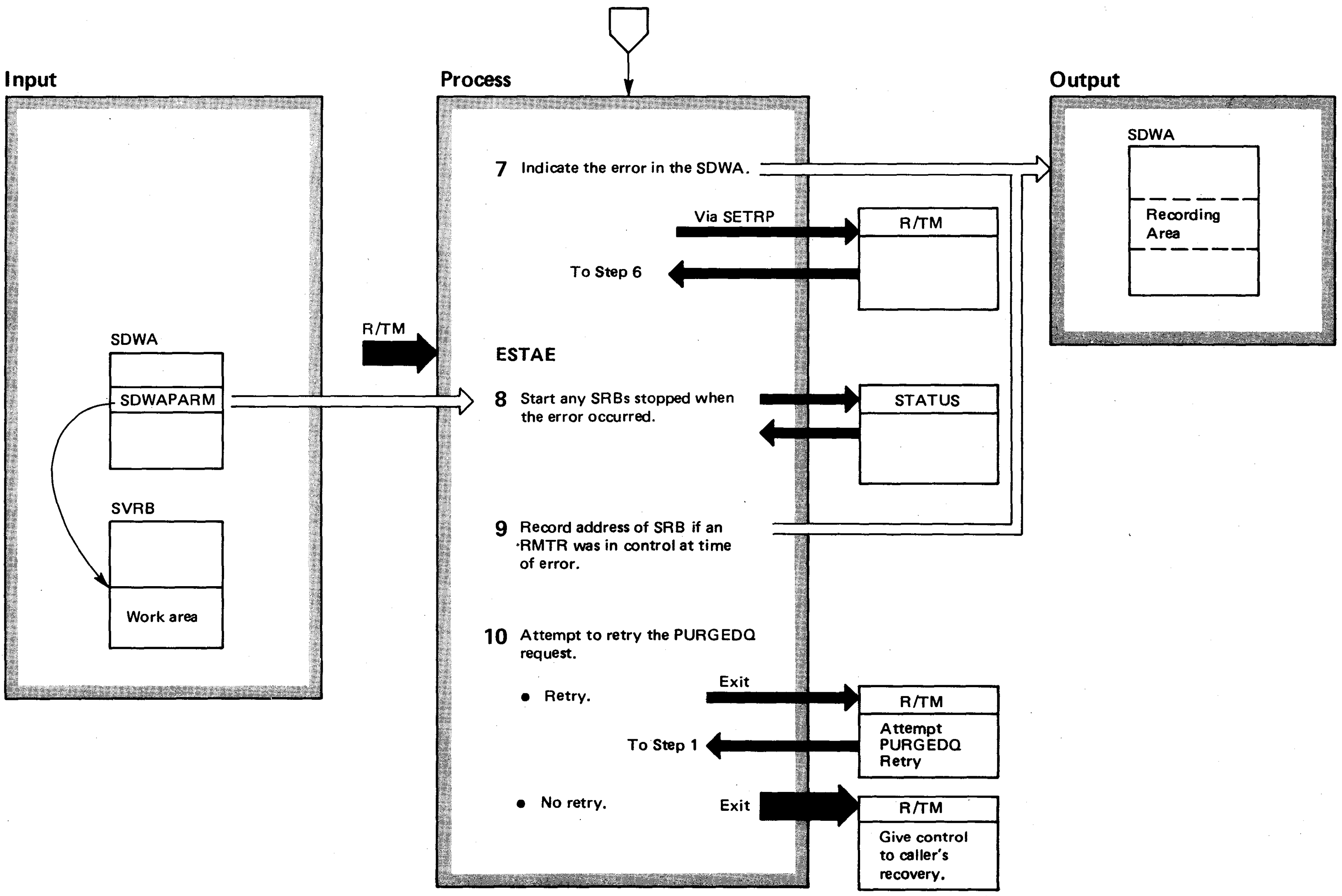

Input
Process
Output
SDWA
SDWAPARM
SVRB
Work area
R/TM
7 Indicate the error in the SDWA.
Via SETRP
R/TM
To Step 6
ESTAE
8 Start any SRBs stopped when the error occurred.
STATUS
9 Record address of SRB if an RMTR was in control at time of error.
10 Attempt to retry the PURGEDQ request.
● Retry.
Exit
R/TM
Attempt PURGEDQ Retry
To Step 1
● No retry.
Exit
R/TM
Give control to caller's recovery.
SDWA
Recording Area

Diagram 19-20. PURGEDQ Processing (IEAVEPD0) (Part 4 of 4)

Extended Description	Module	Label
7 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 R/TM, which percolates the error to the ESTAE.		
8 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.		IEAVEPDE
9 If an error occurred in an RMTR routine, ESTAE records (in the SDWA) the address of the SRB that the RMTR was cleaning up.		
10 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.		

Diagram 19-21. SETLOCK Processing (IEAVELK) (Part 1 of 14)

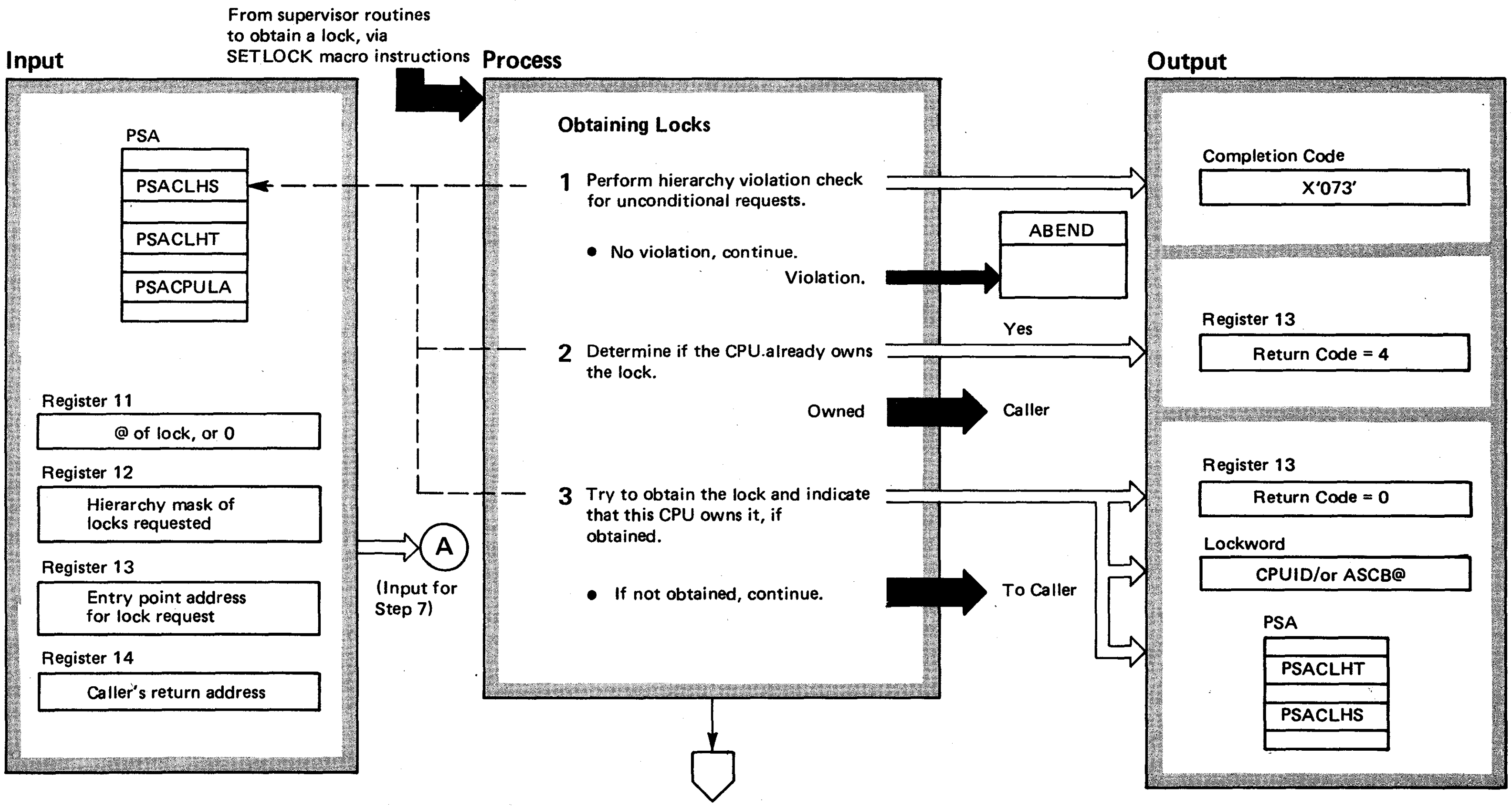
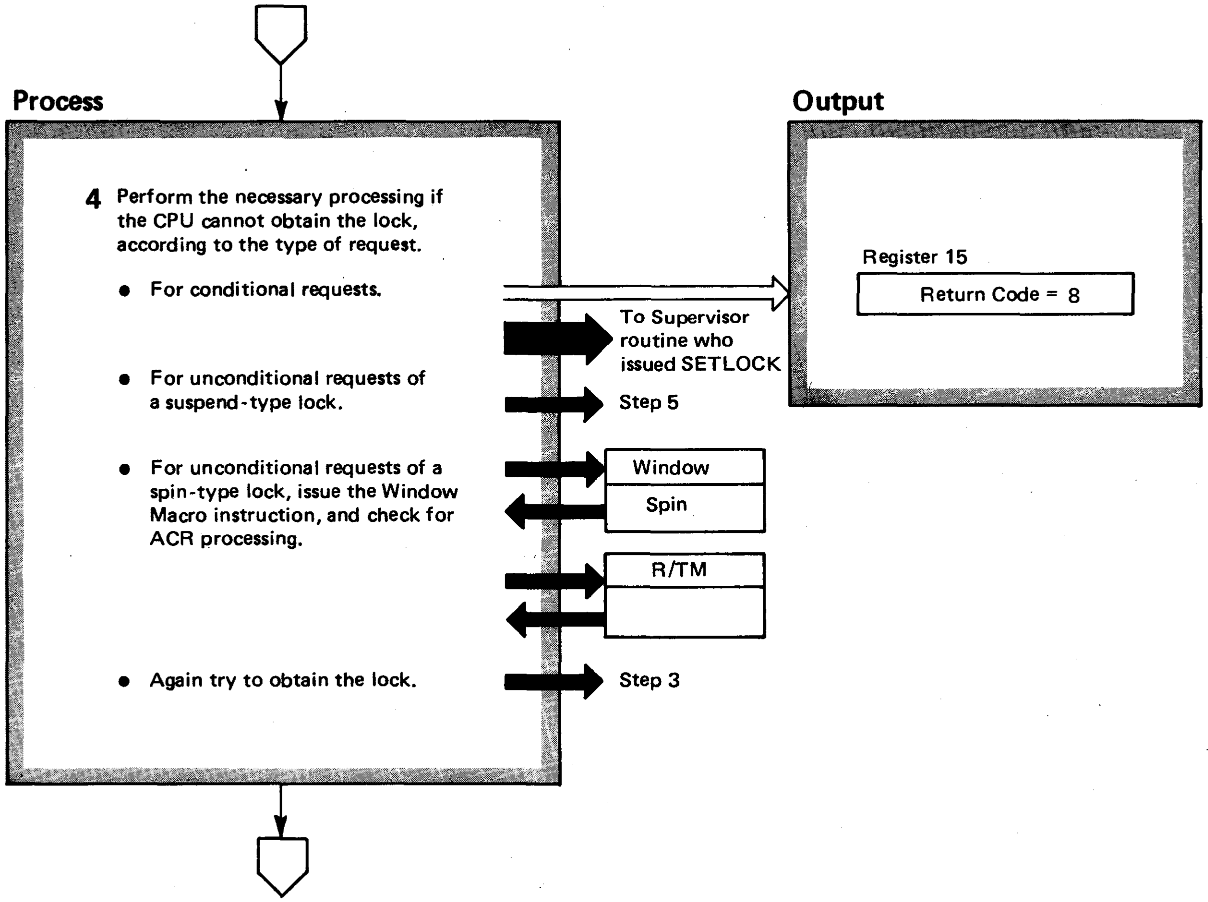

From supervisor routines to obtain a lock, via SETLOCK macro instructions
Input
PSA
PSACLHS
PSACLHT
PSACPULA
Register 11
@ of lock, or 0
Register 12
Hierarchy mask of locks requested
Register 13
Entry point address for lock request
Register 14
Caller's return address
A
(Input for Step 7)
Process
Obtaining Locks
1 Perform hierarchy violation check for unconditional requests.
• No violation, continue.
Violation.
ABEND
2 Determine if the CPU already owns the lock.
Yes
Owned
Caller
3 Try to obtain the lock and indicate that this CPU owns it, if obtained.
• If not obtained, continue.
To Caller
Output
Completion Code
X'073'
Register 13
Return Code = 4
Register 13
Return Code = 0
Lockword
CPUID/or ASCB@
PSA
PSACLHT
PSACLHS

Diagram 19-21. SETLOCK Processing (IEAVELK) (Part 2 of 14)

Extended Description	Module	Label
SETLOCK provides the means for a user to obtain "locks" that serialize the use of a resource. SETLOCK provides 13 locks: • DISP (for dispatcher lock) • IOSCAT (for IOS channel availability lock) • IOSUCB (for IOS unit control block lock) • IOSLCH (for IOS logical channel queue lock) • IOSYNCH (for IOS synchronization lock) • ASM (for auxiliary storage management lock) • SALLOC (for space allocation lock) • SRM (for the system resource management lock) • CMS (for the cross-memory services lock) • LOCAL (for local address space lock) SETLOCK both obtains and releases locks. There are two distinct methods of obtaining locks; conditional obtain and unconditional obtain. SETLOCK will immediately return control to the caller if no lock can be obtained for a conditional request; SETLOCK will not return control until the lock is obtained for an unconditional request.	IEAVELK	
1 SETLOCK determines whether the caller has violated the locking hierarchy by: • Requesting unconditionally a lock lower in the hierarchy while a higher lock is held. • Requesting the CMS lock while not holding the local lock. • Requesting a class lock when another lock in that class is already held. • Requesting a suspend lock while disabled. SETLOCK abnormally terminates callers who violate the hierarchy, with a X'073' completion code.	IEAVELK	
2 First, SETLOCK determines whether this CPU already owns the requested lock. If this CPU owns it, SETLOCK returns a code of 4 in register 13, and returns control to the caller. Otherwise, processing continues.		
3 SETLOCK tries to obtain the lock. If the lock is available (the lockword contains 0), SETLOCK indicates ownership by placing the logical CPUID in the lockword, setting the indicator in the CPU locks held string, PSACLHS, and for class locks, by storing the address of the lockword into the CPU locks held table, PSACLHT. SETLOCK will then return to the caller with a zero return code. If the lock is not available, proceed to step 4.		

Diagram 19-21. SETLOCK Processing (IEAVELK) (Part 3 of 14)



4 Perform the necessary processing if the CPU cannot obtain the lock, according to the type of request.

- For conditional requests.
- For unconditional requests of a suspend-type lock.
- For unconditional requests of a spin-type lock, issue the Window Macro instruction, and check for ACR processing.
- Again try to obtain the lock.

Diagram 19-21. SETLOCK Processing (IEAVELK) (Part 4 of 14)

Extended Description	Module	Label

4

- For conditional requests, SETLOCK will indicate a return code of 8 and return control to the caller.
- For unconditional requests of suspend locks (local or CMS), proceed to step 5.
- For unconditional requests of spin locks, enable for EMS (emergency signal) and MFA (malfunction alert) interruptions via the WINDOW macro. (This is done to prevent deadlock in case of failure on the other CPU.) SETLOCK will then determine if an ACR (alternate CPU recovery) condition has occurred. If so, it will route control to R/TM. SETLOCK again attempts to obtain the lock in step 3.

Diagram 19-21. SETLOCK Processing (IEAVELK) (Part 5 of 14)

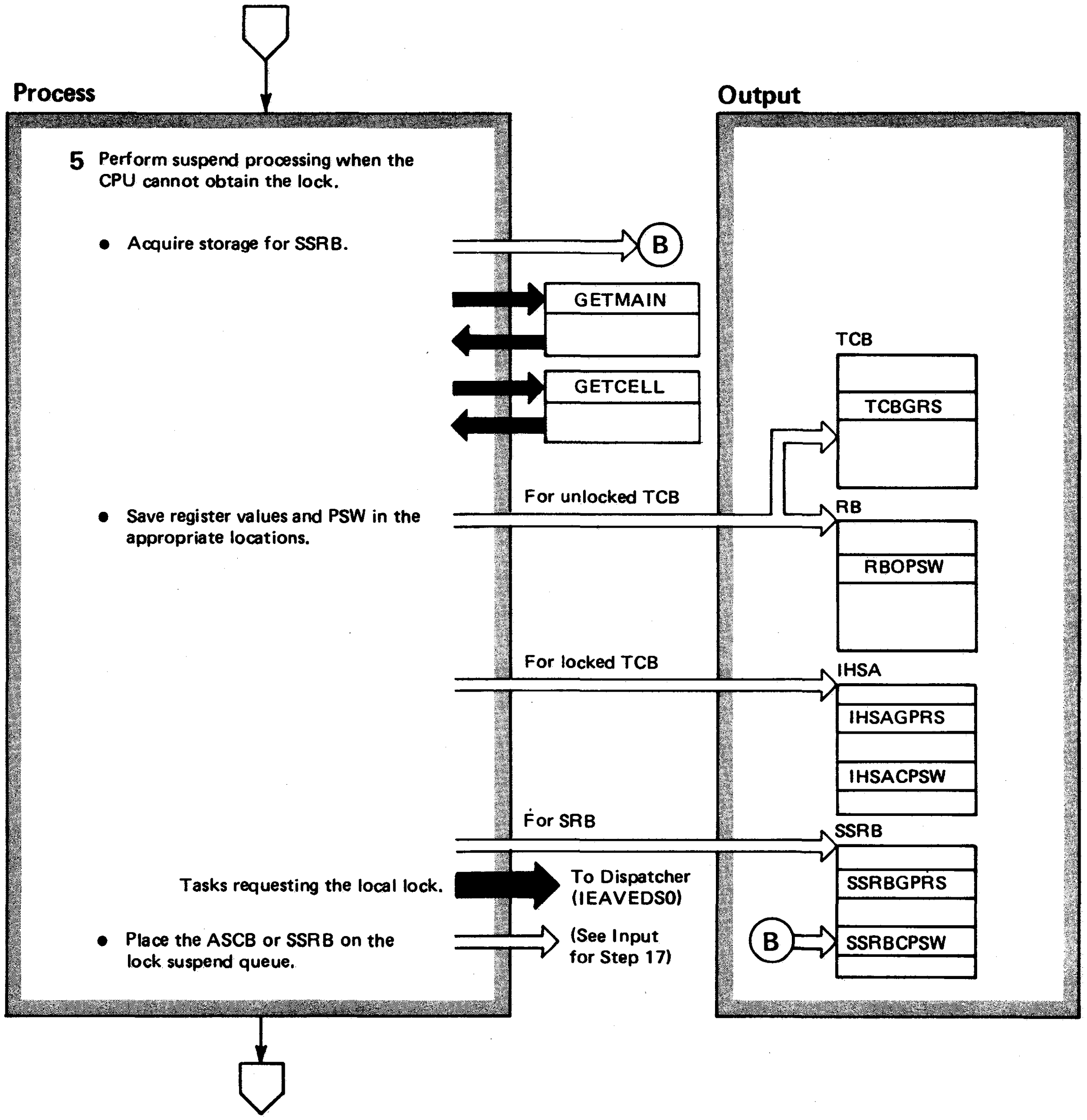

Process
Output
5 Perform suspend processing when the CPU cannot obtain the lock.
● Acquire storage for SSRB.
B
GETMAIN
GETCELL
● Save register values and PSW in the appropriate locations.
For unlocked TCB
TCB
TCBGRS
RB
RBOPSW
For locked TCB
IHSA
IHSAGPRS
IHSACPSW
For SRB
SSRB
SSRBGPRS
SSRBCPSW
Tasks requesting the local lock.
To Dispatcher (IEAVEDS0)
● Place the ASCB or SSRB on the lock suspend queue.
(See Input for Step 17)
B

Diagram 19-21. SETLOCK Processing (IEAVELK) (Part 6 of 14)

Extended Description	Module	Label
5 For callers in SRB mode, SETLOCK will acquire storage from SQA for a suspended SRB (SSRB) in which to save the suspend status. SETLOCK will then set resume registers and PSW to cause reentry to SETLOCK. The location of the saved status depends upon the mode of the caller. SETLOCK places either the ASCB (for tasks requesting the CMS lock) or the SSRB on the lock's suspend queue.		
For callers in task mode and owning no locks, SETLOCK will exit to the dispatcher. For callers in SRB mode or that own the local lock, continue at step **6**.		

Diagram 19-21. SETLOCK Processing (IEAVELK) (Part 7 of 14)

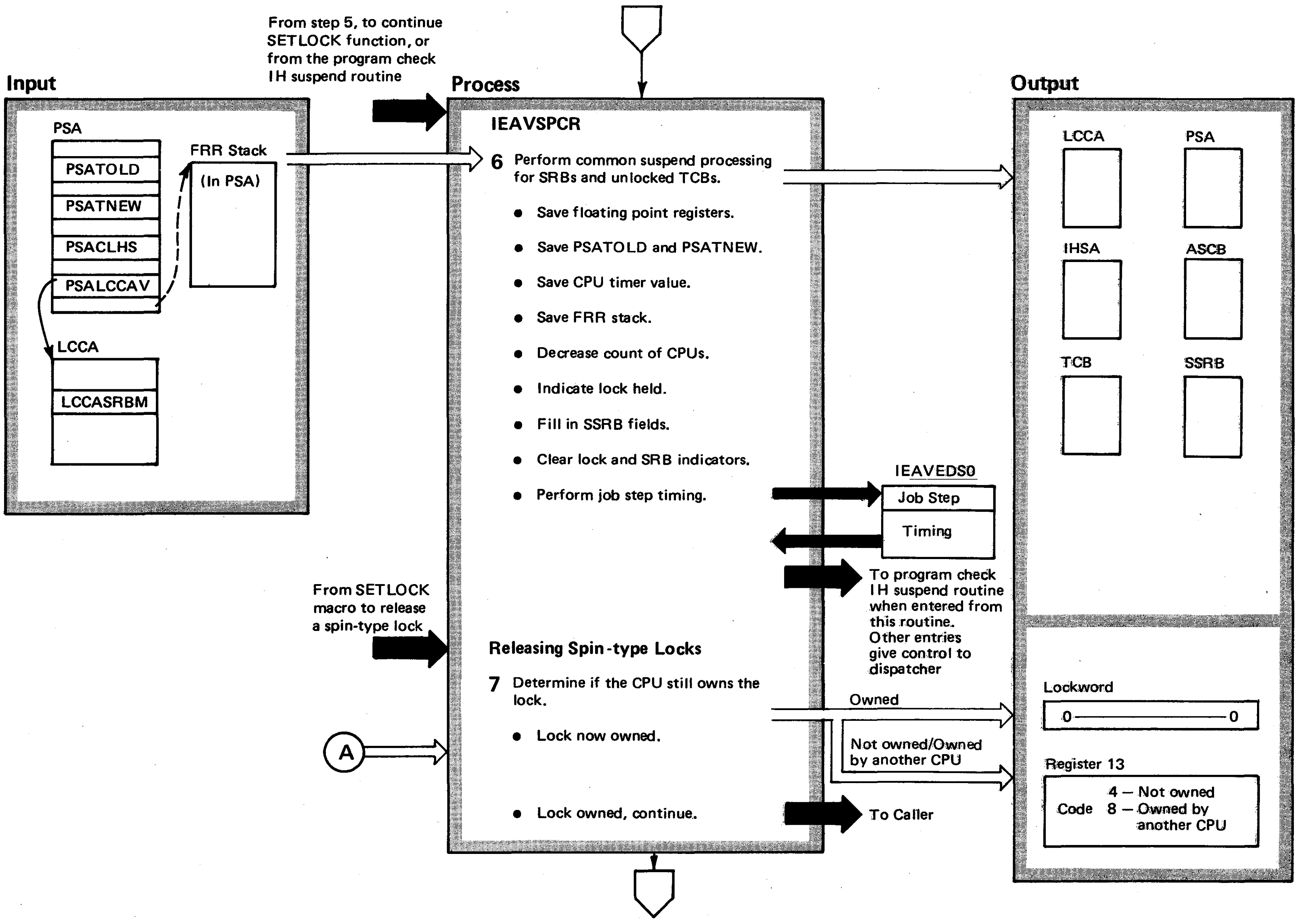

From step 5, to continue SETLOCK function, or from the program check IH suspend routine
Input
PSA
PSATOLD
PSATNEW
PSACLHS
PSALCCAV
FRR Stack
(In PSA)
LCCA
LCCASRBM
Process
IEAVSPCR
6 Perform common suspend processing for SRBs and unlocked TCBs.
• Save floating point registers.
• Save PSATOLD and PSATNEW.
• Save CPU timer value.
• Save FRR stack.
• Decrease count of CPUs.
• Indicate lock held.
• Fill in SSRB fields.
• Clear lock and SRB indicators.
• Perform job step timing.
IEAVEDS0
Job Step Timing
To program check IH suspend routine when entered from this routine. Other entries give control to dispatcher
From SETLOCK macro to release a spin-type lock
A
Releasing Spin-type Locks
7 Determine if the CPU still owns the lock.
• Lock now owned.
• Lock owned, continue.
Owned
Not owned/Owned by another CPU
To Caller
Output
LCCA
PSA
IHSA
ASCB
TCB
SSRB
Lockword
0 — 0
Register 13
Code 4 – Not owned
8 – Owned by another CPU

Diagram 19-21. SETLOCK Processing (IEAVELK) (Part 8 of 14)

Extended Description	Module	Label
6 This is the common suspend routine, entered from step 5 or from the program check IH suspend routine.		

If suspending a locked task:

- save PSATOLD and PSATNEW in IHSAOTCB and IHSANTCB
- decrease count of CPUs (ASCBCPUS)
- save floating point registers in IHSAFPRS
- save value of CPU timer in IHSACPUT
- save current FRR stack in IHSAFRRS

If suspending an SRB:

- clear SRB mode indicator
- set up SSRB for redispatch
- save floating point registers in SSRBFPRS
- save value of CPU timer in SSRBCPUT
- save current FRR stack in SSRBFRRS

For all suspend processing:

- Perform job step timing via the dispatcher's job step timing subroutine (DSJSTCR).
- If suspend lock is held, clear the lock held indicator in PSACLHS.
- If the local lock is held, store the suspend ID (X'7FFFFFFF') into the lockword to prevent any other routine from obtaining it.

Return to the program check IH suspend routine for entries from program check IH. Otherwise, exit to the dispatcher.

Extended Description	Module	Label
7 SETLOCK releases locks when the caller issues the SETLOCK macro using the RELEASE operand. Steps 7-9 describe release of spin locks, while steps 10-14 describe release of suspend locks.		

Determine if the lock is held by this CPU.

- If lock is not held by this CPU, then return to the caller with a return code in register 13. The return code equals 4 if no one owns the lock and equals 8 if another CPU owns the lock.
- If the lock is owned, the lock will be released by setting the lockword to zeros.

Diagram 19-21. SETLOCK Processing (IEAVELK) (Part 9 of 14)

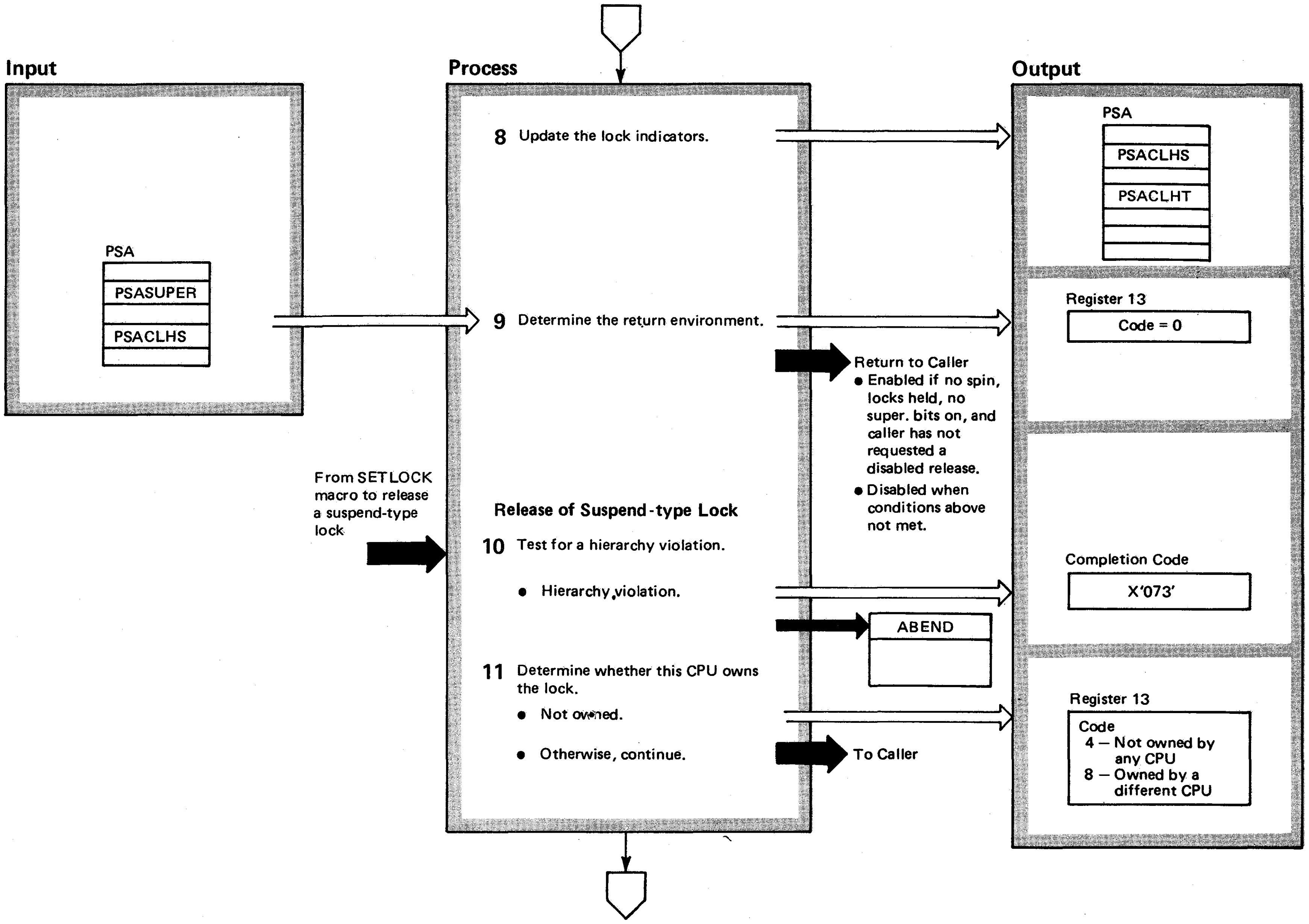
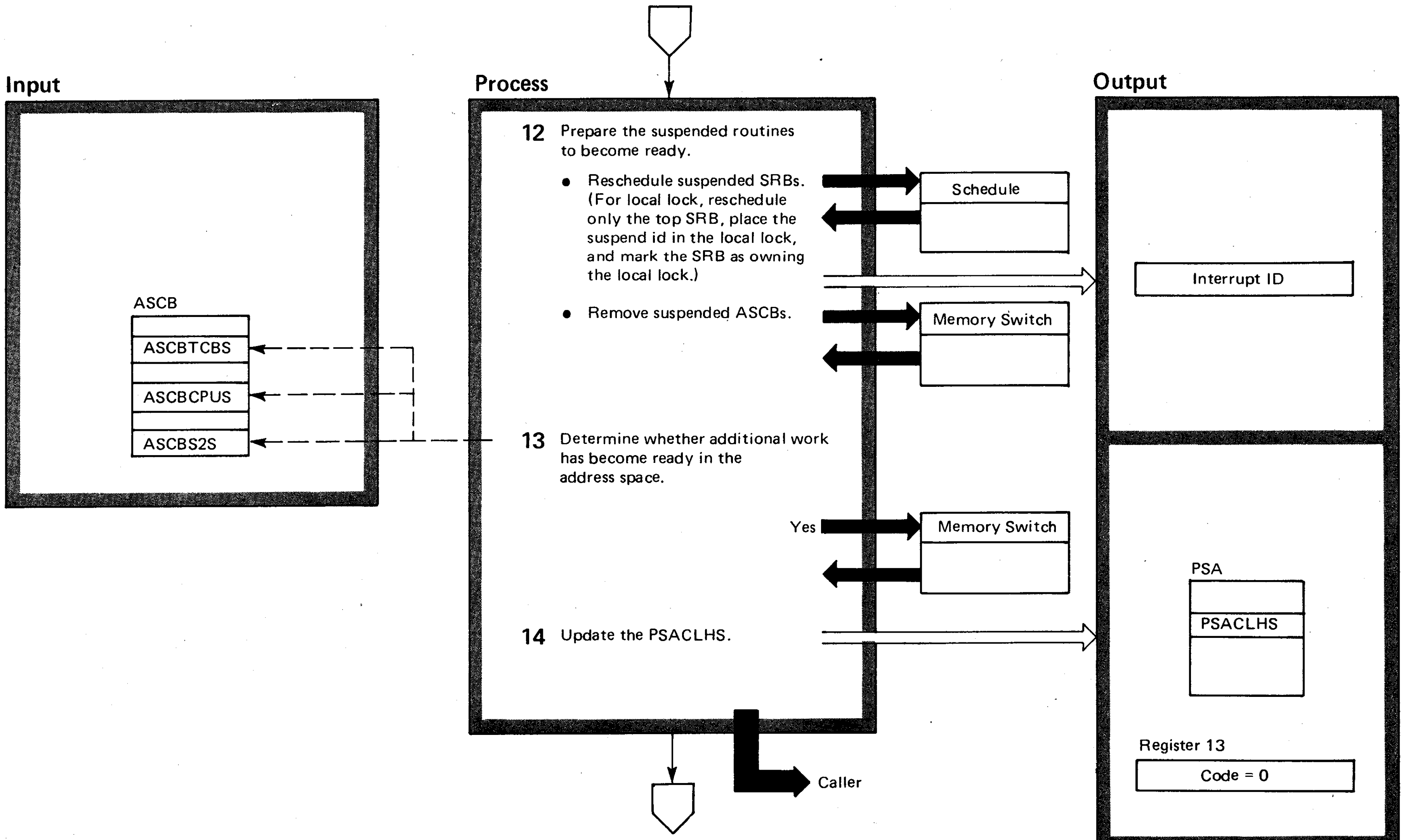

Input
Process
Output
PSA
PSASUPER
PSACLHS
8 Update the lock indicators.
9 Determine the return environment.
From SETLOCK macro to release a suspend-type lock
Release of Suspend-type Lock
10 Test for a hierarchy violation.
• Hierarchy violation.
11 Determine whether this CPU owns the lock.
• Not owned.
• Otherwise, continue.
Return to Caller
• Enabled if no spin, locks held, no super. bits on, and caller has not requested a disabled release.
• Disabled when conditions above not met.
ABEND
To Caller
PSA
PSACLHS
PSACLHT
Register 13
Code = 0
Completion Code
X'073'
Register 13
Code
4 – Not owned by any CPU
8 – Owned by a different CPU

Diagram 19-21. SETLOCK Processing (IEAVELK) (Part 10 of 14)

Extended Description	Module	Label
8 Update lock indicators by clearing the bit in the locks held string and clearing the entry in the locks held table if this is a class lock.		
9 Return to the caller disabled if any spin locks are held, any super bits are set, or the caller requested control returned disabled. Otherwise, enable the PSW. Return to the caller with a zero return code.		
10 SETLOCK tests for a hierarchy violation. The only violation on a release occurs if the caller tries to release the local lock while holding the CMS lock. If this occurs, the caller will be abnormally terminated with a X'073' completion code.		
11 If this CPU does not own the lock, return immediately to the caller.		

Diagram 19-21. SETLOCK Processing (IEAVELK) (Part 11 of 14)



12 Prepare the suspended routines to become ready.

- Reschedule suspended SRBs. (For local lock, reschedule only the top SRB, place the suspend id in the local lock, and mark the SRB as owning the local lock.)
- Remove suspended ASCBs.

13 Determine whether additional work has become ready in the address space.

14 Update the PSACLHS.

Diagram 19-21. SETLOCK Processing (IEAVELK) (Part 12 of 14)

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

12 If this is a suspend lock, make ready the routines suspended off the lock.

For local lock – dequeue and schedule the top SRB on that lock's suspend queue.

For CMS lock – reschedule any suspended SRBs that are on that locks suspend queue. Reset suspended tasks by placing the "interrupt ID" into the local lockword for each address space on the CMS suspend queue.

SETLOCK invokes Memory Switch for each readied address space.

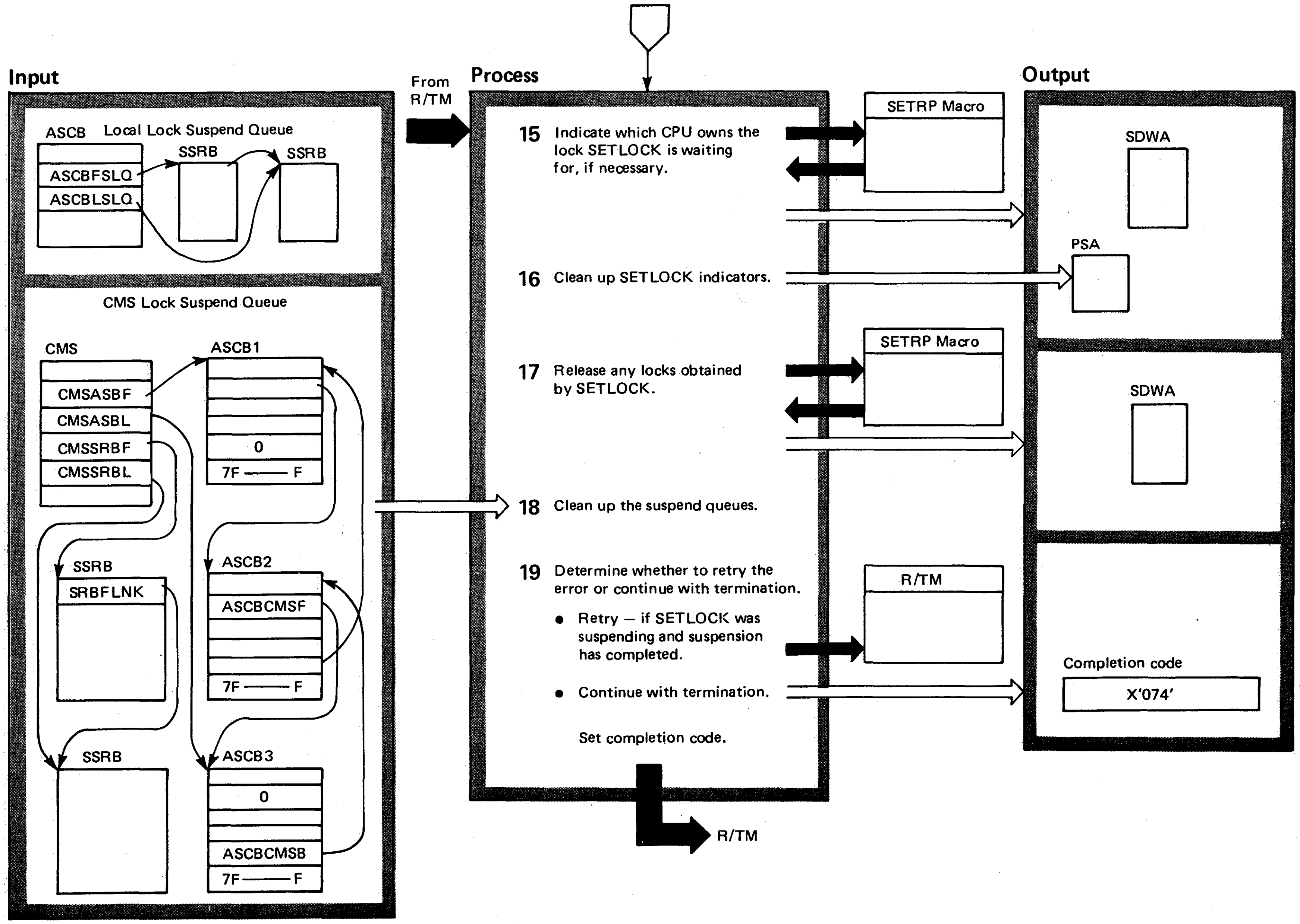
13 For the local lock release invoke Memory Switch for the current address space if there is ready work to be processed in the address space.

SETLOCK checks for the following conditions:

- ASCBTCBS greater than ASCBCPUS
- ASCBS2S set to one.

14 SETLOCK updates the PSACLHS, and returns to the caller with a 0 in register 13.

Diagram 19-21. SETLOCK Processing (IEAVELK) (Part 13 of 14)



15 Indicate which CPU owns the lock SETLOCK is waiting for, if necessary.

16 Clean up SETLOCK indicators.

17 Release any locks obtained by SETLOCK.

18 Clean up the suspend queues.

19 Determine whether to retry the error or continue with termination.

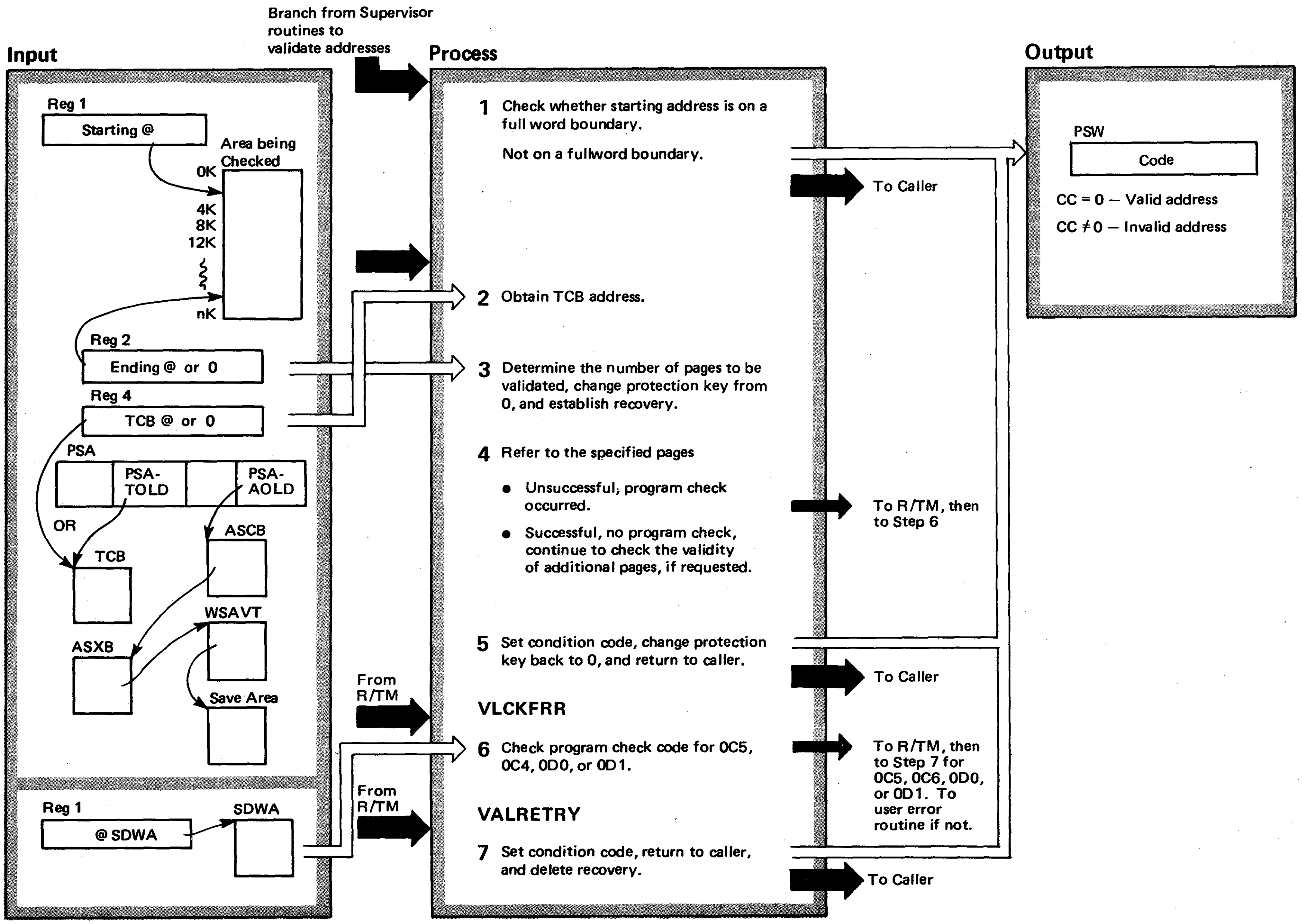
- Retry – if SETLOCK was suspending and suspension has completed.
- Continue with termination.

Set completion code.

Diagram 19-21. SETLOCK Processing (IEAVELK) (Part 14 of 14)

Extended Description	Module	Label
15 The SETLOCK FRR (functional recovery routine) frees any locks that SETLOCK obtained before the error occurred, cleans up any indicators set by SETLOCK, and corrects the suspend queues in use when the error occurred. The SETLOCK FRR then gives R/TM control either to continue with termination or, if one of two conditions occur, to retry the failing operation. The two retry conditions follow: ● A restart interruption occurred while one CPU spins on a lock, or ● An error occurred after lock suspension processing had completed. If SETLOCK was spinning on a lock and a restart interruption occurred, it indicates to R/TM which CPU owns the lock. The SETLOCK FRR uses the SETRP macro instruction to indicate the CPU and to accumulate recording information in the SDWA.	IEAVLKRR	IEAVELKR
16 The SETLOCK FRR cleans up indicators in the PSA.		
17 The SETLOCK FRR requests that R/TM, via the SETRP macro instruction, release any locks obtained by SETLOCK during its processing.		
18 The SETLOCK FRR removes SSRBs from the CMS lock suspend queue, SSRBs from the current local lock suspend queue, and ASCBs from the CMS lock suspend queue. It resets all routines suspended on the lock, so that all these routines must re-request the lock.		
19 Two conditions result in retry of the failing operation: a restart interruption occurs while one CPU waits for a lock owned by another CPU; or an error occurs during CMS lock or local lock suspend processing and the suspend processing has completed. Any other errors result in control going to R/TM with a X'074' completion code and an indication to continue with termination.		

Diagram 19-22. Validity Check Processing (IEAVEVAL) (Part 1 of 2)



1 Check whether starting address is on a full word boundary.

Not on a fullword boundary.

2 Obtain TCB address.

3 Determine the number of pages to be validated, change protection key from 0, and establish recovery.

4 Refer to the specified pages

- Unsuccessful, program check occurred.
- Successful, no program check, continue to check the validity of additional pages, if requested.

5 Set condition code, change protection key back to 0, and return to caller.

VLCKFRR

6 Check program check code for 0C5, 0C4, 0D0, or 0D1.

VALRETRY

7 Set condition code, return to caller, and delete recovery.

Diagram 19-22. Validity Check Processing (IEAVEVAL) (Part 2 of 2)

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 starting addresses not on a fullword boundary. Validity check passes a non-zero condition code (in the PSW) to the caller. Processing continues for address on a fullword boundary (at step 2).	IEAVEVAL	IEA0VL01
2 Validity check obtains the current TCB address from PSATOLD if the requester did not specify one.		IEA0VL00
3 Validity check next determines how many pages must be validated. Then, validity check changes its protection key to match the key specified by the caller's TCBPKF field. Validity check establishes a recovery routine to intercept any program checks.		
4 Validity check uses a compare and swap instruction for validation. 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 R/TM. R/TM gives control to the recovery routine at step 6, entry point VLCKFRR.		
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 R/TM 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, 0C5, 0D0, or 0D1. If one of the four expected errors occurred, control goes to R/TM to retry at entry point VALRETRY. Otherwise, control goes to R/TM to give the caller's error routine control.		VLCKFRR
7 R/TM 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 19-23. ASCBCHAP Processing (IEAVEAC0) (Part 1 of 4)

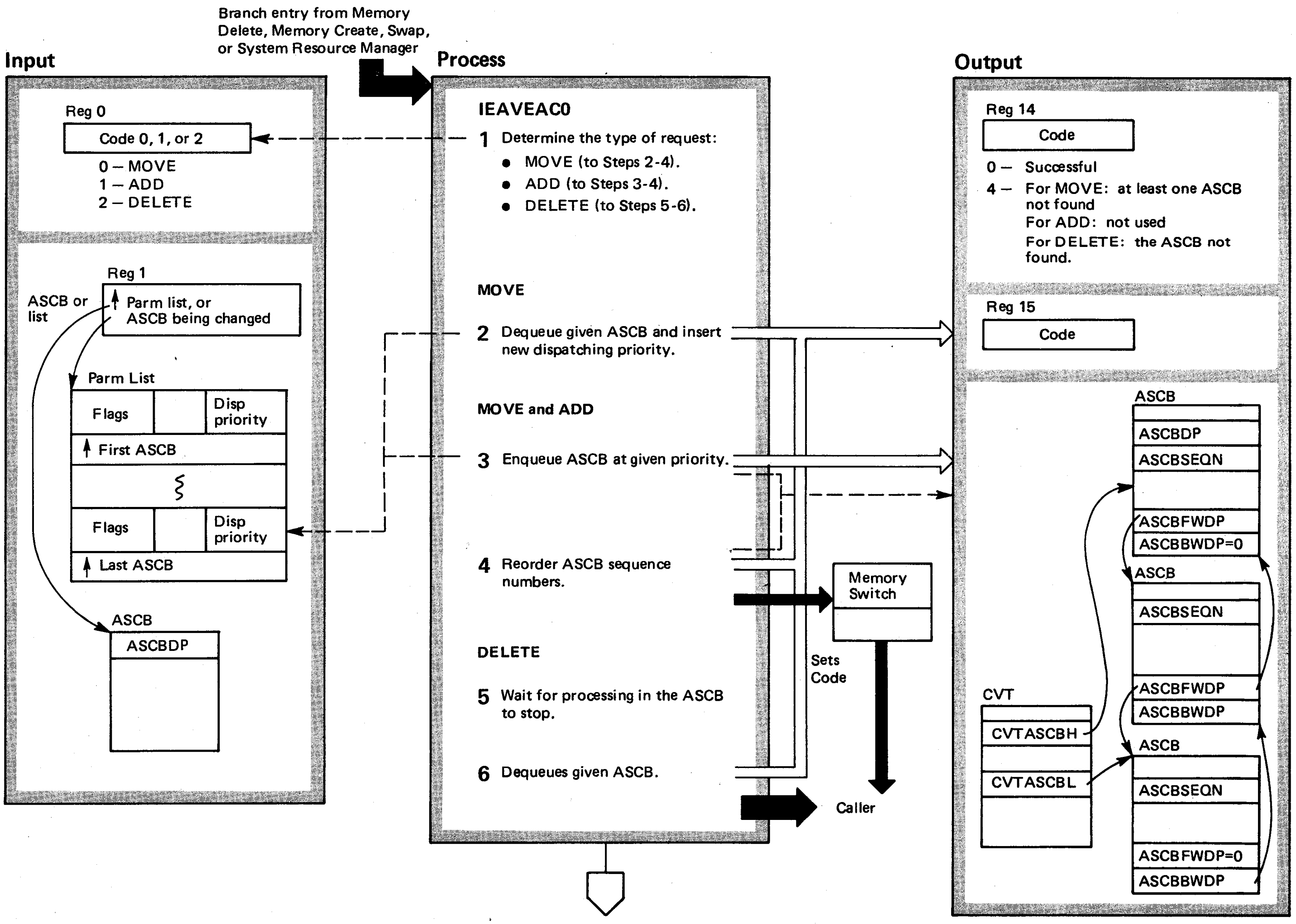


Diagram 19-23. ASCBCHAP Processing (IEAVEAC0) (Part 2 of 4)

Extended Description	Module	Label
ASCBCHAP alters the dispatching priority of ASCBs at the request of the system resource manager, and adds or deletes ASCBs to the ASCB queue for memory create or memory delete (see Obtaining a New Virtual Memory (IEAVEMCR) and Deleting a Virtual Memory (IEAVEMDL)). The ASCBCHAP routine has no SVC entry; it only has branch entry. Only privileged programs use the ASCBCHAP routine.		
The ASCBCHAP routine obtains the global dispatcher lock (if it is not already held).		
1 ASCBCHAP determines the type of request according to the code in register 1.		
2 ASCBCHAP changes the priority of several ASCBs at one time. This enhances performance. The parameter list that register 1 points to contains the list of ASCBs being changed. This parameter list must be in non-pageable storage, because ASCBCHAP refers to it with the global dispatcher lock held. ASCBCHAP dequeues the ASCBs being changed.		

Extended Description	Module	Label
3 For ADD requests, ASCBCHAP refers to register 1, which contains the address of the ASCB being added to the ASCB ready queue. The ASCBDP field has the new dispatching priority prior to entering ASCBCHAP.		
4 To resequence the ASCB ready queue, ASCBCHAP changes fields in the ASCB and CVT, as illustrated. Memory switch (see Memory Switch (IEAVEMS0)) receives control to process the ASCB with the highest dispatching priority.		
5 For DELETE requests, ASCBCHAP refers to register 1, which contains the address of the ASCB being deleted.		
6 ASCBCHAP frees the global dispatcher lock, unless it was already held upon entry.		

Diagram 19-23. ASCBCHAP Processing (IEAVEAC0) (Part 3 of 4)

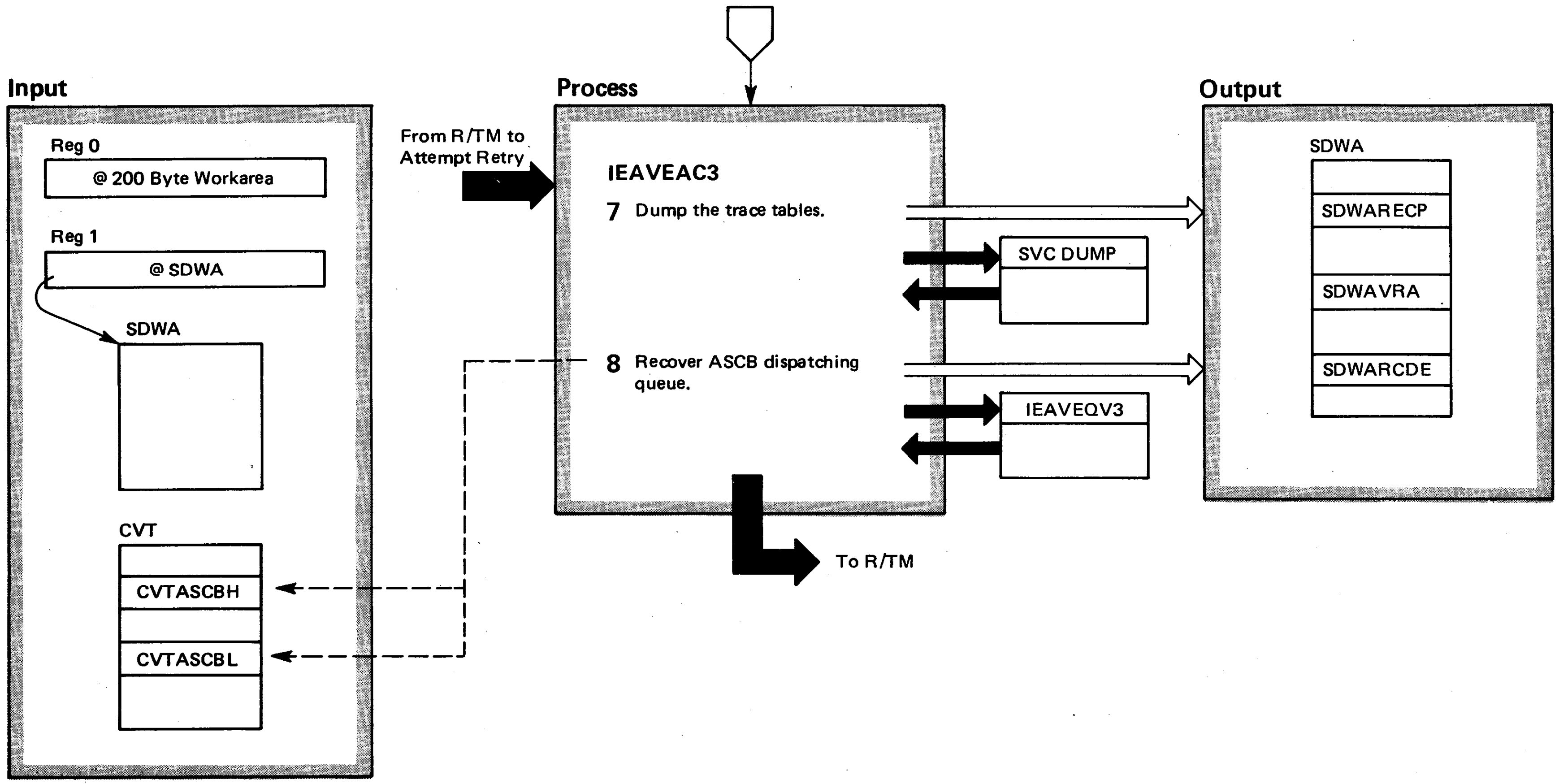
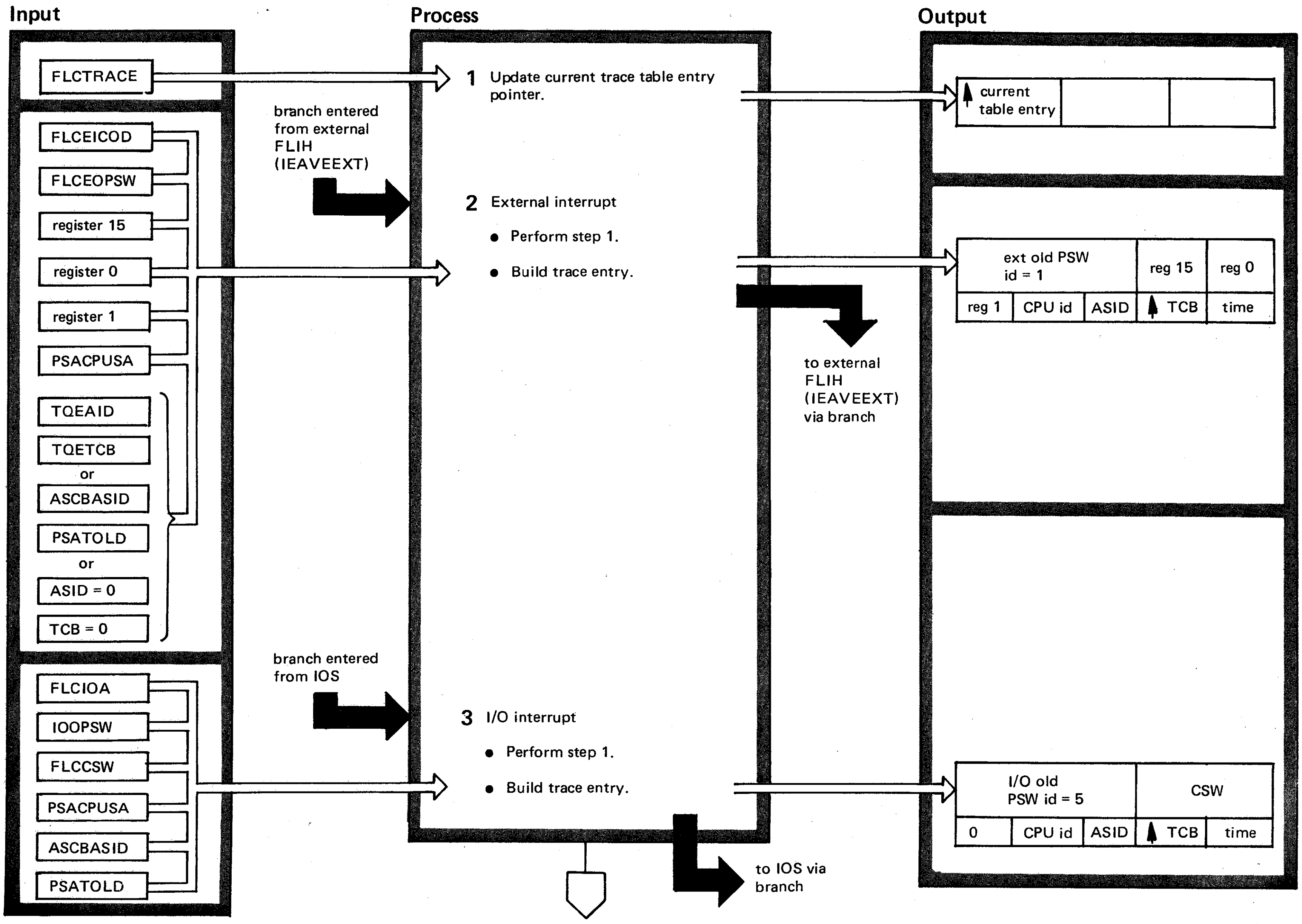


Diagram 19-23. ASCBCHAP Processing (IEAVEAC0) (Part 4 of 4)

Extended Description	Module	Label
7 The ASCBCHAP FRR calls SVC DUMP to dump the contents of the trace table.	ASCBCHAP	
8 Control goes to the IEAVEQV3 routine to recover the ASCB dispatching queue. Then, control goes to R/TM, with a return code of 0, indicating no retry.		

Diagram 19-24. Trace Processing (IEAVTRCE) (Part 1 of 10)



1 Update current trace table entry pointer.

2 External interrupt

- Perform step 1.
- Build trace entry.

3 I/O interrupt

- Perform step 1.
- Build trace entry.

Diagram 19-24. Trace Processing (IEAVTRCE) (Part 2 of 10)

Extended Description		Module	Label
VS2 system trace routine (IEAVTRCE) records system activity in the trace table.			
The following system activities are recorded in the trace table and are discussed in detail in the following steps:	*Identifier*		
• External interrupts	1	IEAVTRCE	TREX
• I/O interrupts	5	IEAVTRCE	TRIO
• Program interrupts	3	IEAVTRCE	TRPI
• SVC interrupts	2	IEAVTRCE	TRSVC
• SIO EVENT	0	IEAVTRCE	TRSIO, TRACE
• Dispatcher event	7	IEAVTRCE	TRDISP
• Initial SRB dispatcher event	4	IEAVTRCE	TRSRB1
• SRB re-dispatch event	6	IEAVTRCE	TRSRB2

The identifier for each activity is located in bit positions 17-19 of the Trace Table Entry. Control is returned to the caller via a branch.

1 The current trace table entry is updated to the system activity being recorded. This step is first for all entries. (The timer value for all trace events is bytes 2-5 of the clock value obtained by a STCK instruction. The CPU id for all trace events is obtained by adding X'40' to the physical CPU id (PSACPUSA) to produce the logical CPU id.)

2 *External interrupt* — Module: IEAVTRCE, Label: TREX

System data is gathered into trace records. ASCBASID and PSATOLD are traced if the interrupt is not a clock comparator. If it is a clock comparator, the TQE and TQETCB are traced. If the TQE address is 0, neither ASID nor TCB are traced.

If the system is waiting and a clock comparator interrupt occurs, it is not traced. This prevents the trace table from overlaying itself with useless information while the system is waiting. The external old PSW id in the output is the EC mode external old PSW with appropriate external interrupt code inserted into the BC mode PSW interrupt code position (bits 16-31).

3 *I/O interrupt* — Module: IEAVTRCE, Label: TRIO

System data is gathered into a trace record. The I/O old PSW id in the output is the EC mode old PSW with the device address inserted into BC mode interrupt code position (bits 16-31).

Diagram 19-24. Trace Processing (IEAVTRCE) (Part 3 of 10)

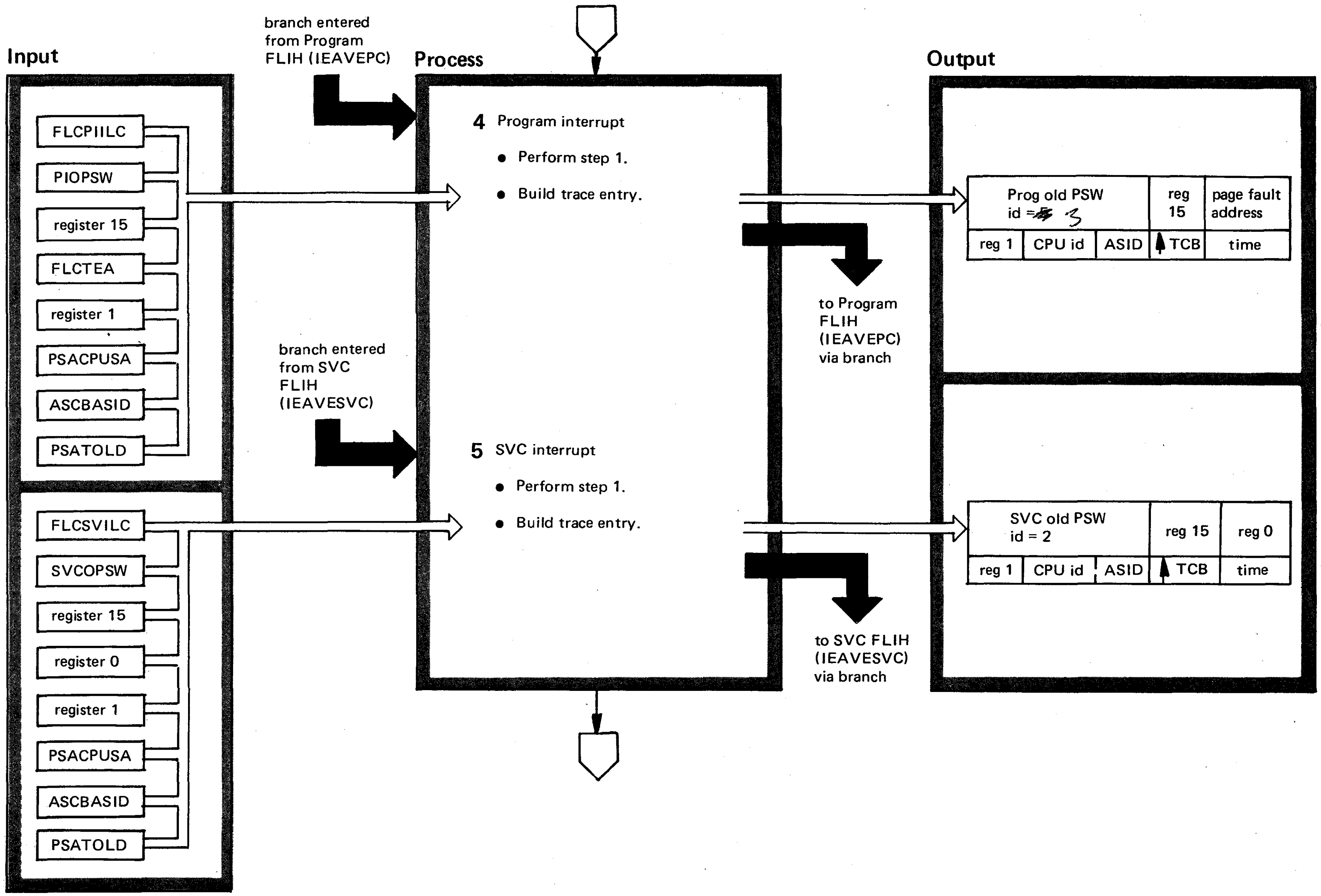


Diagram 19-24. Trace Processing (IEAVTRCE) (Part 4 of 10)

Extended Description	Module	Label
4 *Program interrupt* System data is gathered into a trace record. The last page fault address (FLCTEA) is traced for all program interrupts. The program old PSW id in the output is the EC mode old PSW with a program check interrupt code inserted into the BC mode PSW interrupt code position (bits 16-31).	IEAVTRCE	TRPI
5 *SVC interrupt* System data is gathered into a trace record. The SVC old PSW id in the output is the ECB old PSW with SVC interrupt code inserted into the BC mode PSW interrupt code position (bits 16-31).	IEAVTRCE	TRSVC

Diagram 19-24. Trace Processing (IEAVTRCE) (Part 5 of 10)

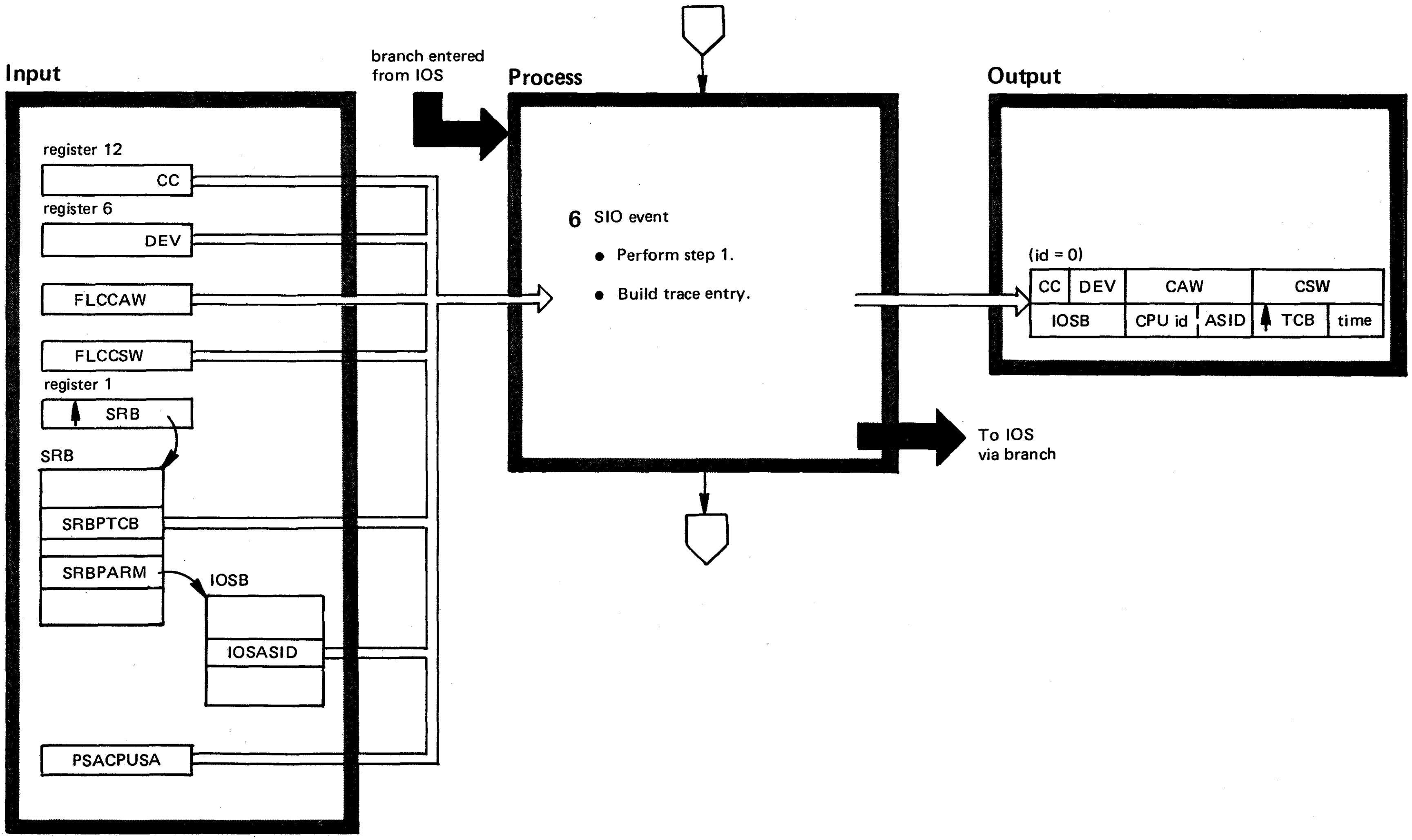
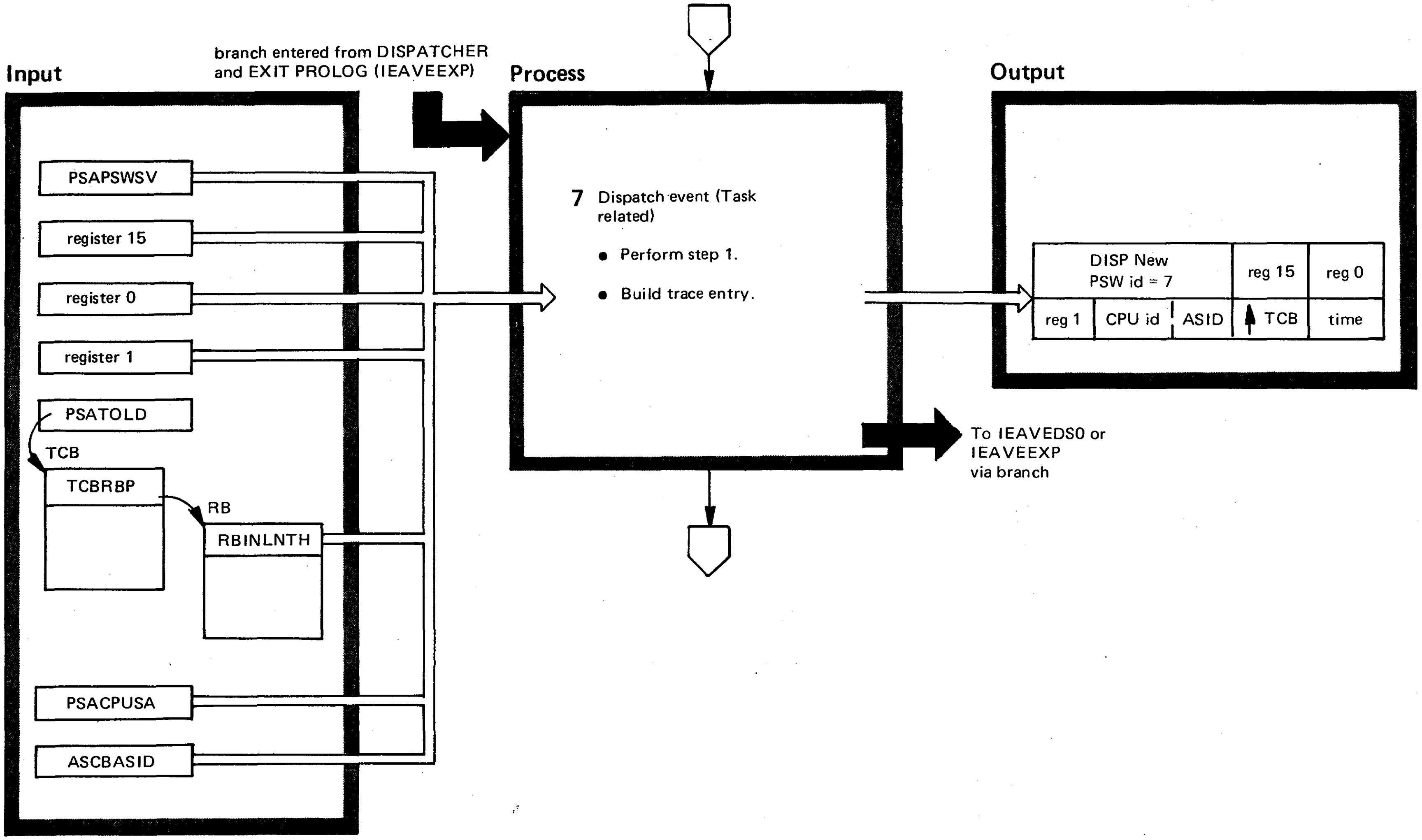


Diagram 19-24. Trace Processing (IEAVTRCE) (Part 6 of 10)

Extended Description	Module	Label
6 *SIO EVENT* System data is gathered into a trace record.		

Diagram 19-24. Trace Processing (IEAVTRCE) (Part 7 of 10)



Extended Description	Module	Label
7 *Dispatcher event* System data is gathered into a trace record.	IEAVTRCE	TRDISP

Only the initial dispatch of the wait task is traced. Subsequent dispatches of the wait task while the system is waiting are not traced.

If a TCB is available (PSATOLD ≠ 0), the interrupt information (ILC and code) is gathered from the top RB's prefix and incorporated in the PSW.

Diagram 19-24. Trace Processing (IEAVTRCE) (Part 9 of 10)

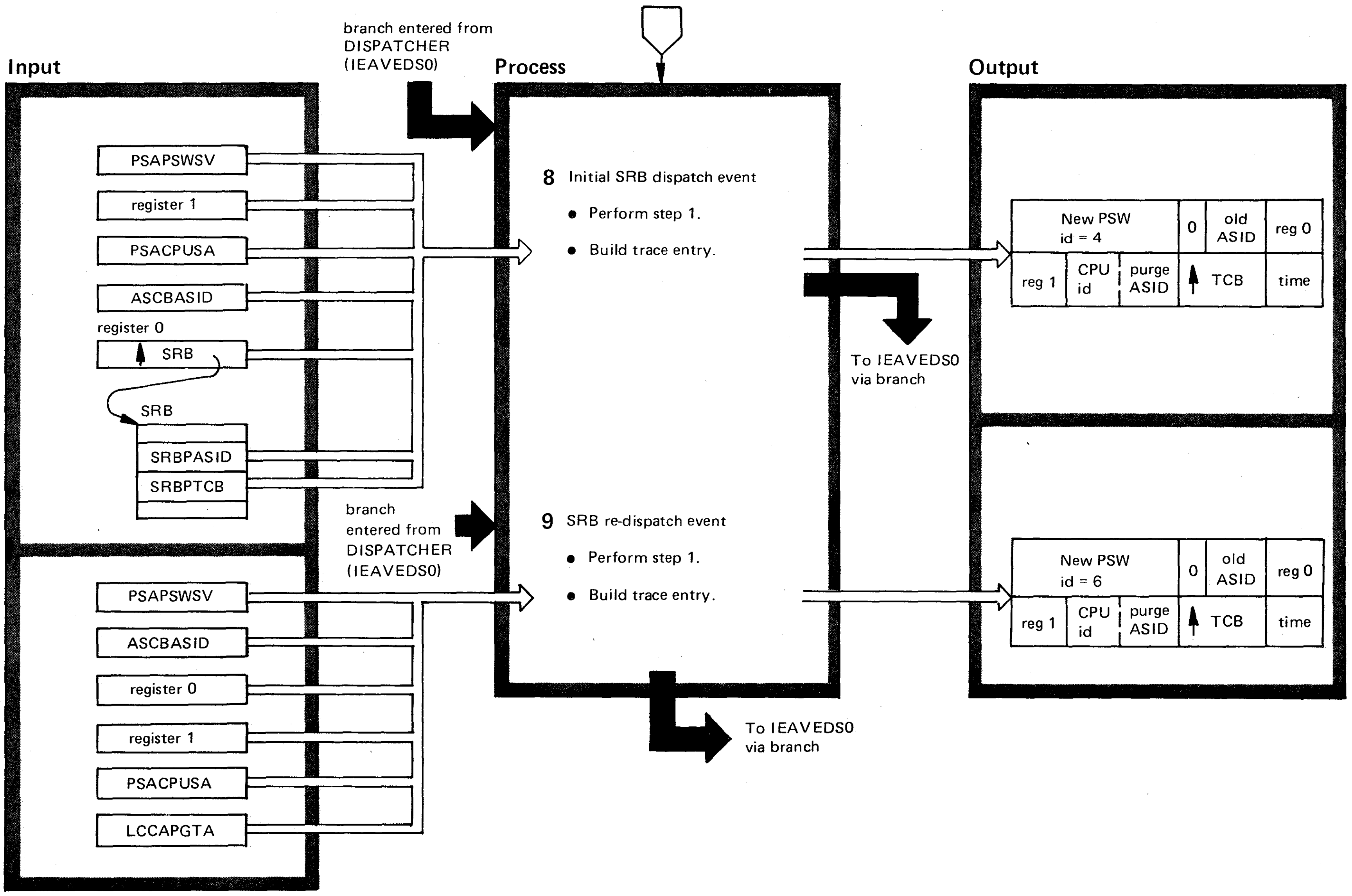
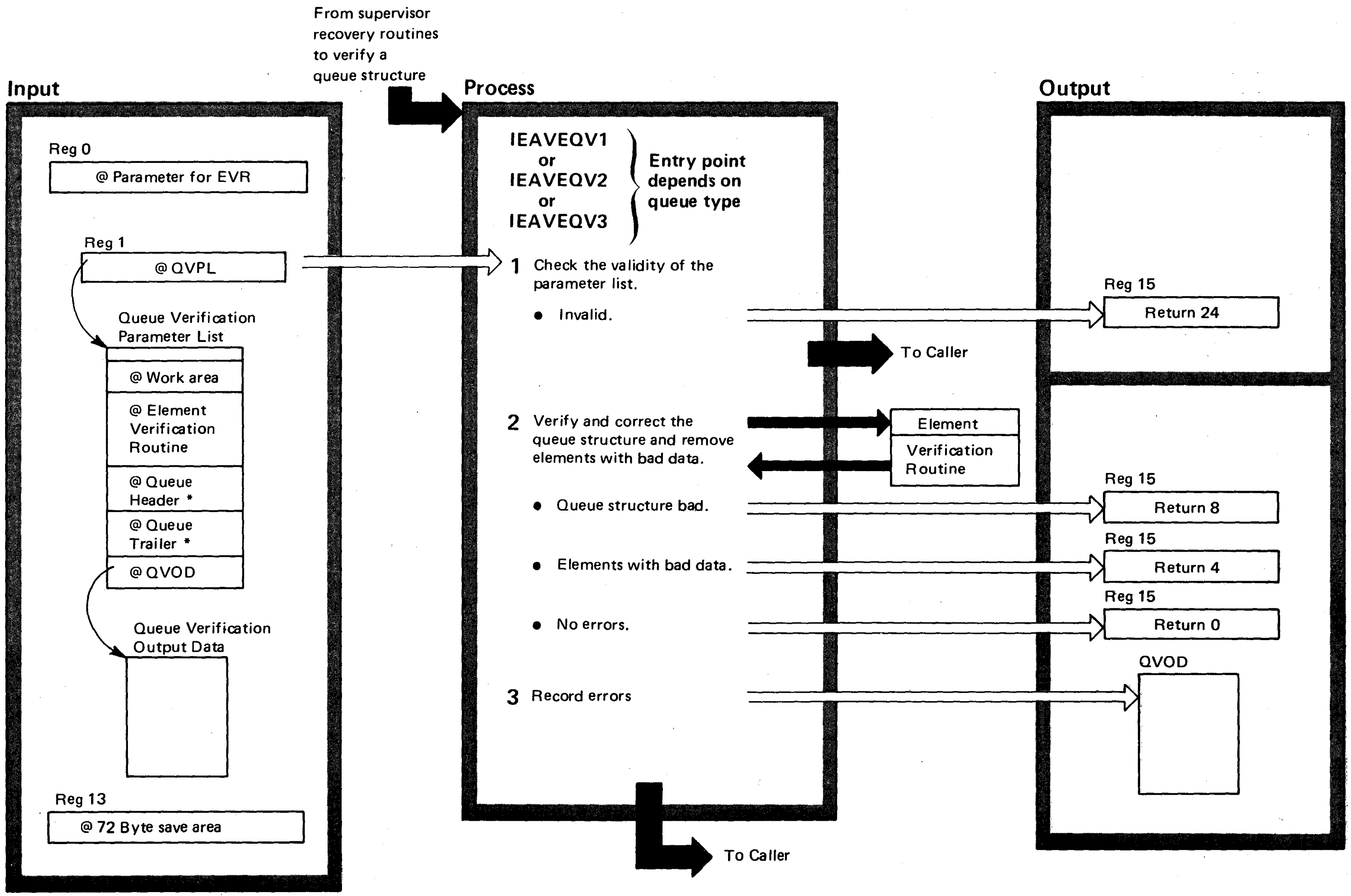


Diagram 19-24. Trace Processing (IEAVTRCE) (Part 10 of 10)

Extended Description	Module	Label
8 *Initial SRB dispatch event* System data is gathered into a trace record.	IEAVTRCE	TRSRB1
9 *SRB re-dispatch event* System data is gathered into a trace record.	IEAVTRCE	TRSRB2

Diagram 19-25. Queue Verification (IEAVEQV0) (Part 1 of 2)

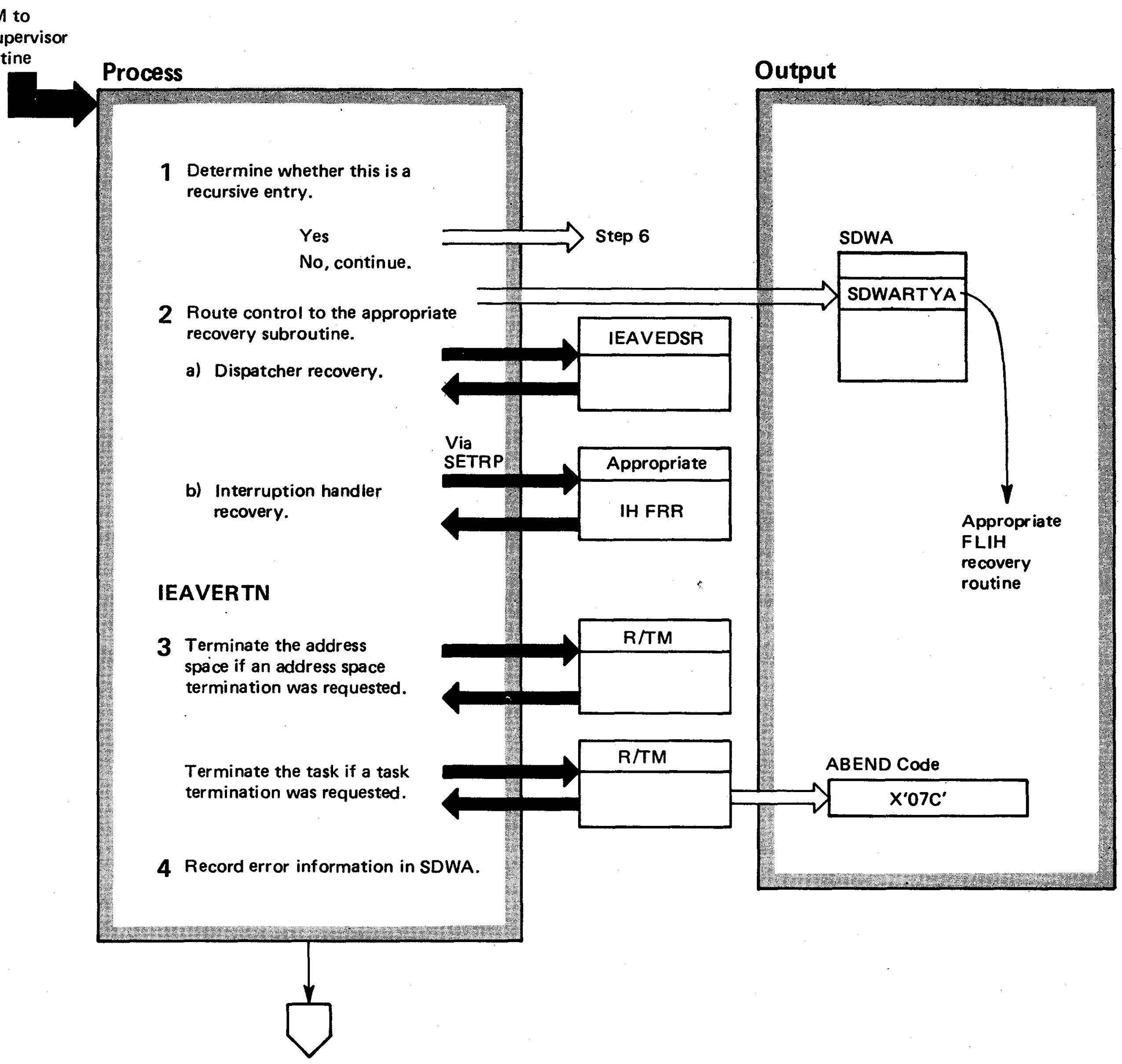

From supervisor recovery routines to verify a queue structure
Input
Process
Output
Reg 0
@ Parameter for EVR
Reg 1
@ QVPL
Queue Verification Parameter List
@ Work area
@ Element Verification Routine
@ Queue Header *
@ Queue Trailer *
@ QVOD
Queue Verification Output Data
Reg 13
@ 72 Byte save area
IEAVEQV1 or IEAVEQV2 or IEAVEQV3
Entry point depends on queue type
1 Check the validity of the parameter list.
• Invalid.
To Caller
2 Verify and correct the queue structure and remove elements with bad data.
Element Verification Routine
• Queue structure bad.
• Elements with bad data.
• No errors.
3 Record errors
To Caller
Reg 15
Return 24
Reg 15
Return 8
Reg 15
Return 4
Reg 15
Return 0
QVOD

* Queue header and trailer point to queue(s) being verified

Diagram 19-25. Queue Verification (IEAVEQV0) (Part 2 of 2)

Extended Description	Module	Label
1 The Queue Verifier performs some validity checking of input parameters to minimize the possibility of the caller incorrectly coding the interface. Queue Verifier returns control to the caller immediately with a return code of 24 in register 15 if it detects invalid input parameters.	IEAVEQV0	
2 Queue Verifier 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 Verifier 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.) 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 Verifier 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.		
3 All errors encountered are recorded in the Queue Verification Output Data (QVOD) area. 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.		

Diagram 19-26. Super FRR (IEAVESPR) (Part 1 of 4)

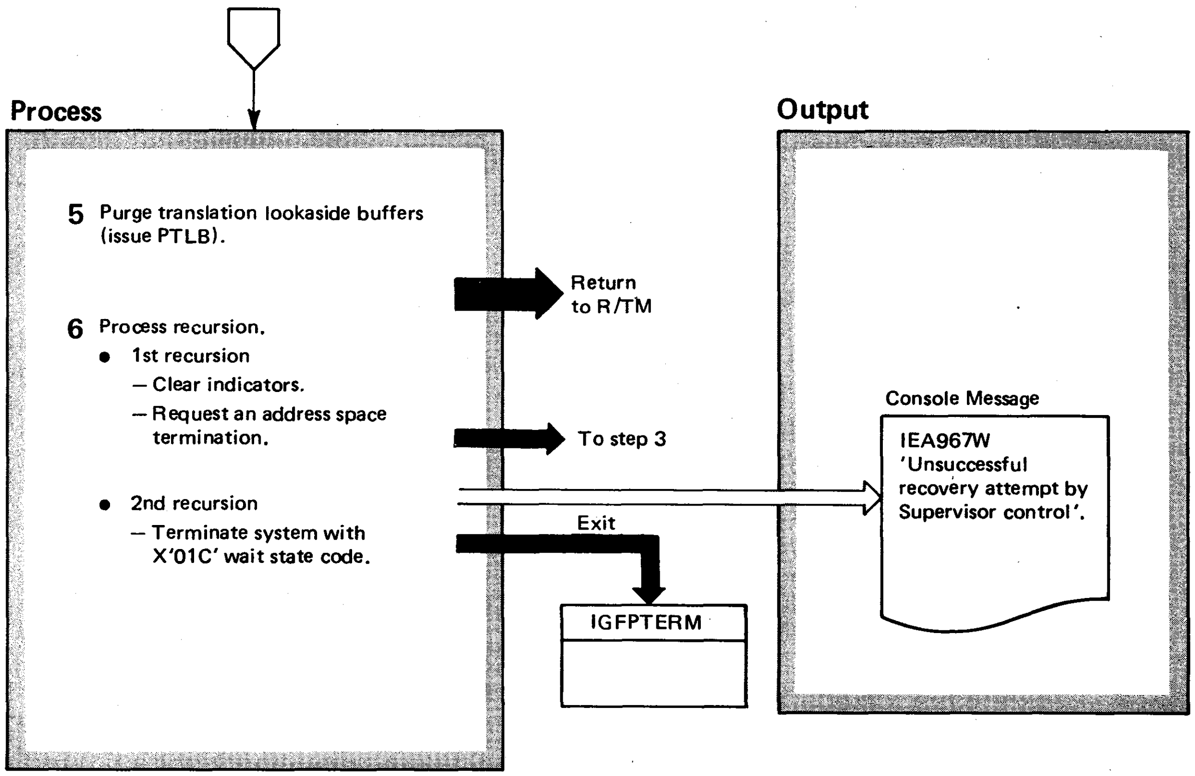


From R/TM to recover a supervisor control routine
Process
Output
1 Determine whether this is a recursive entry.
Yes
Step 6
No, continue.
SDWA
SDWARTYA
2 Route control to the appropriate recovery subroutine.
IEAVEDSR
a) Dispatcher recovery.
Via SETRP
Appropriate
IH FRR
b) Interruption handler recovery.
Appropriate FLIH recovery routine
IEAVERTN
3 Terminate the address space if an address space termination was requested.
R/TM
Terminate the task if a task termination was requested.
R/TM
ABEND Code
X'07C'
4 Record error information in SDWA.

Diagram 19-26. 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 6 for recursive entries; otherwise, processing continues. If a DAT error occurred, Super FRR requests an address space termination (see step 3).	IEAVESPR	
2 The Super FRR uses SETRP to indicate a retry address to one of the FLIH recovery routines. After Super FRR returns to R/TM, R/TM routes control to the specified retry address. The recovery routines that protect the dispatcher and the interruption handler are: • Dispatcher – IEAVEDSR • SVC IH – IEAVESVR • I/O IH – IEAVEIOR • External IH – IEAVEE1R, IEAVEE2R, and IEAVEE3R • Program check IH – IEAVEPCR • Restart IH – IEAVERER		
3 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 6.		
4 The Super FRR records error information in the SDWA (system diagnostic work area).		

Diagram 19-26. Super FRR (IEAVESPR) (Part 3 of 4)



5 Purge translation lookaside buffers (issue PTLB).

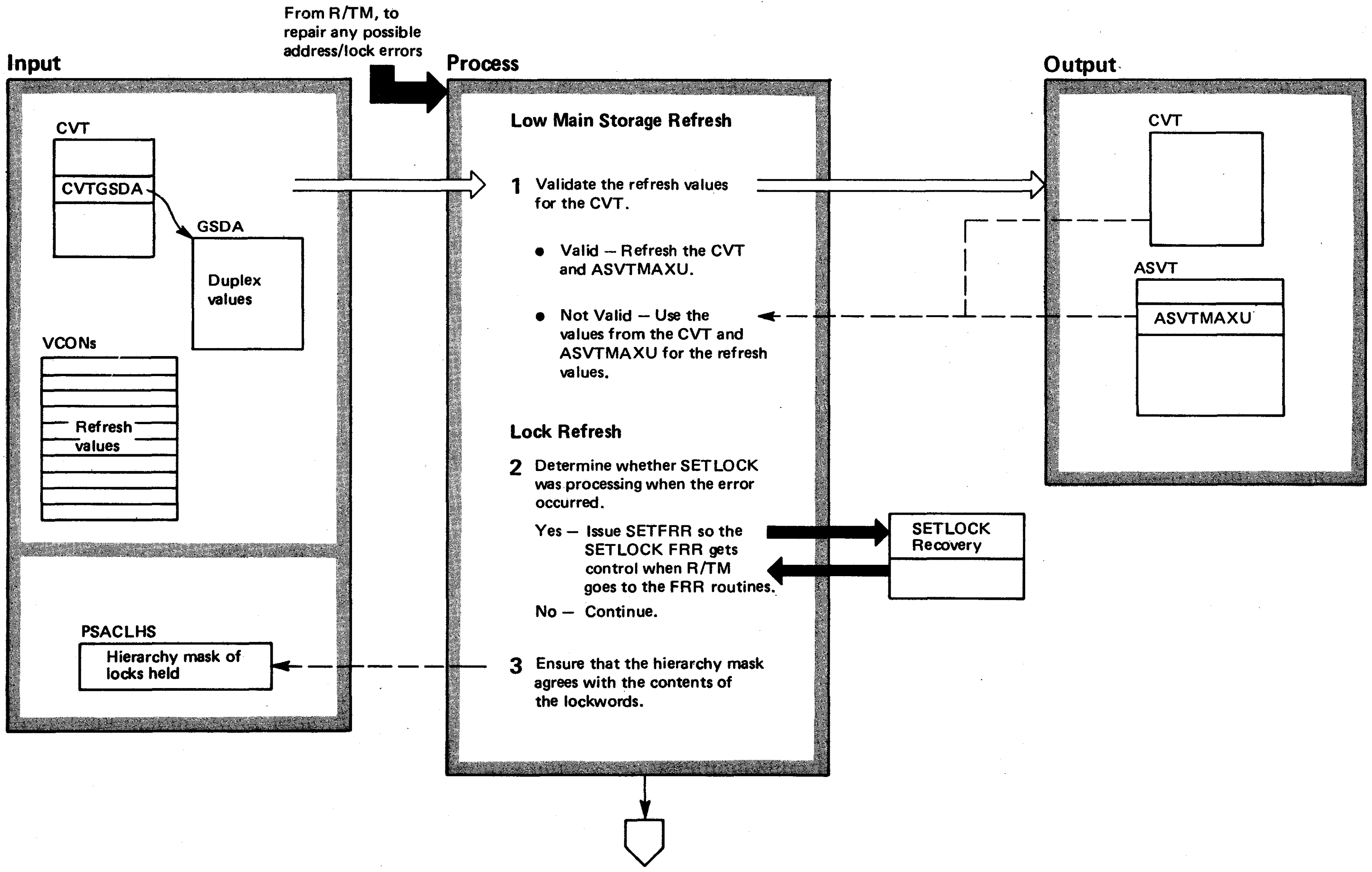
6 Process recursion.
- 1st recursion
  - Clear indicators.
  - Request an address space termination.
- 2nd recursion
  - Terminate system with X'01C' wait state code.

IEA967W 'Unsuccessful recovery attempt by Supervisor control'.

Diagram 19-26. Super FRR (IEAVESPR) (Part 4 of 4)

Extended Description	Module	Label
5 The Super FRR purges the translation lookaside buffers via a PTLB (purge translation lookaside buffers) instruction. Control returns to R/TM when the PTLB operation completes.		
6 For one recursion, the Super FRR terminates the 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 19-27. Address Space/Lock Verification Processing (IEAVELCR) (Part 1 of 4)



Low Main Storage Refresh

1 Validate the refresh values for the CVT.

- Valid — Refresh the CVT and ASVTMAXU.
- Not Valid — Use the values from the CVT and ASVTMAXU for the refresh values.

Lock Refresh

2 Determine whether SETLOCK was processing when the error occurred.

Yes — Issue SETFRR so the SETLOCK FRR gets control when R/TM goes to the FRR routines.

No — Continue.

3 Ensure that the hierarchy mask agrees with the contents of the lockwords.

Diagram 19-27. Address Space/Lock Verification Processing (IEAVELCR) (Part 2 of 4)

Extended Description	Module	Label
Address/lock verification processing consists of 3 modules that correct errors in the addressing/locking mechanism. The modules are entered from R/TM on every error before any recovery routine receives control.		
1 The low main storage refresh routine replaces the current, and possibly inaccurate, values in the CVT and ASVT with accurate, valid values from VCONs, and duplex values in the GSDA (global system duplex area). If the refresh values are not accurate, the low main storage refresh routine uses copies of the CVT and ASVT values, to refresh the GSDA and VCONs.	IEAVELCR	IEAVELCR
2 The lock refresh subroutine first determines whether an error occurred during SETLOCK processing. If so, a SETFRR is issued so the SETLOCK FRR will get control when R/TM goes to the FRR routines. Otherwise, normal processing continues.	IEAVELKR	IEAVELKR
3 The lock refresh subroutine ensures that the hierarchy mask – the mask that shows the sequence of locks held – agrees with the value in the lockwords. If it does not agree, the lock refresh subroutine will ensure the agreement. The subroutine may also seize the CMS lock if it determines that the owner of the CMS lock was suspended because of a page fault which was never resolved. Finally, the subroutine terminates the address space that owned the CMS lock.		

Diagram 19-27. Address Space/Lock Verification Processing (IEAVELCR) (Part 3 of 4)

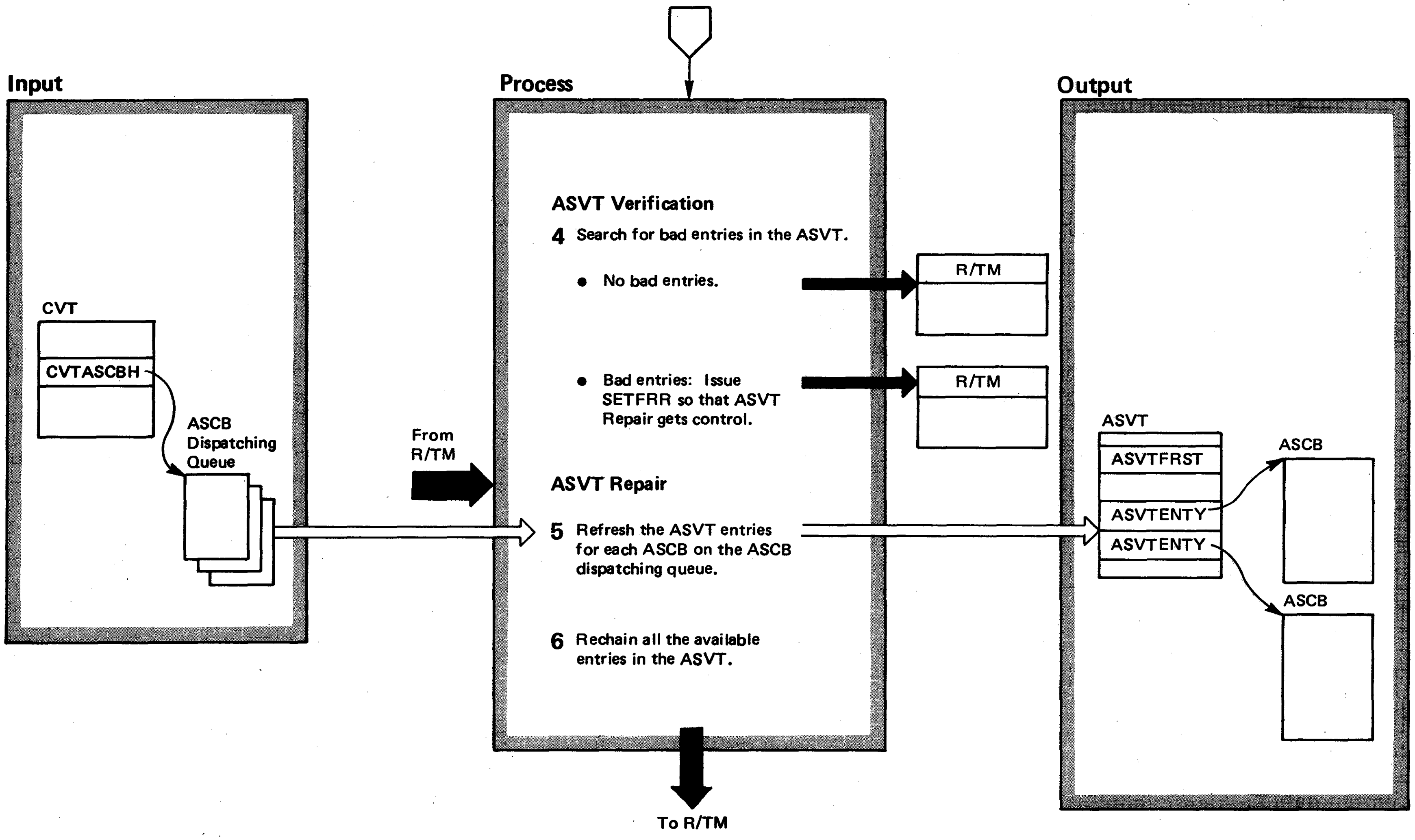
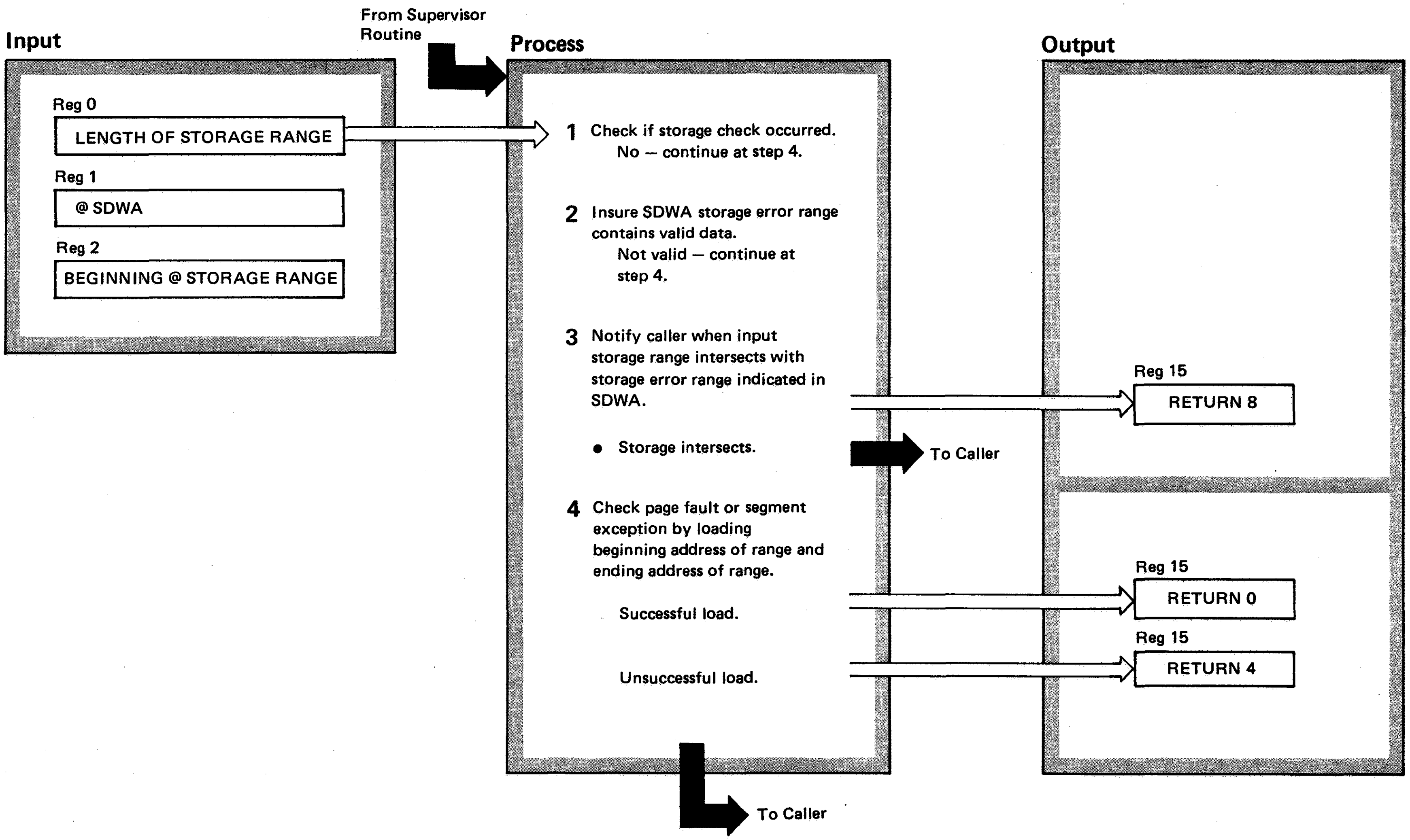

Input
CVT
CVTASCBH
ASCB Dispatching Queue
From R/TM
Process
ASVT Verification
4 Search for bad entries in the ASVT.
• No bad entries.
• Bad entries: Issue SETFRR so that ASVT Repair gets control.
ASVT Repair
5 Refresh the ASVT entries for each ASCB on the ASCB dispatching queue.
6 Rechain all the available entries in the ASVT.
To R/TM
R/TM
R/TM
Output
ASVT
ASVTFRST
ASVTENTY
ASVTENTY
ASCB
ASCB

Diagram 19-27. Address Space/Lock Verification Processing (IEAVELCR) (Part 4 of 4)

Extended Description	Module	Label
4 The ASVT (address space vector table) verification routine searches for invalid ASVT entries. If the routine finds no bad entries, control returns to R/TM. Otherwise, if bad entries are found, a SETFRR is issued so the ASVT repair routine will receive control later.	IEAVEVRR	IEAVEVRR
5 The ASVT repair routine refreshes the entries in the ASVT for each address space that is on the dispatching queue.	IEAVEVRR	IEAVVFRR
6 ASVT repair chains the available entries in the ASVT to show which ASIDs have not been assigned to any address space.		

Diagram 19-28. Address Verification (IEAVEADV) (Part 1 of 2)



1 Check if storage check occurred.
No — continue at step 4.

2 Insure SDWA storage error range contains valid data.
Not valid — continue at step 4.

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

- Storage intersects.

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

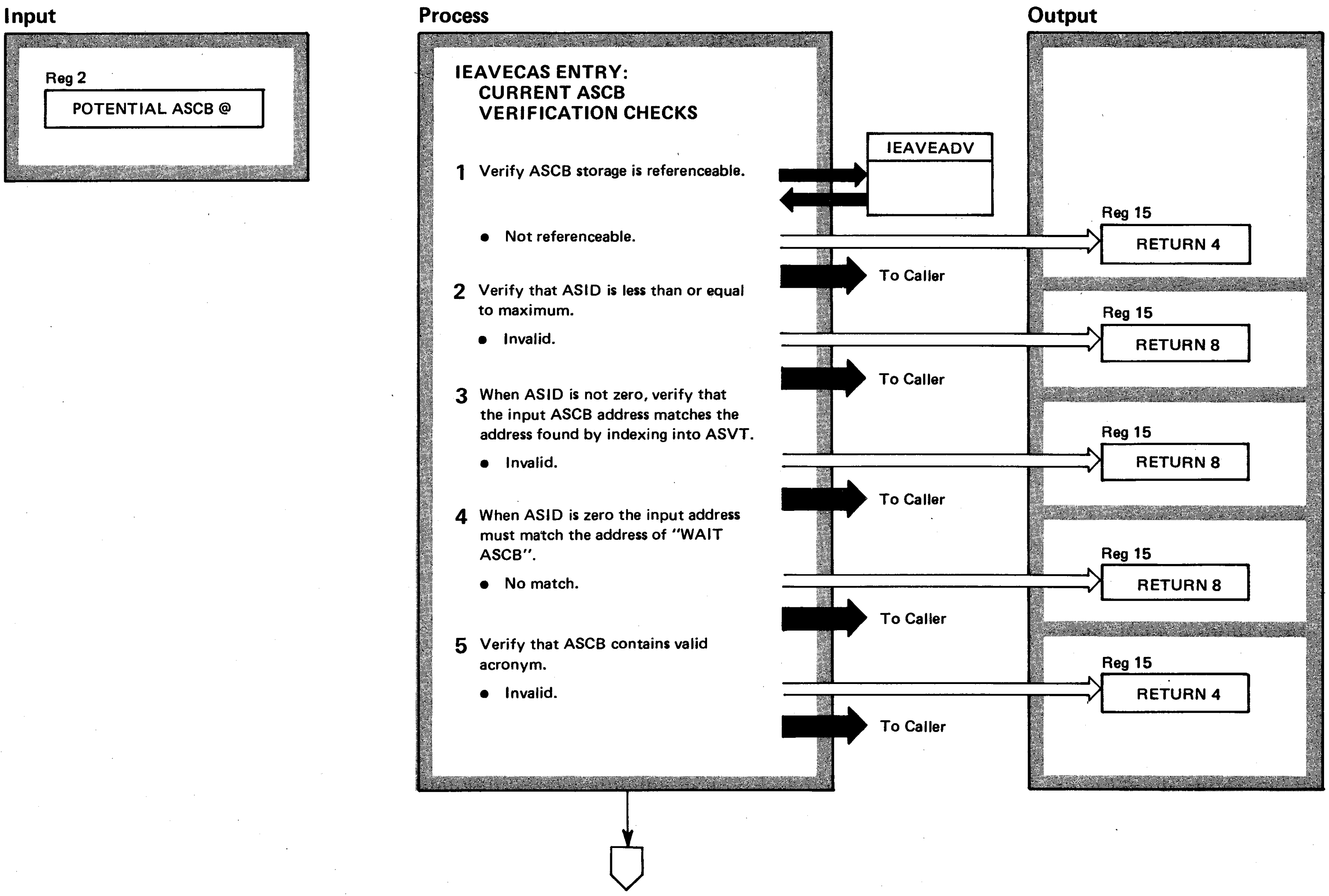
Successful load.

Unsuccessful load.

Diagram 19-28. 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 19-29. Control Block Verification Routine (IEAVECBV) (Part 1 of 10)



IEAVECAS ENTRY:
CURRENT ASCB
VERIFICATION CHECKS

1 Verify ASCB storage is referenceable.

- Not referenceable.

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

- Invalid.

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

- Invalid.

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

- No match.

5 Verify that ASCB contains valid acronym.

- Invalid.

Diagram 19-29. 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, or 4) TCB.

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

- Referenceable potential ASCB storage.
- ASID $\leq$ maximum.
- When ASID $\neq$ 0, input address matches the address found by indexing into ASVT.
- When ASID = 0, input address must match the address of "WAIT ASCB".
- Valid acronym (ASCB).
- Referenceable and valid SPL address.
- Referenceable ASXB.
- ASXB must have valid acronym, referenceable IHSA, and referenceable local work/save area vector table.

A return code of 0 indicates valid control block.

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

A return code of 8 indicates not a control block.

Diagram 19-29. Control Block Verification Routine (IEAVECBV) (Part 3 of 10)

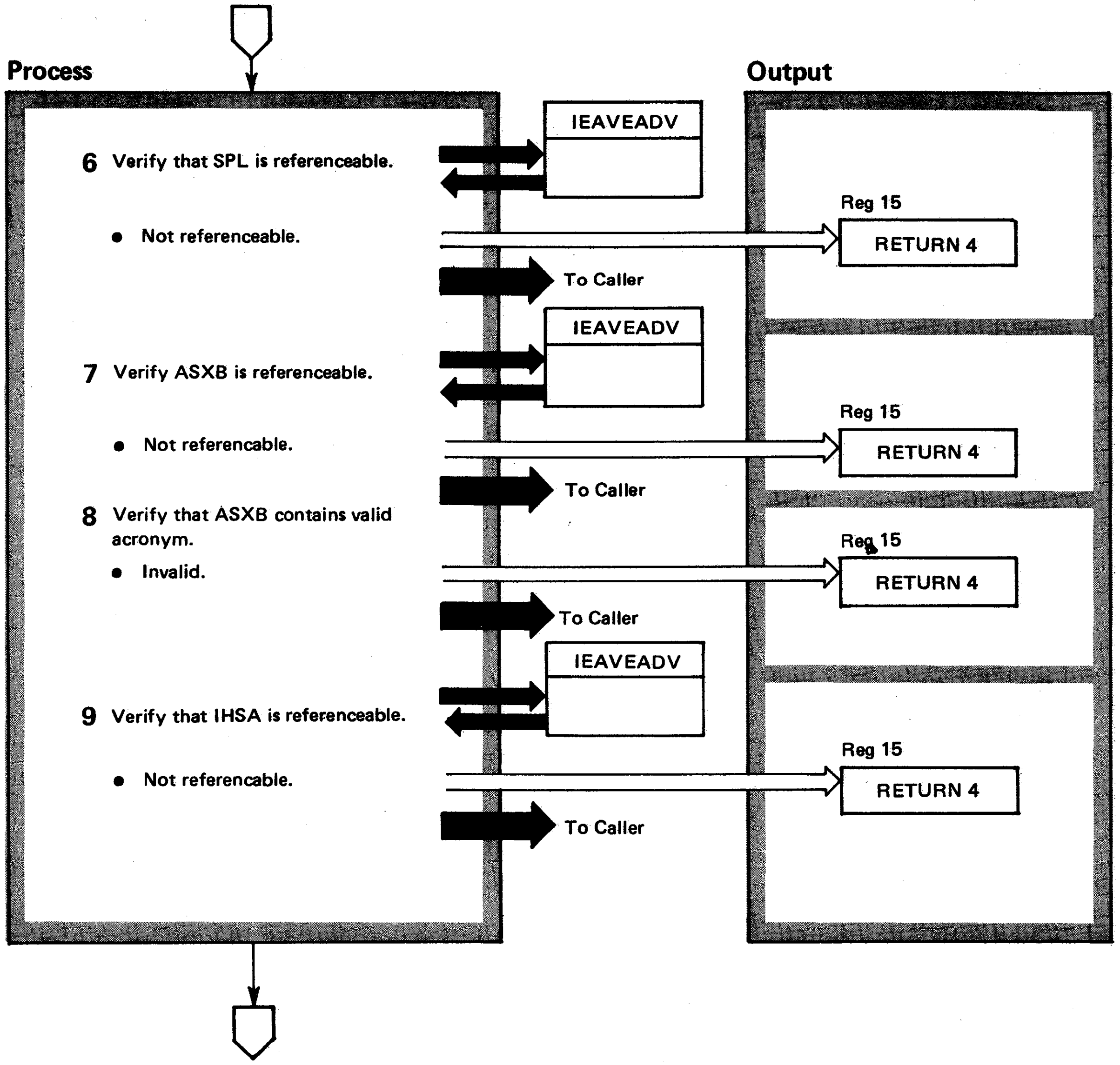


Diagram 19-29. Control Block Verification Routine (IEAVECBV) (Part 4 of 10)

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

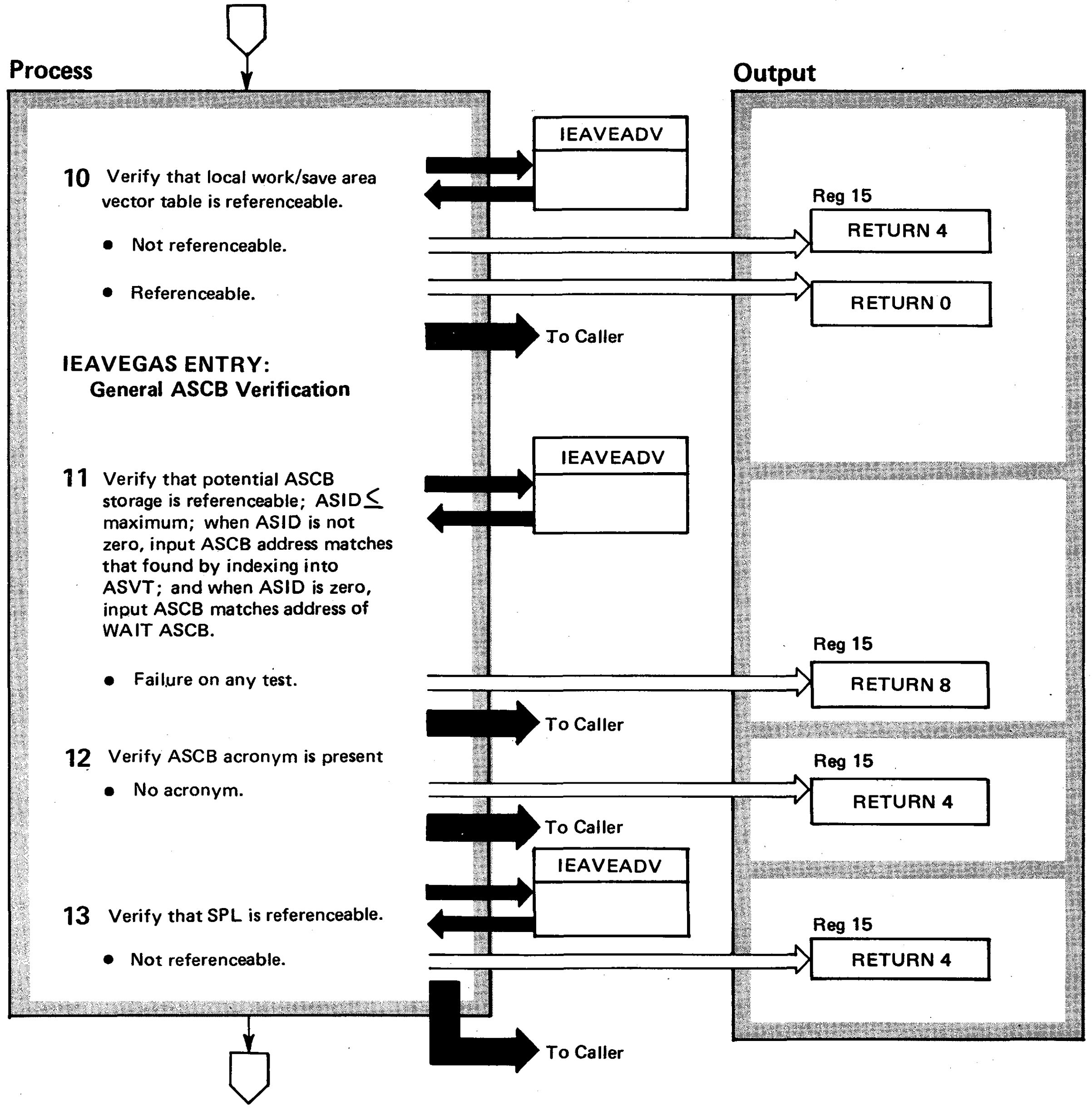
- Referenceable potential ASCB storage.
- ASID $\leq$ maximum.
- When ASID $\neq$ 0, input address matches the address found by indexing into ASVT.
- When ASID = 0, input address must match the address of "WAIT ASCB".
- Valid acronym (ASCB).
- Referenceable and valid SPL address.
- Referenceable ASXB.
- ASXB must have valid acronym, referenceable IHSA, and referenceable local work/save area vector table.

A return code of 0 indicates valid control block.

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

A return code of 8 indicates not a control block.

Diagram 19-29. Control Block Verification Routine (IEAVECBV) (Part 5 of 10)



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

- Not referenceable.
- Referenceable.

IEAVEGAS ENTRY: General ASCB Verification

11 Verify that potential ASCB storage is referenceable; ASID $\leq$ maximum; when ASID is not zero, input ASCB address matches that found by indexing into ASVT; and when ASID is zero, input ASCB matches address of WAIT ASCB.

- Failure on any test.

12 Verify ASCB acronym is present

- No acronym.

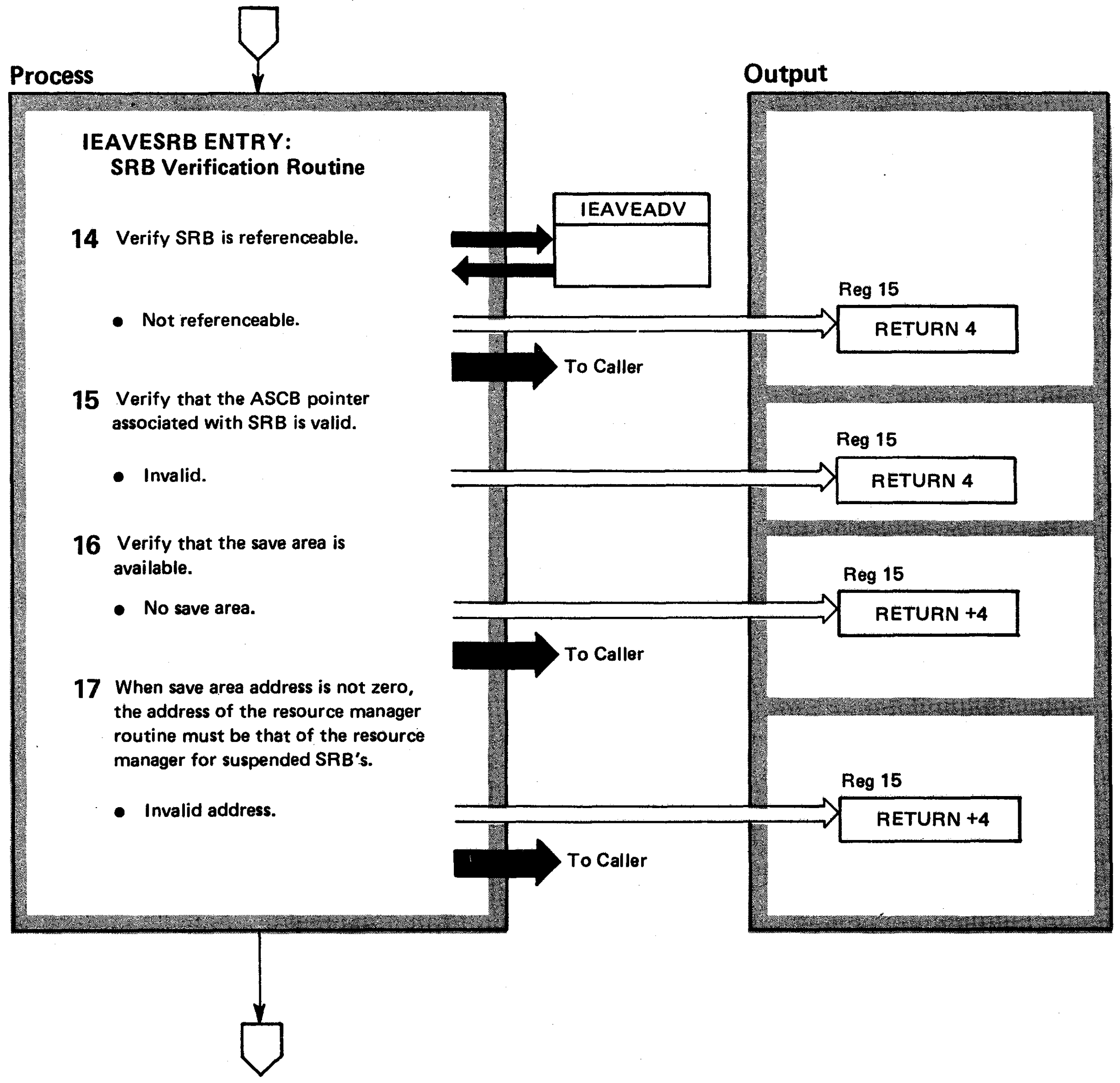
13 Verify that SPL is referenceable.

- Not referenceable.

Diagram 19-29. Control Block Verification Routine (IEAVECBV) (Part 6 of 10)

Extended Description		Module	Label
10	See the extended description for steps 6-9.		
11-13	For general ASCB verification (IEAVEGAS), the input address must pass the first six criteria listed under current ASCB verification. Return codes indicate same conditions.		

Diagram 19-29. Control Block Verification Routine (IEAVECBV) (Part 7 of 10)



Process

IEAVESRB ENTRY: SRB Verification Routine

14 Verify SRB is referenceable.

- Not referenceable.

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

- Invalid.

16 Verify that the save area is available.

- No save area.

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

- Invalid address.

Diagram 19-29. 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 save area address $\neq$ 0, the address of the resource manager routine must be that of the resource manager for suspended SRB's. • When save address = 0, routine entry point address must be non-zero. Return codes indicate same conditions as indicated under current ASCB verification.		

Diagram 19-29. Control Block Verification Routine (IEAVECBV) (Part 9 of 10)

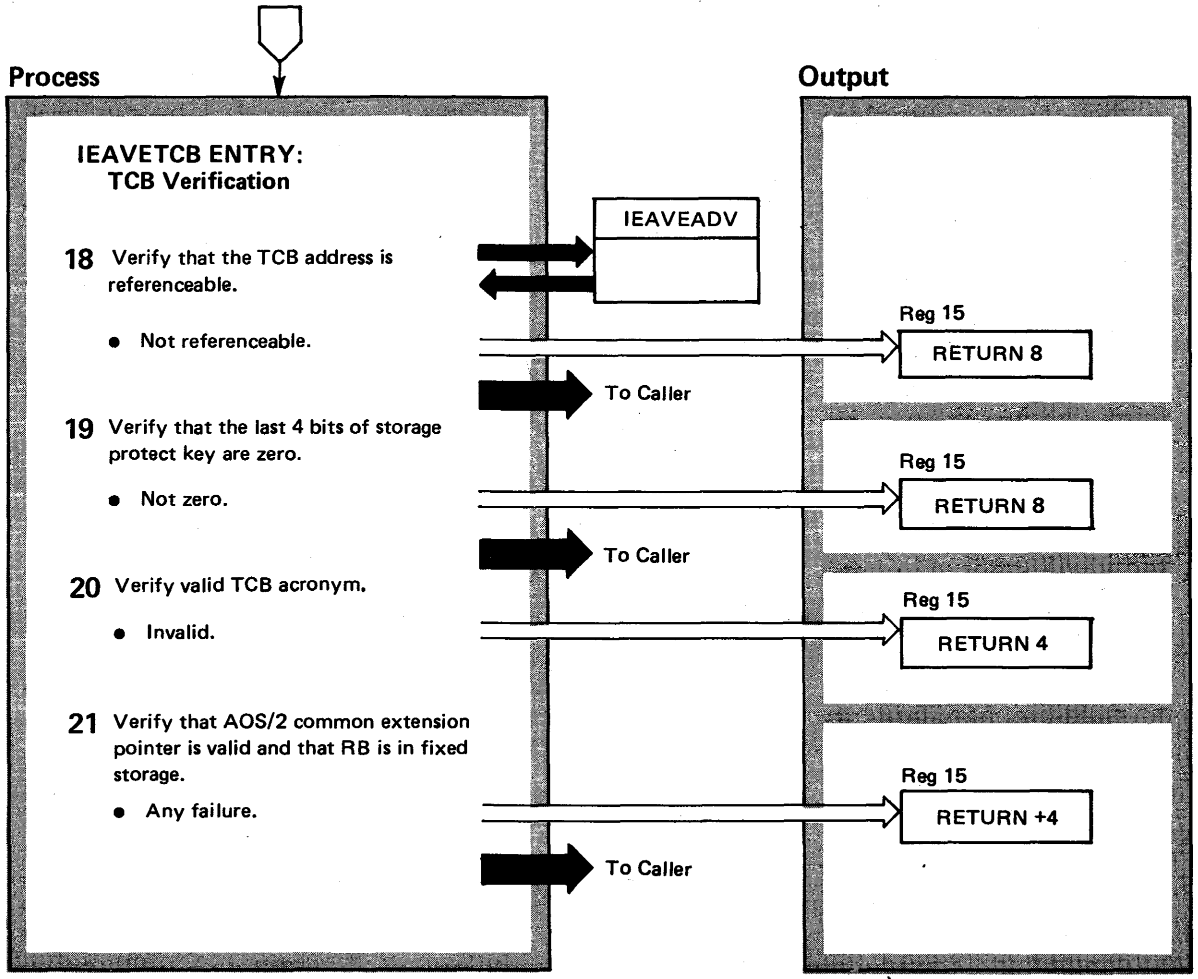

Process
Output
IEAVETCB ENTRY:
TCB Verification
18 Verify that the TCB address is referenceable.
• Not referenceable.
19 Verify that the last 4 bits of storage protect key are zero.
• Not zero.
20 Verify valid TCB acronym.
• Invalid.
21 Verify that AOS/2 common extension pointer is valid and that RB is in fixed storage.
• Any failure.
IEAVEADV
To Caller
To Caller
To Caller
Reg 15
RETURN 8
Reg 15
RETURN 8
Reg 15
RETURN 4
Reg 15
RETURN +4

Diagram 19-29. Control Block Verification Routine (IEAVECBV) (Part 10 of 10)

Extended Description	Module	Label
18-21 For TCB verification, the following criteria must be met:		

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

Return codes same as for current ASCB verification routine.

Diagram 19-28. Suspend Routine (IEAVETCL) (Part 1 of 2)

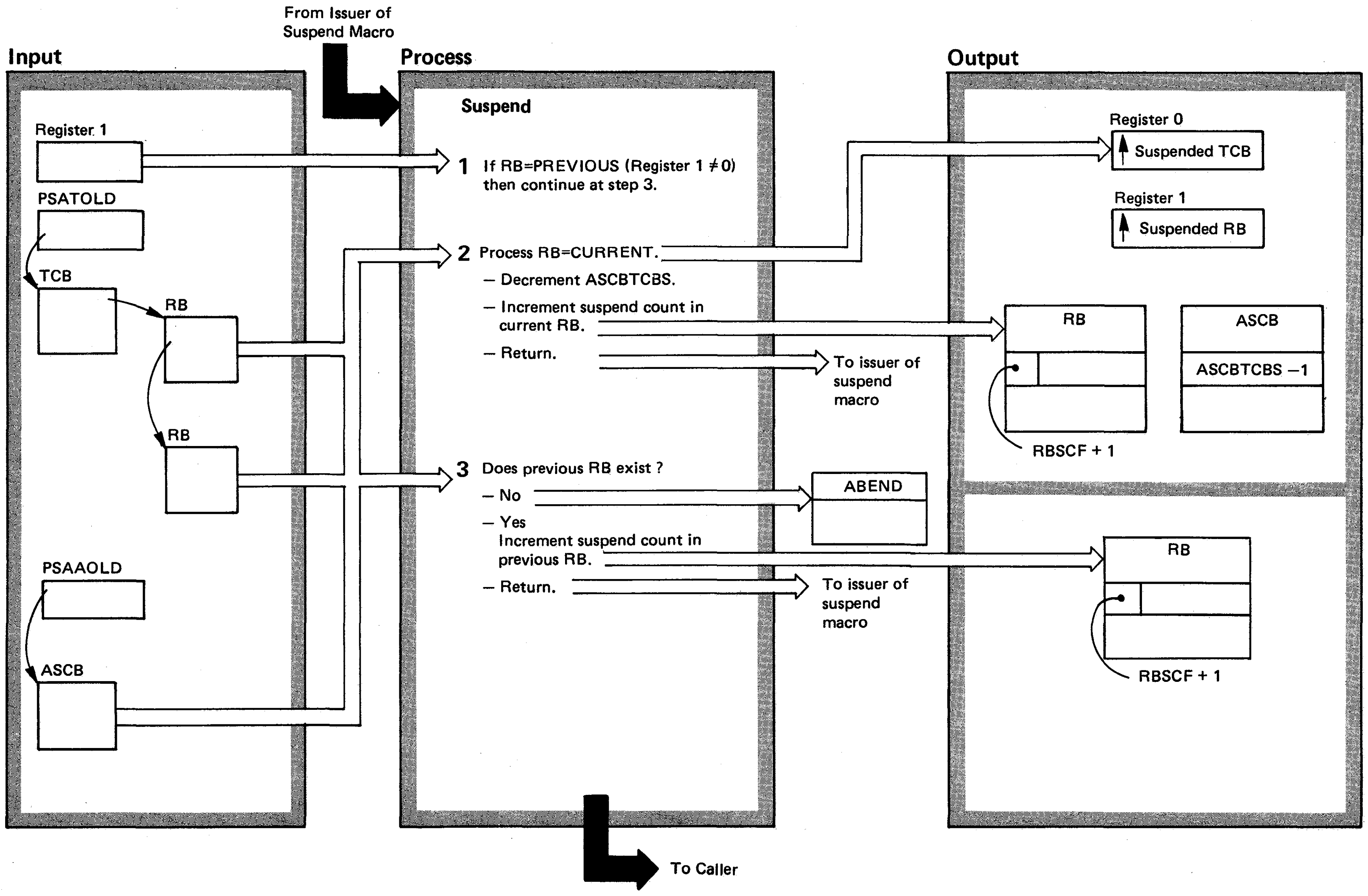
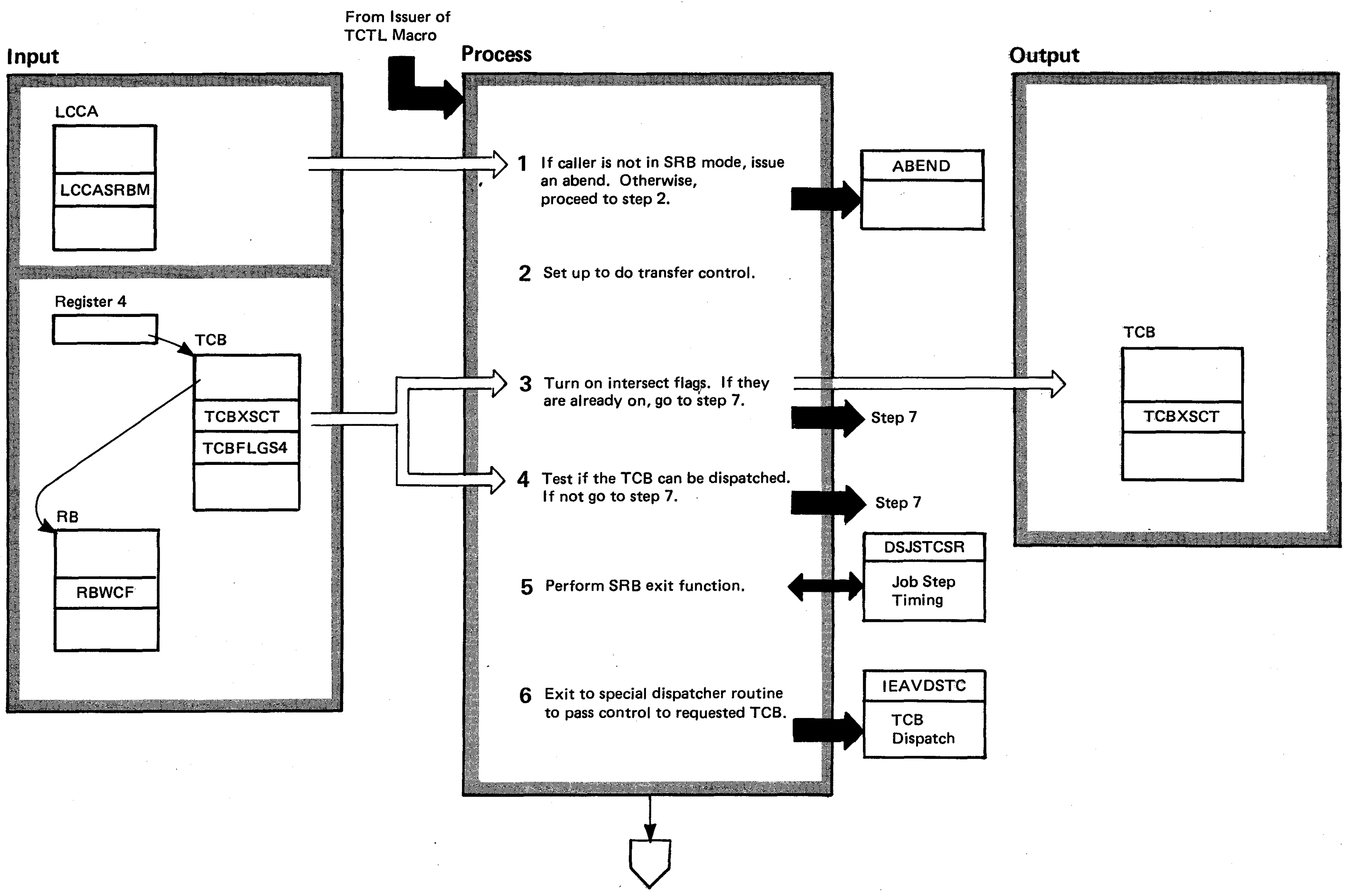

From Issuer of Suspend Macro
Input
Register 1
PSATOLD
TCB
RB
RB
PSAAOLD
ASCB
Process
Suspend
1 If RB=PREVIOUS (Register 1 ≠ 0) then continue at step 3.
2 Process RB=CURRENT.
– Decrement ASCBTCBS.
– Increment suspend count in current RB.
– Return.
To issuer of suspend macro
3 Does previous RB exist ?
– No
– Yes Increment suspend count in previous RB.
– Return.
To issuer of suspend macro
ABEND
To Caller
Output
Register 0
Suspended TCB
Register 1
Suspended RB
RB
RBSCF + 1
ASCB
ASCBTCBS –1
RB
RBSCF + 1

Diagram 19-28. Suspend Routine (IEAVETCL) (Part 2 of 2)

Extended Description	Module	Label
The Suspend routine (IEAVETCL) is a fast means for placing a TCB in the wait state.		
1 Suspend checks the contents of register 1. If it is nonzero, then RB=PREVIOUS was requested and processing continues at step 3. If it is zero, then RB=CURRENT was requested.	IEAVETCL	IEAVSUSP
2 Since the TCB will no longer be dispatchable, the count of ready TCBs in the current ASCB must be decremented by one (ASCBTCBS). Next, the suspend count in the current RB (RBSCF) is incremented by one to place the task in the suspended state. Registers 0 and 1 are initialized with the TCB and RB addresses, respectively, and control is returned to the caller.		
3 If no previous RB exists, the caller is terminated with an abend code of 070 and register 15 is zeroed. Otherwise, the previous RB is obtained and the RBSCF field is incremented by one. Since this is done in the previous RB, the ability to dispatch the task is not changed. Registers 0 and 1 are initialized with the TCB and RB addresses respectively, and control is returned to the caller.		PREVIOUS

Diagram 19-29. Transfer Control – Transfer Logical (TCTL) (IEAVETCL) (Part 1 of 4)



1 If caller is not in SRB mode, issue an abend. Otherwise, proceed to step 2.

2 Set up to do transfer control.

3 Turn on intersect flags. If they are already on, go to step 7.

4 Test if the TCB can be dispatched. If not go to step 7.

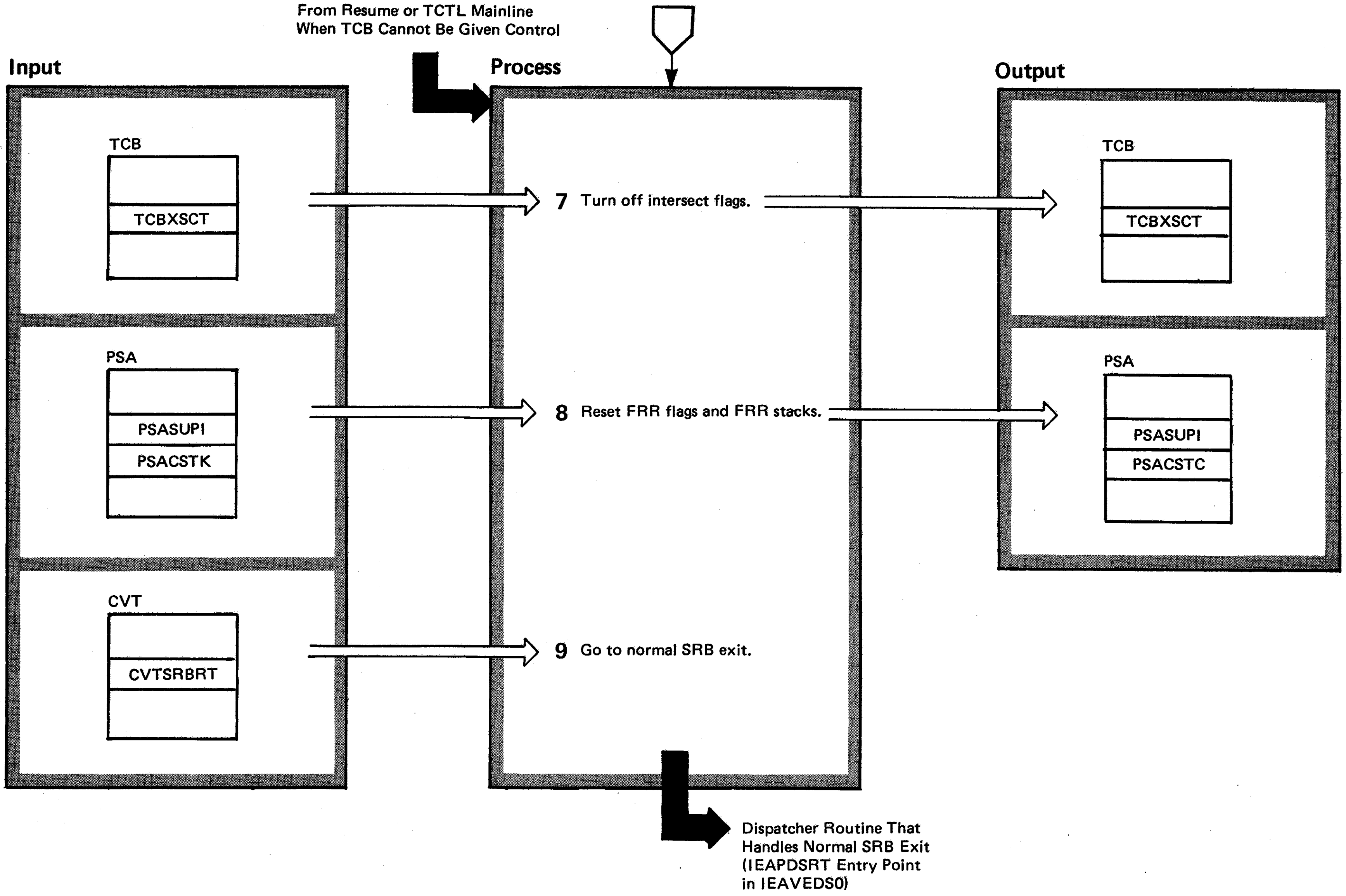
5 Perform SRB exit function.

6 Exit to special dispatcher routine to pass control to requested TCB.

Diagram 19-29. Transfer Control – Transfer Logic (TCTL) (IEAVETCL) (Part 2 of 4)

Extended Description	Module	Label
1 A transfer control – transfer logical (TCTL) can only be issued by an SRB routine. If the caller is not in SRB mode, it is terminated.	IEAVTCTL	IEAVTCTL
2 The following is done to set up for the transfer of control: • Disable I/O and external interrupts. • Set up super FRR. • Turn on CDALTCTL to indicate transfer control is active. • If status is active (ASCBSTA=ON), go to the normal SRB exit.		
3 Turn on the intersect flags in the TCB via a compare and swap. If already on, go to the normal SRB exit.		
4 If the TCB cannot be dispatched, go to the normal SRB dispatcher. (To test whether the TCB can be dispatched, check to see that TCBFLGS4=0, TCBFLGS5=0, RBWCF=0, and ASCBSTND=0. If one or more are not zero, the TCB cannot be dispatched.)		
5 Perform the following SRB functions: • Call job step timing. • Turn off SRB mode flag. • Decrement count of SRBs by one. • Increment count of CPUs by one.		
6 A special entry in the dispatcher accepts a TCB address as input and passes control to that TCB.		

Diagram 19-29. Transfer Control – Transfer Logical (TCTL) (IEAVETCL) (Part 3 of 4)

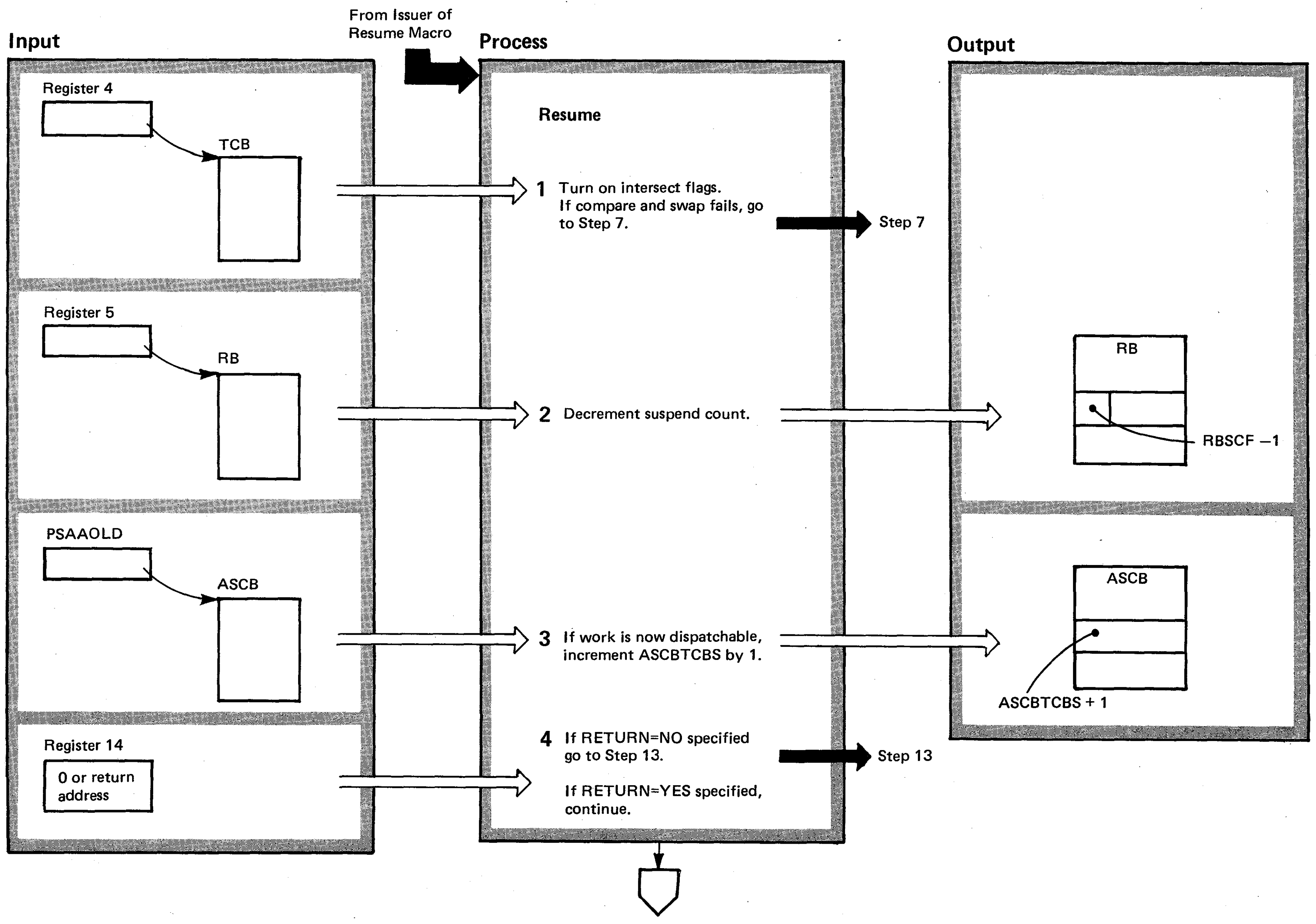


From Resume or TCTL Mainline When TCB Cannot Be Given Control
Input
Process
Output
TCB
TCBXSCT
PSA
PSASUPI
PSACSTK
CVT
CVTSRBRT
7 Turn off intersect flags.
8 Reset FRR flags and FRR stacks.
9 Go to normal SRB exit.
TCB
TCBXSCT
PSA
PSASUPI
PSACSTC
Dispatcher Routine That Handles Normal SRB Exit (IEAPDSRT Entry Point in IEAVEDSO)

Diagram 19-29. Transfer Control – Transfer Logic (TCTL)(IEAVETCL) (Part 4 of 4)

Extended Description	Module	Label
7 Intersect flag TCBACTIV is turned off.		TCTL003
8 Flags PSADISP and PSATCTL are turned off and the current FRR stack (PSACSTK) is set to normal (PSANSTK).		
9 The normal SRB exit routine (IEAPDSRT) in the dispatcher is called.		

Diagram 19-30. Resume Routine (IEAVETCL) (Part 1 of 6)



1 Turn on intersect flags. If compare and swap fails, go to Step 7.

2 Decrement suspend count.

3 If work is now dispatchable, increment ASCBTCBS by 1.

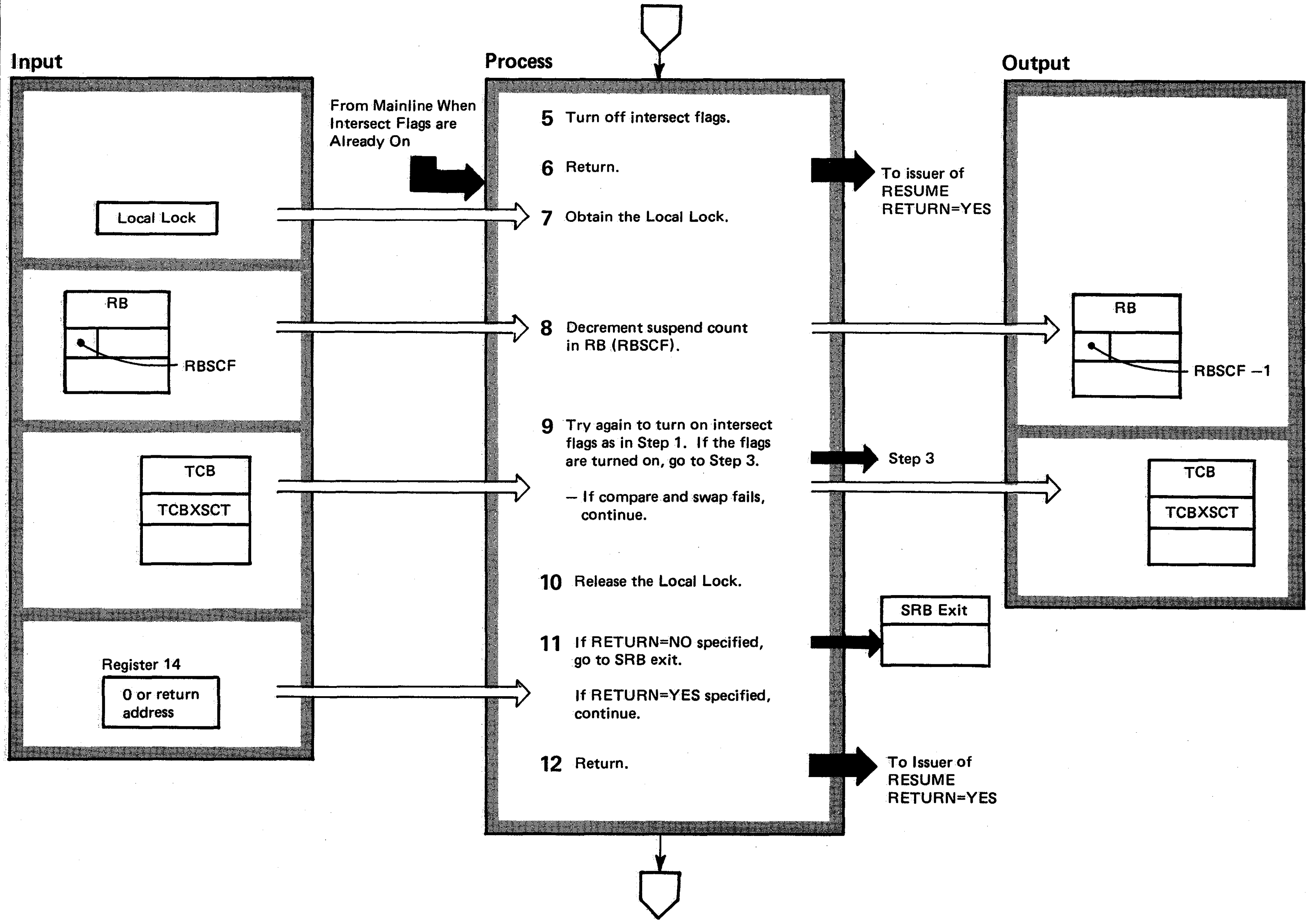
4 If RETURN=NO specified go to Step 13.

If RETURN=YES specified, continue.

Diagram 19-30. Resume Routine (IEAVETCL) (Part 2 of 6)

Extended Description	Module	Label
1 Turn on TCB intersect flags (TCBS3A and TCBACT) via a compare and swap. (If TCBS3A=1, the stage 3 exit effector is locked out. If TCBACT=1, the dispatcher is locked out.) If an intersect flag is already on, the local lock must be acquired after branching to step 7.	IEAVETCL	IEAVRSME
2 The suspend count in the RB (RBSCF) is decremented by one.		
3 If the Resume was for the top RB and the unit of work is not dispatchable, the count of ready TCBs (ASCBTCBS) must be incremented by one.		
4 If register 14 is zero, RETURN=NO was requested. Therefore, Resume will attempt to do a TCTL to the resumed TCB at step 13. If register 14 is a return address, Resume will not do a TCTL to the resumed TCB.		

Diagram 19-30. Resume Routine (IEAVETCL) (Part 3 of 6)


Input
Process
Output
From Mainline When Intersect Flags are Already On
Local Lock
RB
RBSCF
TCB
TCBXSCT
Register 14
0 or return address
To issuer of RESUME RETURN=YES
RB
RBSCF −1
Step 3
TCB
TCBXSCT
SRB Exit
To Issuer of RESUME RETURN=YES

5 Turn off intersect flags.

6 Return.

7 Obtain the Local Lock.

8 Decrement suspend count in RB (RBSCF).

9 Try again to turn on intersect flags as in Step 1. If the flags are turned on, go to Step 3.

– If compare and swap fails, continue.

10 Release the Local Lock.

11 If RETURN=NO specified, go to SRB exit.

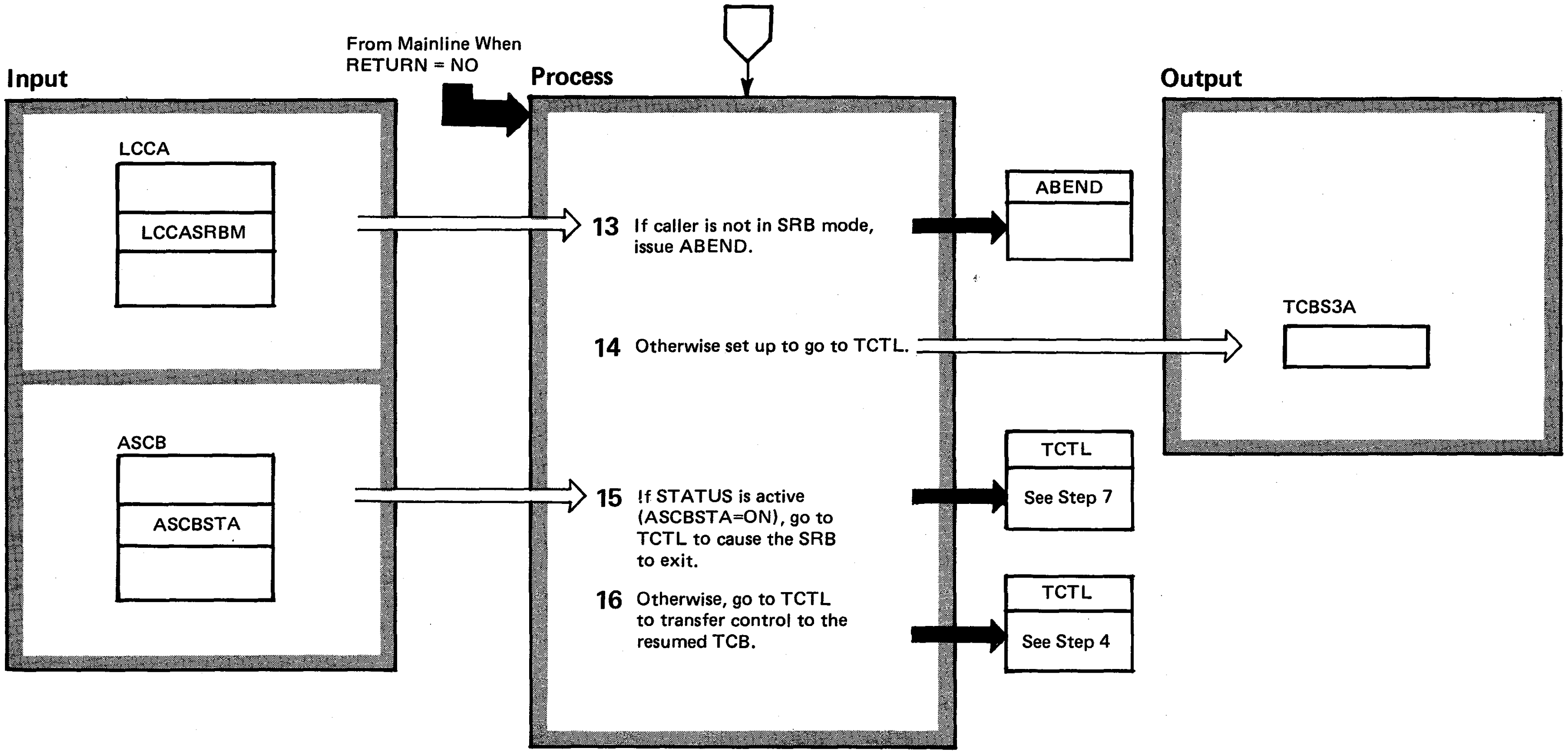
If RETURN=YES specified, continue.

12 Return.

Diagram 19-30. Resume Routine (IEAVETCL) (Part 4 of 6)

Extended Description	Module	Label
5 The intersect flags turned on in step 1 are turned off.		
6 Return to the caller.		
7 If the TCB was active or the stage 3 exit effector was active for this TCB, the local lock is acquired.	GETLOCK	GETLOCK
8 The suspend count in the RB (RBSCF) is decremented by one.		
9 A second attempt is made to turn on the intersect flags. If successful, control goes back to normal mainline processing.		
10 Otherwise, the local lock is released.		
11 If register 14 is zero (RETURN=NO was specified), control is passed to the normal SRB routine. If the caller was not in SRB mode, he is terminated. If register 14 is not zero (RETURN=YES was specified or implied) then go to the next step.		
12 Return to the caller.		

Diagram 19-30. Resume Routine (IEAVETCL) (Part 5 of 6)



From Mainline When RETURN = NO
Input
Process
Output
LCCA
LCCASRBM
ASCB
ASCBSTA
13 If caller is not in SRB mode, issue ABEND.
14 Otherwise set up to go to TCTL.
15 If STATUS is active (ASCBSTA=ON), go to TCTL to cause the SRB to exit.
16 Otherwise, go to TCTL to transfer control to the resumed TCB.
ABEND
TCTL
See Step 7
TCTL
See Step 4
TCBS3A

Diagram 19-30. Resume Routine (IEAVETCL) (Part 6 of 6)

Extended Description	Module	Label
13 Only SRBs are allowed to specify RESUME RETURN=NO. If the caller was not in SRB mode, it is terminated.		RESU006
14 The following is done to be able to enter TCTL at a special internal entry: ● Turn off TCBS3A. ● Disable I/O and external interrupts. ● Set up super FRR. ● Turn on DCALTCTL to indicate transfer control is active.		
15 If STATUS is active, TCTL is entered at a point that will cause normal SRB exit to occur.		
16 If STATUS is not active, TCTL is entered to transfer control to the resumed TCB.		

Task Management performs services for both problem and system programs. These services fall into three catagories: creating and deleting subtasks, controlling the execution of tasks in one or more address spaces, and providing informational services for the requester.

Creating and deleting subtasks consists of the following services:

- Creating a new subtask. The requester issues an ATTACH macro instruction to perform this service.
- Terminating or deleting a subtask. The requester issues a DETACH macro instruction to perform this service.

Controlling the execution of tasks in one or more address spaces consists of the following services:

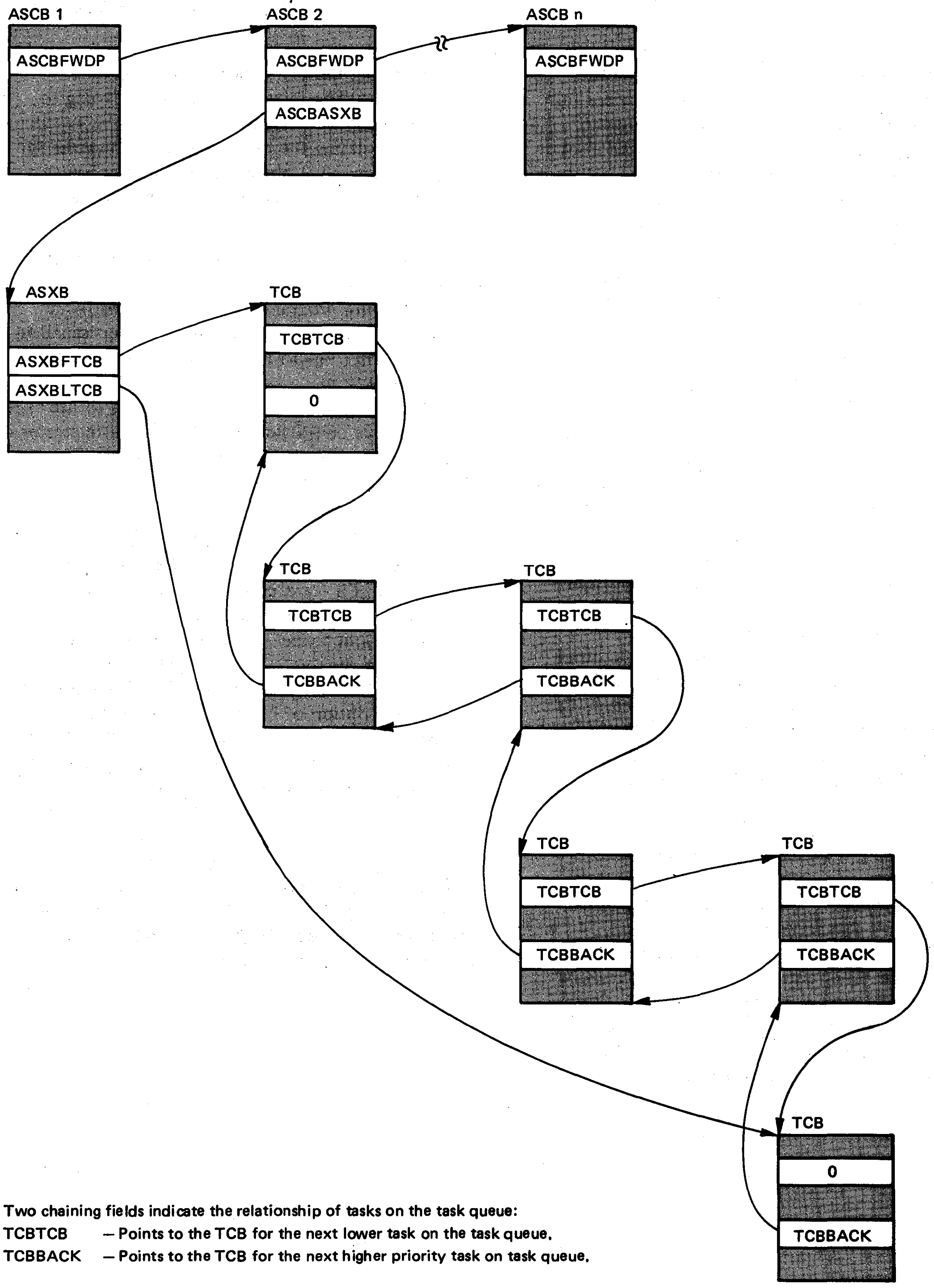
- Changing the dispatching priority of a subtask. The requester issues a CHAP macro instruction to perform this service.
- Allowing a program to stop executing until a specified event or number of events occur. The requester issues a WAIT macro instruction to perform this service.
- Allowing a program to stop executing until one of n events completes and be directly informed which events have completed. The requestor issues a sequence of EVENTS macro instructions to perform this service.
- Signifying the completion of an event. The requester issues a POST macro instruction to perform this service.
- Providing a serialization mechanism for a resource or resources. The requester issues ENQ, DEQ, or RESERVE macro instructions to perform this service.
- Specifying a program check interruption routine. The requester issues a SPIE macro instruction to perform this service.
- Handling the exiting procedures for programs other than type 1 SVCs. The requester issues an EXIT SVC to perform this service.
- Handling the exit procedures for SVC routines. The requester uses EXIT Prolog to perform this service.
- Manipulating the dispatchability indicators of system control blocks. The requester issues a STATUS macro instruction to perform this service.

Providing informational services consists of the following services:

- Providing programs with information from system control blocks. The requester issues an EXTRACT macro instruction to perform this service.
- Verifying routines for authorization to use sensitive or privileged routines. The requester issues a TESTAUTH macro instruction to perform this service.

Creating and Deleting Subtasks

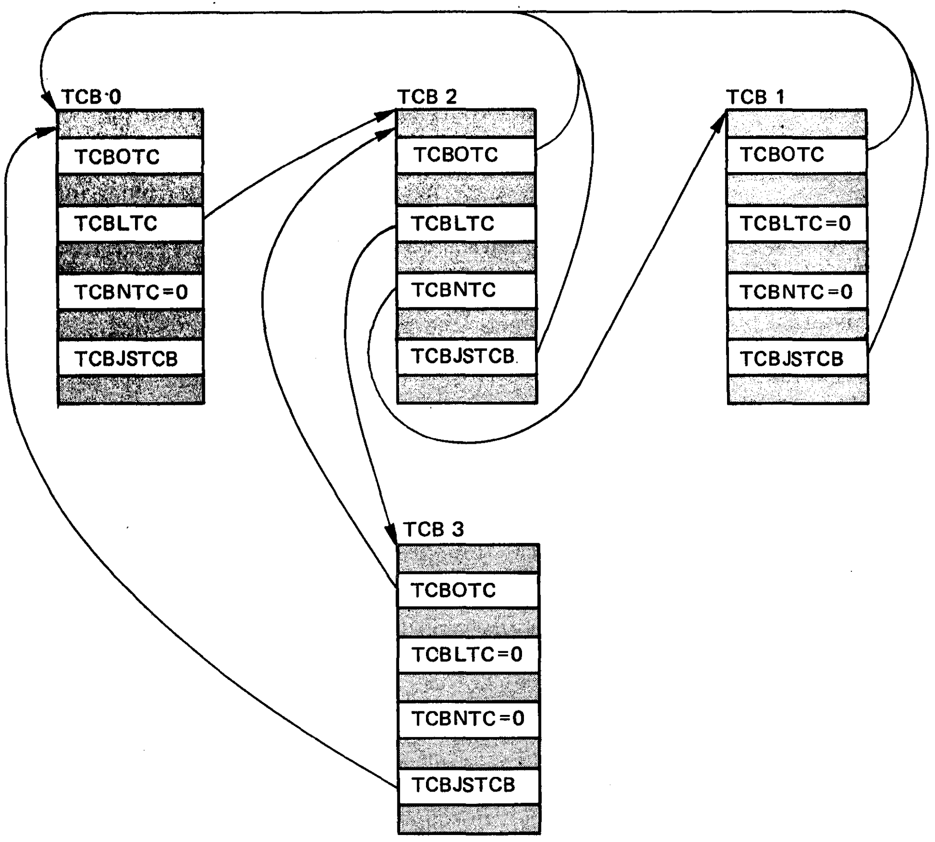
Services related to creating and deleting subtasks involve the TCB (task control block). When a problem or system program issues an ATTACH macro instruction, the ATTACH routine receives control from the SVC IH (interruption handler) and creates a TCB. (See Supervisor Control, section 19, for the description of interruption handling). ATTACH then places the newly created TCB on the TCB ready queue in the appropriate address space, according to the priority written on the ATTACH macro. Figure 2-37 illustrates the task queue. It shows the relationship between the address space — represented by the ASCB (address space control block) — and the tasks running in it — represented by the TCBs. Figure 2-38 depicts the family subtask queue. It shows the relationship between job step tasks and subtasks.



Two chaining fields indicate the relationship of tasks on the task queue:

TCBTCB	– Points to the TCB for the next lower task on the task queue.
TCBBACK	– Points to the TCB for the next higher priority task on task queue.

Figure 2-37. The TCB Ready Queue



Four chaining fields indicate the relationship of subtasks on a subtask queue:

TCBOTC	– Points to the TCB for the task that attached this subtask.
TCBLTC	– Points to the TCB for the task last attached by this task.
TCBNTC	– Points to the TCB for the task previously attached by the task that attached this task.
TCBJSTCB	– Points to the first TCB for the job step.

Figure 2-38. The TCB Family Queue

After the requester issues a DETACH macro, the DETACH routine receives control from the SVC IH, and removes the pointers from other TCBs to the deleted TCB. This effectively takes the specified TCB from the TCB queue.

Controlling Task Execution

Task management services control task execution directly and indirectly. Direct control of task execution means that the requester uses a task management service to immediately alter the execution of a task. Indirect control of task execution means that the requester uses a task management service to perform a service that alters task execution sometime in the future.

Direct Control of Tasks

Requesters can use the following task management services to alter immediately task execution:

- CHAP
- WAIT
- POST
- STATUS
- MODESET
- EVENTS
- EXIT
- EXIT Prologue.

The CHAP, STATUS, and MODESET services alter the dispatching of tasks. (See the Supervisor Control section for a discussion of task dispatching.) Requesters alter the dispatching of tasks to indicate or to cause changes in task execution. After CHAP receives control from the SVC IH, the CHAP routine replaces the value that represents the dispatching priority in the TCB with the new value that represents the changed dispatching priority. Then, CHAP changes the position in the TCB queue of the TCB to reflect the changed priority. STATUS, after receiving control from the SVC IH, changes dispatchability indicators; and MODESET, after receiving control from the SVC IH, changes the mode or system key of the requester.

The POST and WAIT services operate as a pair to indicate the occurrence of an event to the requester. The WAIT service receives control from the SVC IH, and then indicates a wait condition in an ECB (event control block). The POST service receives control from the SVC IH and "posts" the occurrence of an event in an ECB. POST marks the completion of an event, and WAIT waits for the event. In effect, these two services control task execution by synchronization.

A new service EVENTS has been added to further enhance the synchronization previously provided only by WAIT and POST. EVENTS and POST also operate as a pair to indicate the occurence of an event to the requester. The EVENTS service routine first receives control from the SVC IH through the Extended SVC router to create an events table for the user. Then the EVENTS service routine receives control from the SVC IH to initialize the ECB. The ECB is initialized with the WAIT bit on in the high-order byte; the low-order three bytes contain the event table address, with bit 32 turned on. EVENTS service routine may or may not wait on an EVENT-type event to complete.

EXIT and Exit prolog perform the exiting services for system and user programs.

EXIT performs the exiting procedures for system and user programs; Exit prolog performs the exiting procedures for SVCs.

Indirect Control of Tasks

Requesters can use the following task management services to alter tasks at a later time:

- ENQ/DEQ/RESERVE
- SPIE

The ENQ/DEQ/RESERVE services enable a requester to gain control of the specified resources needed to execute the requester's program. ENQ/DEQ/RESERVER queue requests for resources after receiving control from the SVC IH.

The SPIE service constructs an SCA (SPIE control area) which contains information that enables a task to regain control after a program interruption. (See the Supervisor Control section for a description of interruption types.) SPIE receives control from the SVC IH after a SPIE service request occurs. SPIE constructs the SCA, and sets indicators in the TCB and RB of the requester.

Providing Informational Services

Two task management services, EXTRACT and TESTAUTH, provide requesters with task-related information, such as contents of control blocks, and authorization of requesters. The EXTRACT service enables a requester to extract control information from the TCB, JSCB (job step control block), or CSCB (command scheduling control block) or combinations of those control blocks. EXTRACT receives control from the SVC IH to furnish the specified information for the requester. TESTAUTH ensures that a caller of a supervisor service has the necessary authorization to use the service. (The *Introduction to VS2* discusses authorization.) TESTAUTH receives control from the SVC IH, or after a branch entry from a supervisor routine.

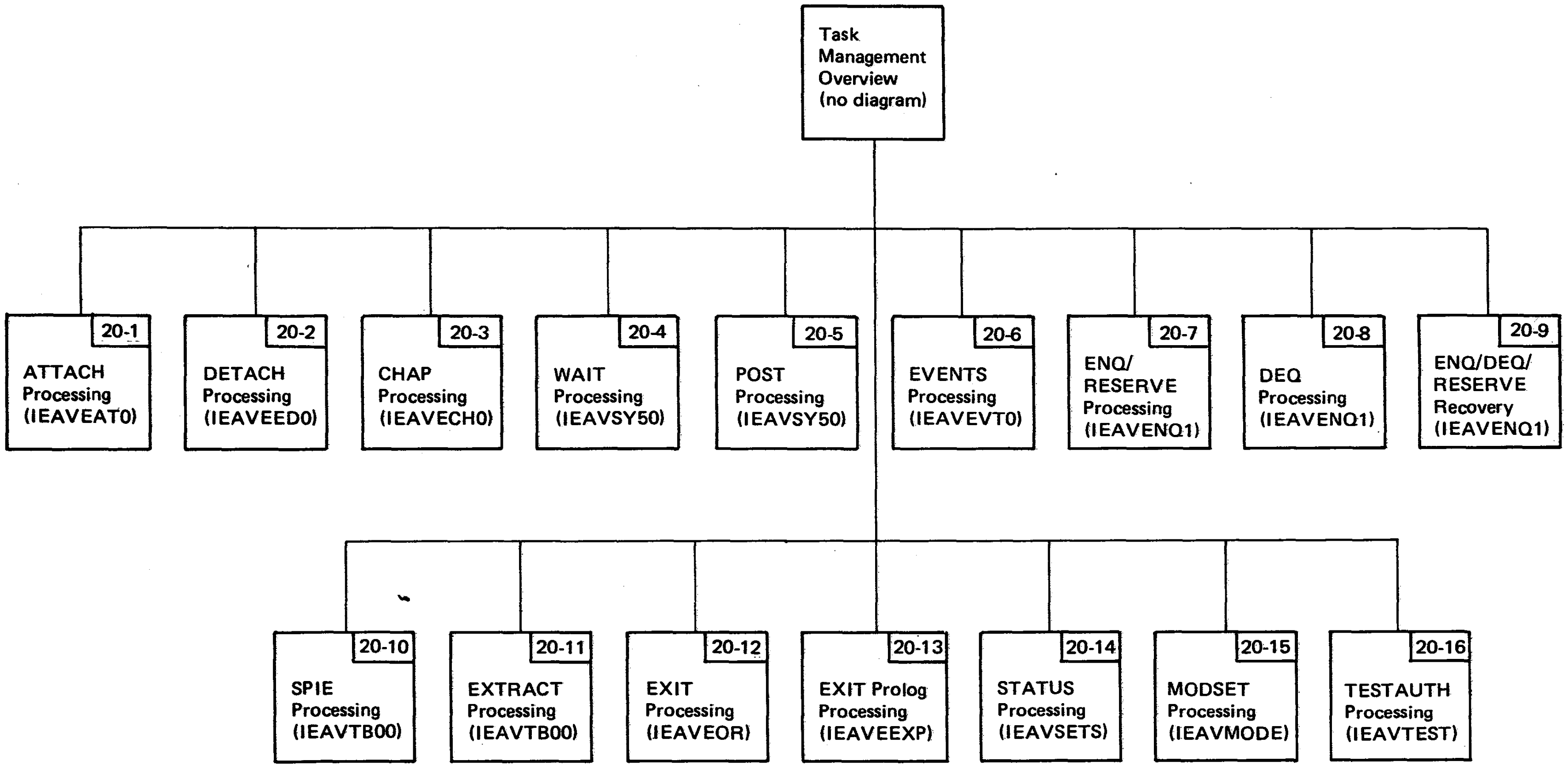
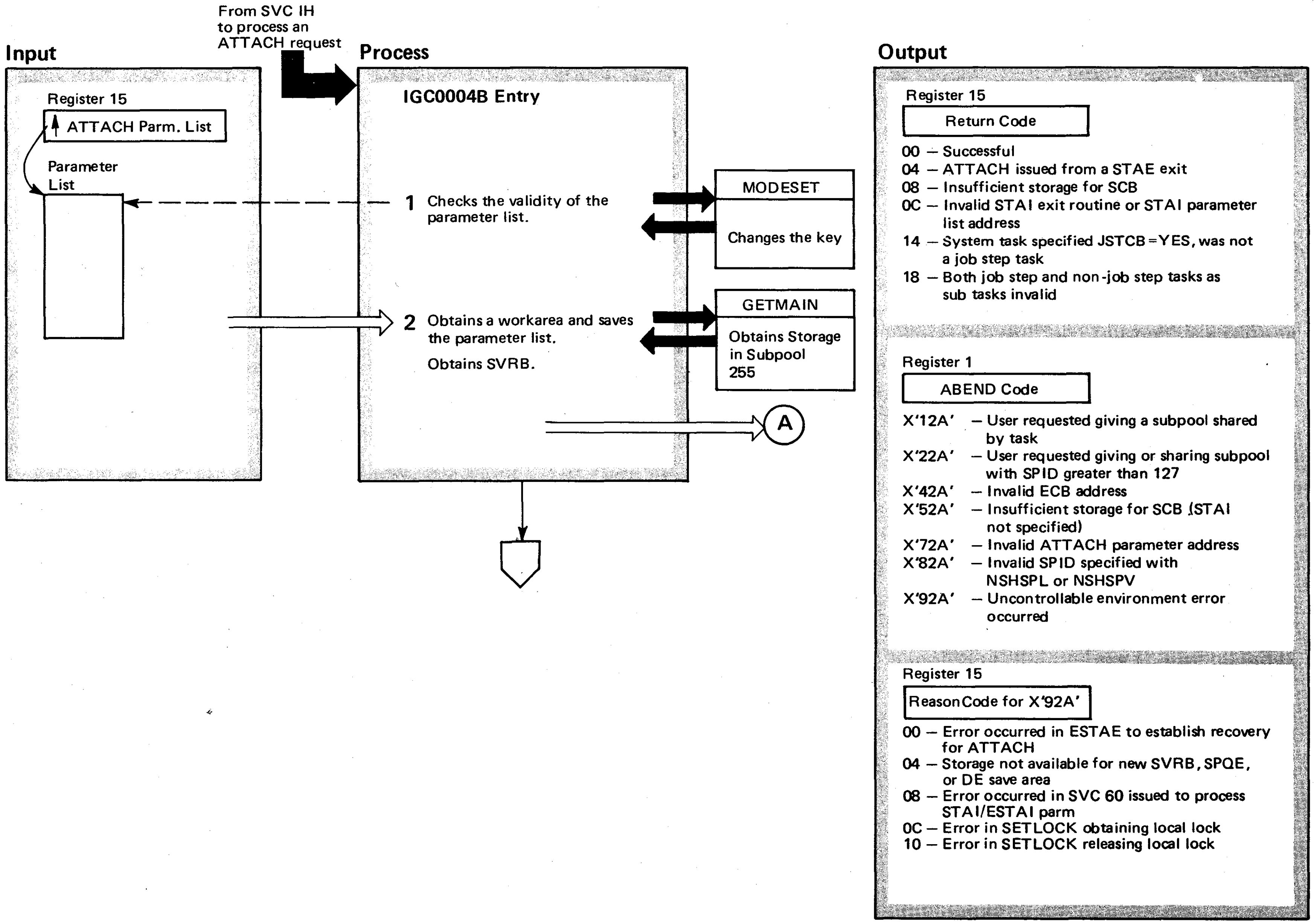

Task Management Overview (no diagram)
20-1 ATTACH Processing (IEAVEATO)
20-2 DETACH Processing (IEAVEEDO)
20-3 CHAP Processing (IEAVECHO)
20-4 WAIT Processing (IEAVSY50)
20-5 POST Processing (IEAVSY50)
20-6 EVENTS Processing (IEAVEVTO)
20-7 ENQ/ RESERVE Processing (IEAVENQ1)
20-8 DEQ Processing (IEAVENQ1)
20-9 ENQ/DEQ/ RESERVE Recovery (IEAVENQ1)
20-10 SPIE Processing (IEAVTB00)
20-11 EXTRACT Processing (IEAVTB00)
20-12 EXIT Processing (IEAVEOR)
20-13 EXIT Prolog Processing (IEAVEEXP)
20-14 STATUS Processing (IEAVSETS)
20-15 MODSET Processing (IEAVMODE)
20-16 TESTAUTH Processing (IEAVTEST)

Figure 2-39. Task Management Visual Contents

Diagram 20-1. ATTACH Processing (IEAVEAT0) (Part 1 of 8)


From SVC IH to process an ATTACH request
Input
Register 15
ATTACH Parm. List
Parameter List
Process
IGC0004B Entry
1 Checks the validity of the parameter list.
2 Obtains a workarea and saves the parameter list.
Obtains SVRB.
MODESET
Changes the key
GETMAIN
Obtains Storage in Subpool 255
A
Output

Register 15

Return Code

00	Successful
04	ATTACH issued from a STAE exit
08	Insufficient storage for SCB
0C	Invalid STAI exit routine or STAI parameter list address
14	System task specified JSTCB=YES, was not a job step task
18	Both job step and non-job step tasks as sub tasks invalid

Register 1

ABEND Code

X'12A'	User requested giving a subpool shared by task
X'22A'	User requested giving or sharing subpool with SPID greater than 127
X'42A'	Invalid ECB address
X'52A'	Insufficient storage for SCB (STAI not specified)
X'72A'	Invalid ATTACH parameter address
X'82A'	Invalid SPID specified with NSHSPL or NSHSPV
X'92A'	Uncontrollable environment error occurred

Register 15

Reason Code for X'92A'

00	Error occurred in ESTAE to establish recovery for ATTACH
04	Storage not available for new SVRB, SPQE, or DE save area
08	Error occurred in SVC 60 issued to process STAI/ESTAI parm
0C	Error in SETLOCK obtaining local lock
10	Error in SETLOCK releasing local lock

Diagram 20-1. ATTACH Processing (IEAVEAT0) (Part 2 of 8)

Extended Description	Module	Label
ATTACH processing allows a problem program or a system program to attach a subtask. The ATTACH routine creates a new TCB that represents a subtask of the original task, fills in control information in the new TCB, places the new TCB on the TCB queue and branches to the LINK routine to provide the linkage to the first program to be executed under the new subtask.		
1 ATTACH checks the validity of the input, and passes control to ABEND to terminate the caller if any invalid input. (Refer to the *OS/VS2 Debugging Handbook, OS/VS2 SPL: Supervisor,* or *OS/VS2 Supervisor Services and Macro Instructions* for a more detailed description of the ATTACH input parameter list.) ATTACH uses MODESET to change the key to that of the caller. Then ATTACH refers to the input data while in the caller's key. ATTACH uses MODESET to change back into key 0. If invalid input is found, a program check error occurs, and the FRR gets control, and diagnoses the error.	IEAVEAT0	CHKPARM
2 ATTACH obtains a workarea and saves the input parameters. The workarea also includes storage for the DE operand, if specified, and provides storage for the SVRB (supervisor request block) constructed for ATTACH re-entry and LINK processing. (See step 6) ATTACH passes the DE parameter area to the LINK routine, if requested. This storage resides in subpool 255.		

Diagram 20-1. ATTACH Processing (IEAVEAT0) (Part 3 of 8)

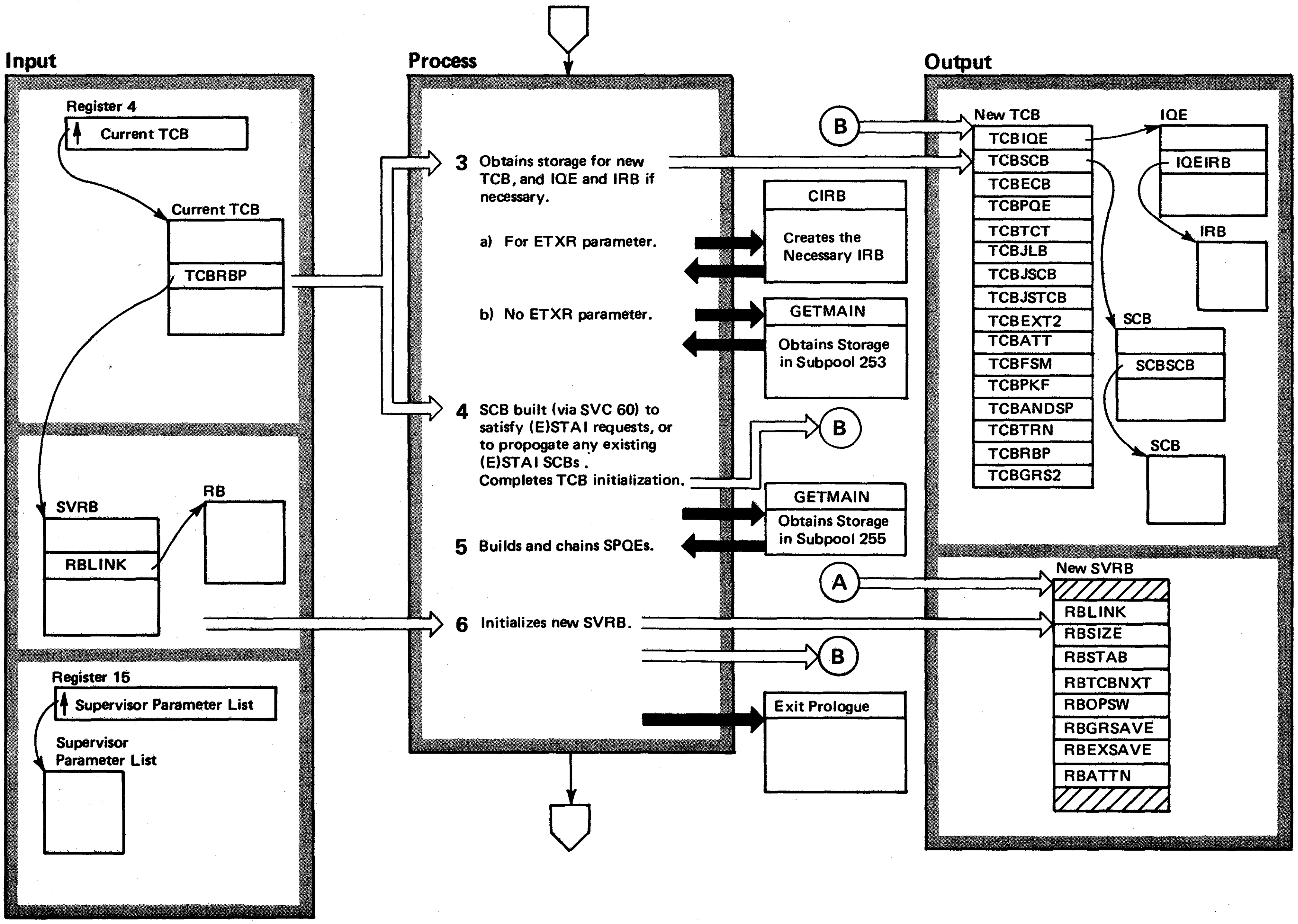
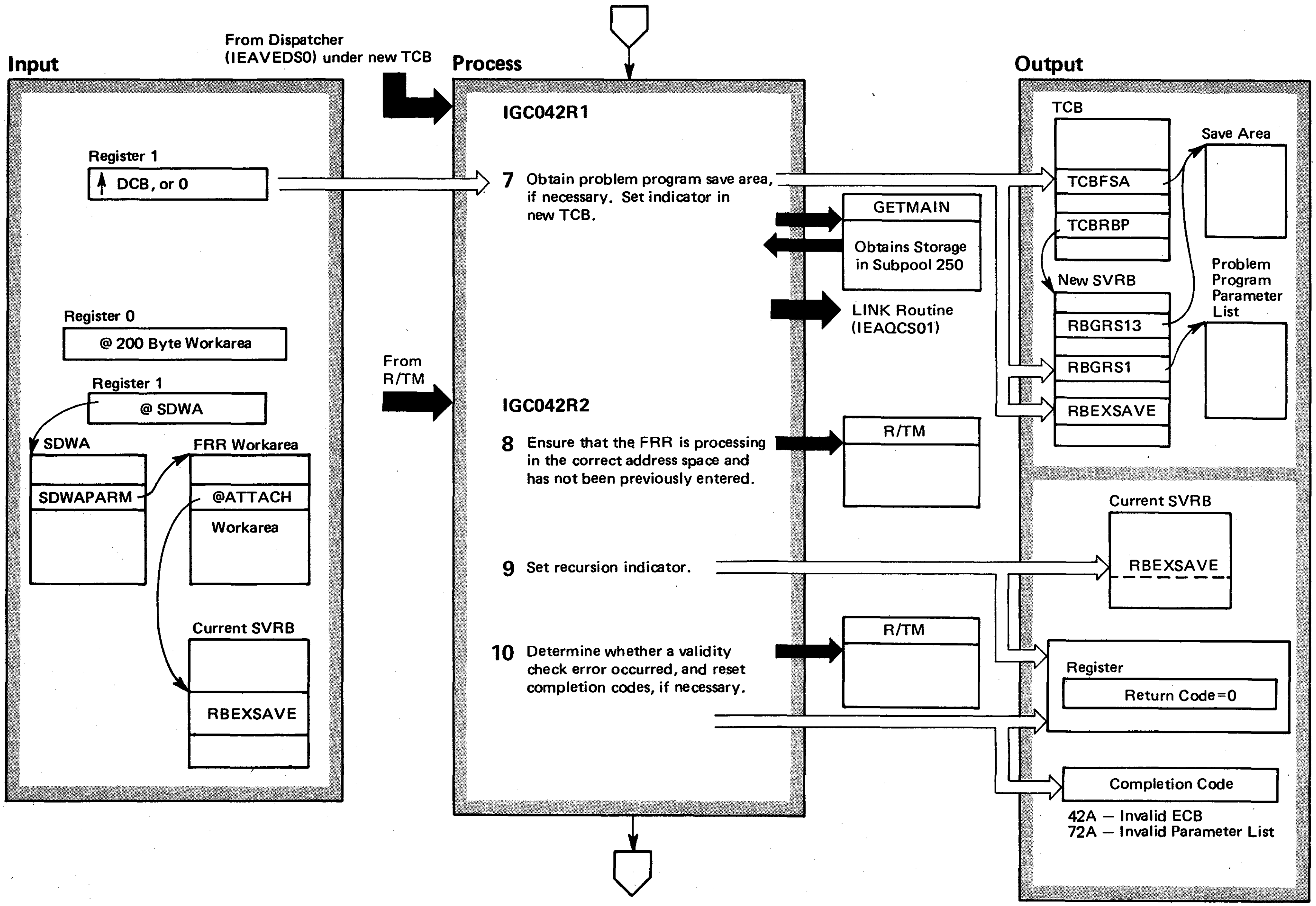


Diagram 20-1. ATTACH Processing (IEAVEAT0) (Part 4 of 8)

Extended Description	Module	Label
3 ATTACH obtains storage for an IQE (interruption queue element) and IRB (interruption request block) when the request contains the ETXR parameter, as well as the storage for the new TCB (task control block). Before ATTACH builds a new IRB, the current TCB's subtask queue is searched for an existing IRB with the same ETXR address using the TCBIQE and IQEIRB fields. ATTACH increases the RBUSE count by one if an IRB exists having the same EXTR address, and then chains the IRB off of the new IQE. ATTACH creates a new IRB, as well as an IQE, and a TCB when one does not exist, using the CIRB (Create IRB) routine..The storage for these control blocks resides in subpool 253.		GETCBS
4 The SVC 60 (STAI/ESTAI) routine builds SCBs (STAE control blocks) to satisfy any requests for STAI or ESTAI on the ATTACH request. SVC 60 also propagates any STAI/ESTAI existing SCBs from the current TCB to the new TCB. The ATTACH routine sets other fields in the new TCB according to the parameters on the ATTACH request, to zero if they are not explicitly set, or propagates the value from the current TCB.		STAIRTN
5 The ATTACH routine builds a queue of SPQEs off of the TCBMSS field according to the values specified in the SHSPL, SHSPV, GSPL, GSPV, and SZERO operands. New SPQEs are built from subpool 255. ATTACH builds shared SPQEs for subpools 236 and 237 if these SPQEs existed for the current TCB, and in accordance with the NSHSPV or NSHSPL parameter. These shared SPQEs are chained off the TCBSWA field.		SHARESP GIVESP SPCONTRL SHARESWA SWARTN
6 The new SVRB contains control information: • RBGRSAVE – contains caller's register 1 through 12. Register 1 contains the address of problem program parameter list. • RBEXSAVE – contains control information previously stored in the current SVRB's RBEXSAVE. • RBOPSW – contains entry point IGC042R1. The TCBGRS field contains the registers used by the IGC042R1 entry point. ATTACH branches to EXIT prolog; control returns to ATTACH at entry point IGC042R1.		

Diagram 20-1. ATTACH Processing (IEAVEAT0) (Part 5 of 8)



7 Obtain problem program save area, if necessary. Set indicator in new TCB.

8 Ensure that the FRR is processing in the correct address space and has not been previously entered.

9 Set recursion indicator.

10 Determine whether a validity check error occurred, and reset completion codes, if necessary.

Diagram 20-1. ATTACH Processing (IEAVEAT0) (Part 6 of 8)

Extended Description	Module	Label
7 Obtains a problem program save area from subpool 250 for SVAREA=YES requests, and places the address of the save area into TCBFSA and RBGRS13. Then the ATTACH re-entry routine initializes the RBEXSAVE field in the SVRB for use by LINK.	IEAVEAT0	IGC042R1
8 After receiving control from R/TM, the ATTACH FRR (functional recovery routine) ensures that it is operating in the address space used by the ATTACH routine. If the FRR is processing in the wrong address space, or if recursion has occurred, control goes to R/TM, with a no-retry indicator.	IEAVEAT0	IGC042R2
9 The ATTACH FRR sets a recursion indicator in the workarea of the current SVRB's RBEXSAVE field.		
10 If a validity check error occurred, the ATTACH FRR changes the completion code to X'42A', for an invalid ECB (event control block) or X'72A', for an invalid parameter list. Control then goes to R/TM, with a return code of 0, indicating that there will be no retry of the failing operation.		

Diagram 20-1. ATTACH Processing (IEAVEAT0) (Part 7 of 8)

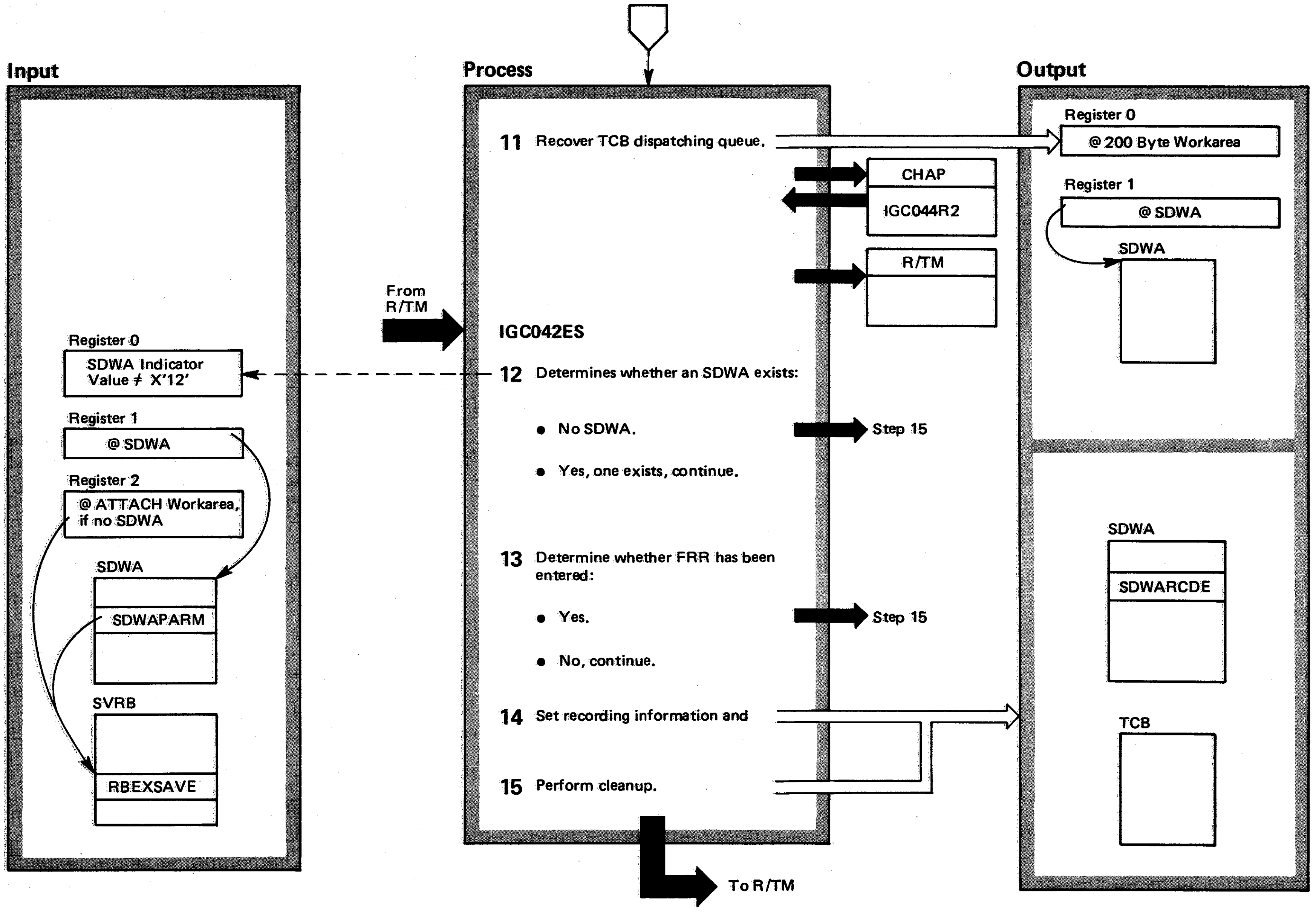


Diagram 20-1. ATTACH Processing (IEAVEAT0) (Part 8 of 8)

Extended Description	Module	Label
11 The ATTACH FRR recovers the TCB dispatching queue by routing control to a CHAP recovery routine (IGC044R2). Control returns, and the ATTACH FRR gives R/TM control with a return code of 0. The SDWA contains recording information. The variable recording area (SDWAVRA) contains the contents of ATTACH's permanent workarea, which contains a code that isolates the portion of ATTACH processing in which the error occurred, the addresses of the new TCB and the current TCB, and other information. This is followed by the recording information supplied by routine IGC044R2 (see the description of CHAP SVC). The recording area, SDWARECP, contains the module name (IEAVEAT0), the CSECT name (IGC0004B), and the FRR name (IGC042R2).		
12 The ATTACH ESTAE (extended STAE) routine receives control from R/TM. First, the routine checks register 0 for a non X'12' value. (A X'12' value indicates that no SDWA exists.) If a SDWA exists, control goes to step 13; otherwise, control goes to step 15.	IGC042E5	IGC042ES
13 The ATTACH ESTAE routine next determines whether the ATTACH FRR routine has already received control by checking the recursion indicator in the RBEXSAVE field of the current SVR3. If it has, no recording is done and control goes to step 15; otherwise, control goes to step 14.		
14 The variable recording area SDWAVRA is set to the contents of the permanent workarea (from the current SVRB's RBEXSAVE field), as described in step 11. The recording area SDWARECP is set to the module name (IEAVEAT0), CSECT name (IGC0004B), and ESTAE routine name (IGC042Es).		
15 The following internal ATTACH subroutines perform clean-up functions: • RTN1 • RTN2 • RTN3 • LOCK • UNLOCK Control goes to R/TM, with a 0 completion code, indicating that no retry of the failing operation will occur.		

Diagram 20-2. DETACH Processing (IEAVEED0) (Part 1 of 8)

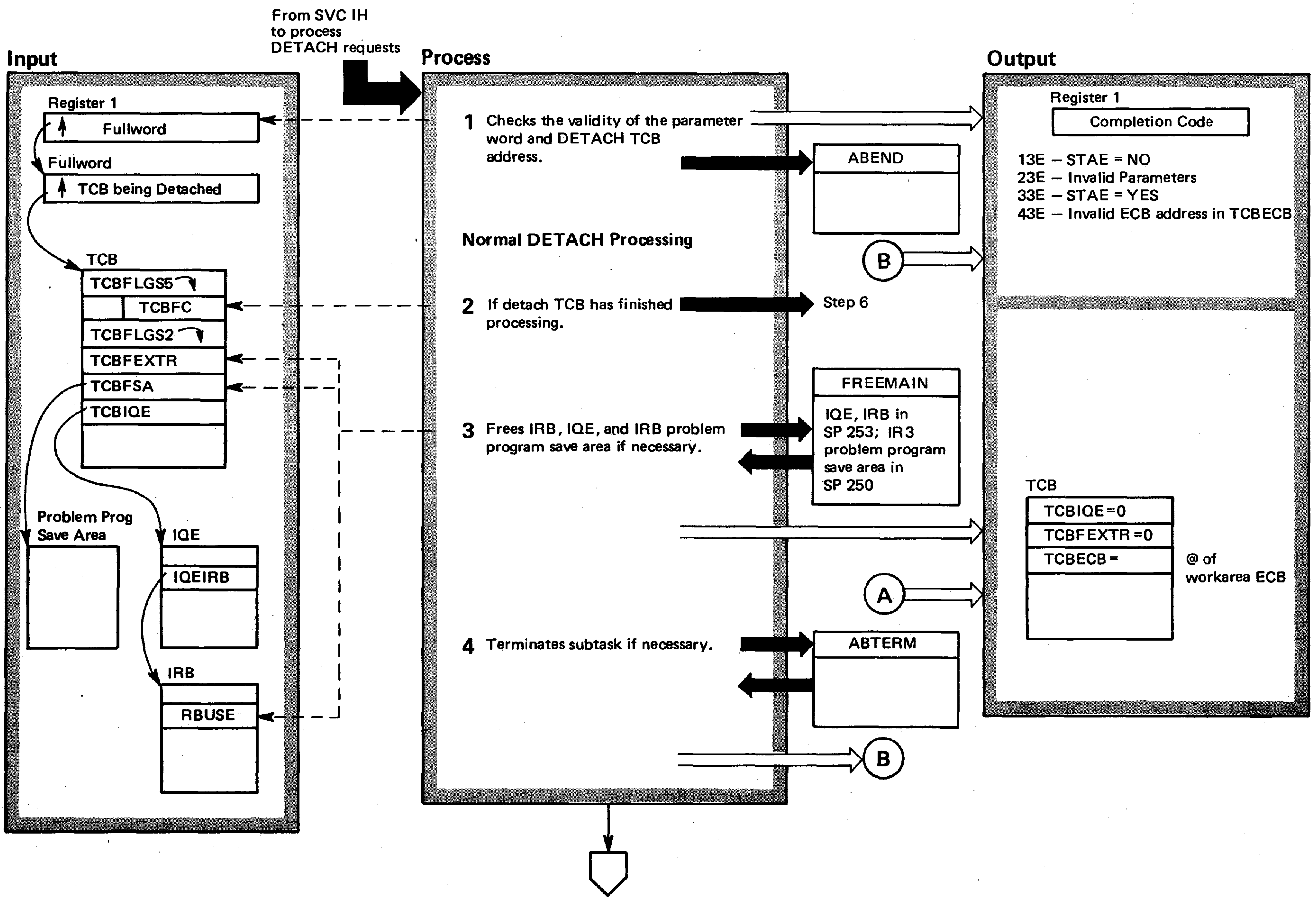
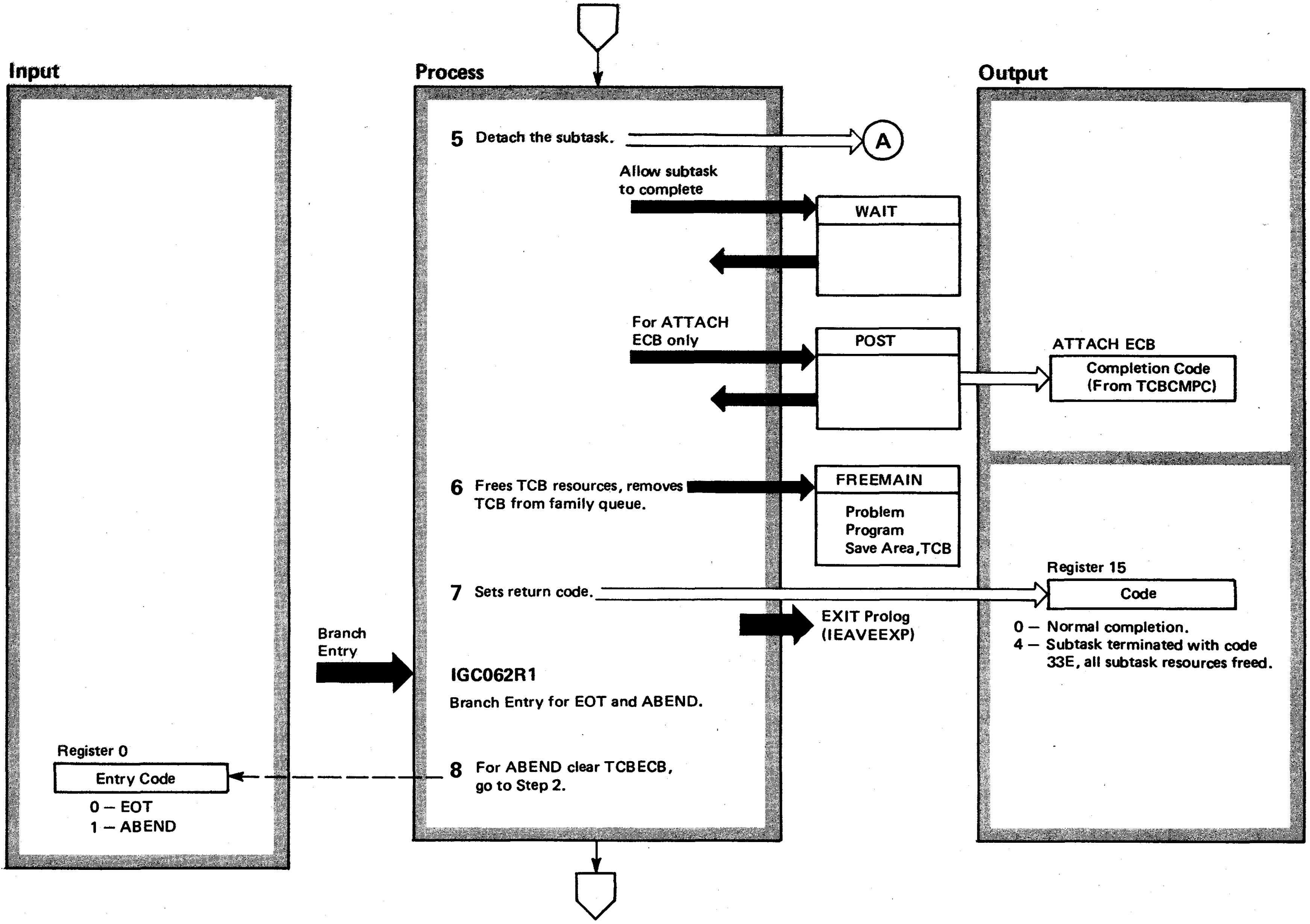

From SVC IH to process DETACH requests
Input
Register 1
Fullword
Fullword
TCB being Detached
TCB
TCBFLGS5
TCBFC
TCBFLGS2
TCBFEXTR
TCBFSA
TCBIQE
Problem Prog Save Area
IQE
IQEIRB
IRB
RBUSE
Process
1 Checks the validity of the parameter word and DETACH TCB address.
ABEND
B
Normal DETACH Processing
2 If detach TCB has finished processing.
Step 6
3 Frees IRB, IQE, and IRB problem program save area if necessary.
FREEMAIN
IQE, IRB in SP 253; IR3 problem program save area in SP 250
A
4 Terminates subtask if necessary.
ABTERM
B
Output
Register 1
Completion Code
13E – STAE = NO
23E – Invalid Parameters
33E – STAE = YES
43E – Invalid ECB address in TCBECB
TCB
TCBIQE=0
TCBFEXTR=0
TCBECB=
@ of workarea ECB

Diagram 20-2. DETACH Processing (IEAVEED0) (Part 2 of 8)

Extended Description	Module	Label
DETACH processing frees subtask resources — the subtask TCB, and possibly a problem program save area — still held after the task has completed. End of task processing frees these resources automatically, except when the creating task had specified the ECB (used to indicate termination of the task) or the ETXR (used to indicate the address of an exit routine) operands on the ATTACH macro. DETACH also provides a means for mother tasks to purge any subtasks not yet terminated.		
The DETACH routine has defined a branch entry available for certain privileged programs. The branch entry provides two functions:		
• Provides a directed detach for use by ABEND to detach subtasks not belonging to the current TCB.		
• Provides a clean-up routine for end-of-task (End-of-task Resource Manager).		

Extended Description	Module	Label
1 Register 1 supplies the address of a fullword containing the address of the subtask TCB to be detached. If an abend code is necessary and bit 0 of Register 1 is 1 (that is, STAE=YES was specified), the abend code is 33E; if bit 0 of Register 1 is 0 (that is, STAE=NO was specified), the abend code is 13E. DETACH checks the input, and passes control to ABEND to terminate callers with invalid input.	IEAVEED0	
2 DETACH frees TCB resources if the TCBFC bit of the TCBFLGS5 field is set to one. DETACH sets a return code of 0 and returns to the caller (see Step 6). DETACH will stop an active TCB by using the STATUS routine (see STATUS Processing diagram).		
3 DETACH checks for an ETXR (end-of-task exit routine) for the TCB being detached. An ETXR exists if the TCBIQE field does not equal zero and bit TCBFETXR equals 1. DETACH uses FREEMAIN to free the IQE and DETACH sets TCBIQE and bit TCBFETXR to zero if an ETXR exists. If the IRB use count (RBUSE) equals 1, DETACH uses FREEMAIN to free the IRB and its associated problem program save area. If the IRB use count exceeds 1, DETACH decreases the IRB use count by one.		
4 DETACH passes control to terminate the subtask.		

Diagram 20-2. DETACH Processing (IEAVEED0) (Part 3 of 8)

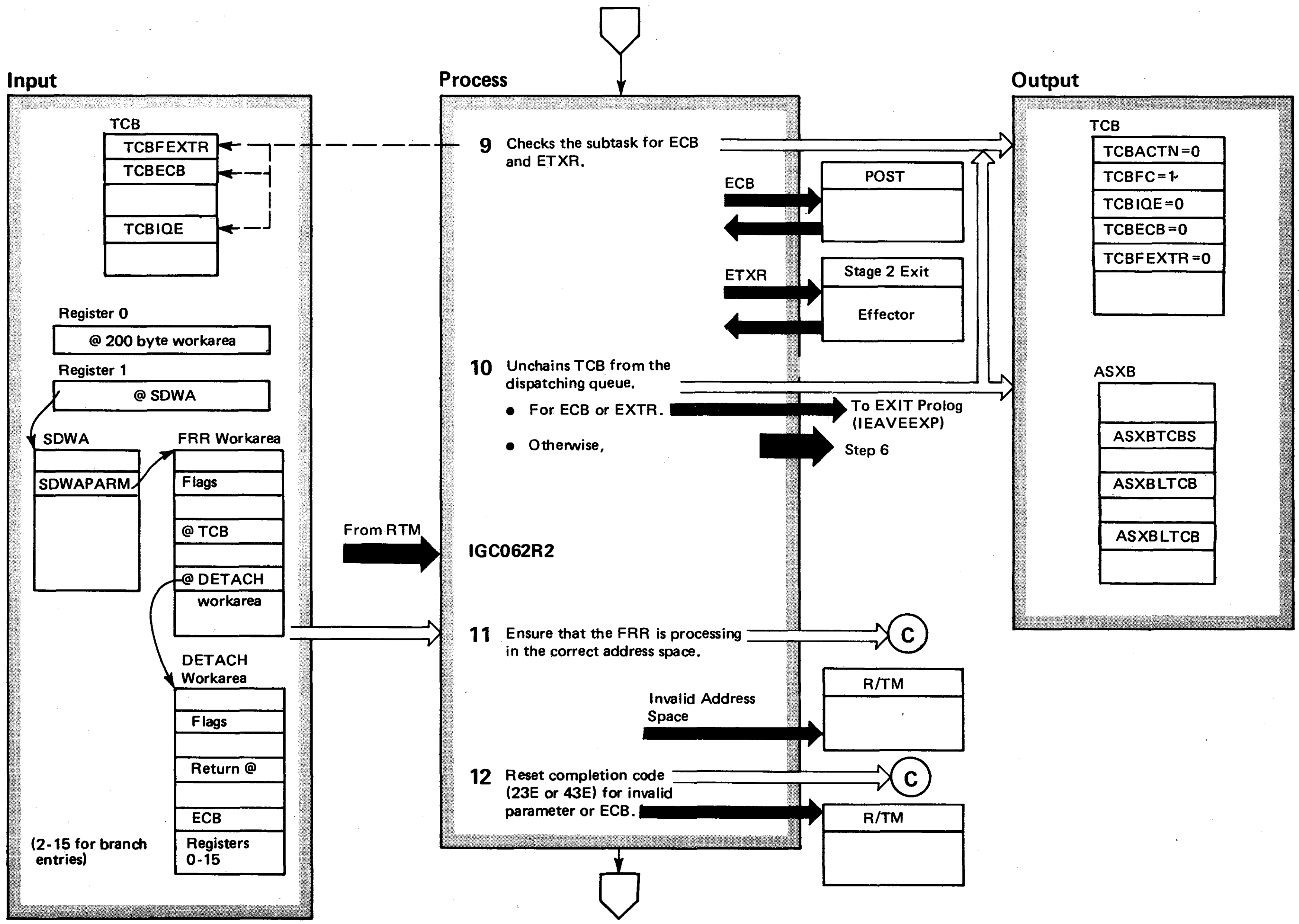


Input
Process
Output
5 Detach the subtask.
A
Allow subtask to complete
WAIT
For ATTACH ECB only
POST
ATTACH ECB
Completion Code (From TCBCMPC)
6 Frees TCB resources, removes TCB from family queue.
FREEMAIN
Problem Program Save Area,TCB
7 Sets return code.
Register 15
Code
0 — Normal completion.
4 — Subtask terminated with code 33E, all subtask resources freed.
EXIT Prolog (IEAVEEXP)
Branch Entry
IGC062R1
Branch Entry for EOT and ABEND.
Register 0
Entry Code
0 — EOT
1 — ABEND
8 For ABEND clear TCBECB, go to Step 2.

Diagram 20-2. DETACH Processing (IEAVEED0) (Part 4 of 8)

Extended Description	Module	Label
5 DETACH saves the TCBECB, resets with the address of the workarea ECB, allows the subtask to complete processing, and posts the TCB completion code if an ATTACH ECB exists.		
6 DETACH frees the TCB problem program save area, located in subpool 250, if one exists, unchains the TCB from the family queue, and frees the detach TCB.		
7 DETACH sets the return code for SVC entry according to completion conditions.		
8 ABEND processing is the same as normal processing, except the TCBECB field is cleared to zeros.		

Diagram 20-2. DETACH Processing (IEAVEED0) (Part 5 of 8)



9 Checks the subtask for ECB and ETXR.

10 Unchains TCB from the dispatching queue.

- For ECB or EXTR.
- Otherwise,

11 Ensure that the FRR is processing in the correct address space.

12 Reset completion code (23E or 43E) for invalid parameter or ECB.

Diagram 20-2. DETACH Processing (IEAVEED0) (Part 6 of 8)

Extended Description	Module	Label

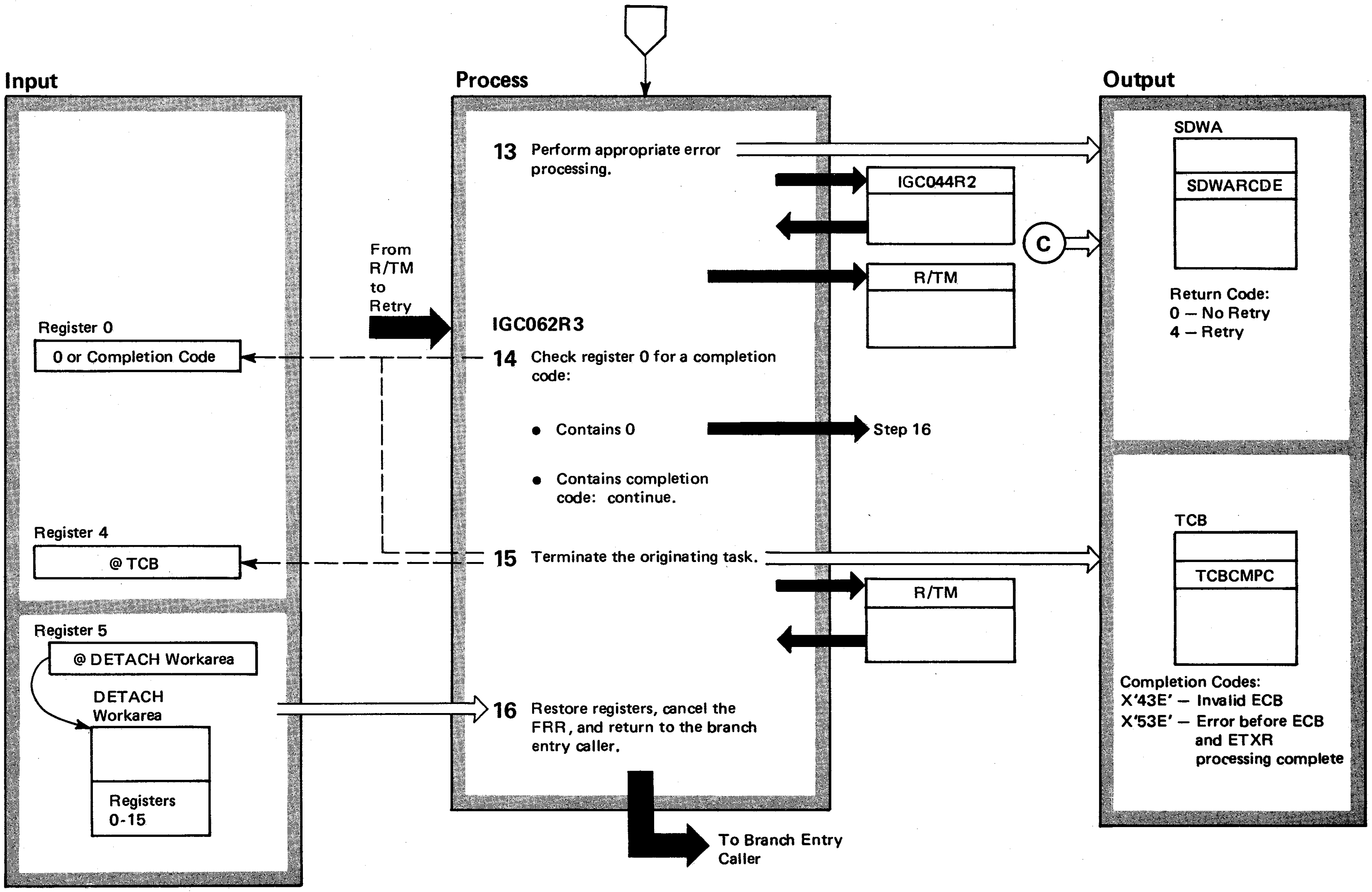
9 DETACH gives control to the POST routine (see the POST Processing (IEAVSY50) diagram) for an ATTACH ECB, and gives control to the Stage 2 Exit Effector (see the Stage 2 Exit Effector (IEAVEEE2) diagram) for end-of-task exit routine processing.

10 DETACH unchains the TCB from the dispatching queue, and decreases the count of TCBs on the dispatching queue in field ASXBTCBS. DETACH clears TCB fields TCBECB and TCBIQE, and TCBFEXTR when either ECB or ETXR conditions exist for the detach TCB, sets TCBFC to equal 1, but does not free the TCB itself. If neither ECB or ETXR conditions exist, FREEMAIN frees the TCB and its problem program save area (if one exists).

11 After receiving control from R/TM, the DETACH FRR (functional recovery routine) ensures that it is operating in the address space used by the DETACH routine. If the FRR is processing in the wrong address space, control goes to R/TM.

12 If DETACH was entered via SVC, field SDWACMPC is set to X'23E' for an invalid parameter or to X'43E' for an invalid ECB address. Control then goes to R/TM with a return code of 0 in field SDWARCDE. If DETACH was branch-entered, an indicator is set and step 13 is done.

Diagram 20-2. DETACH Processing (IEAVEED0) (Part 7 of 8)



13 Perform appropriate error processing.

IGC062R3

14 Check register 0 for a completion code:

- Contains 0 → Step 16
- Contains completion code: continue.

15 Terminate the originating task.

16 Restore registers, cancel the FRR, and return to the branch entry caller.

Diagram 20-2. DETACH Processing (IEAVEED0) (Part 8 of 8)

Extended Description	Module	Label
13 The DETACH FRR recovers the TCB dispatching queue by routing control to a CHAP recovery routine (IGC044R2). Control returns. If DETACH was branch entered for end-of-task resource manager processing, the terminating TCB is removed from the family queue; if an invalid ECB was detected, ABTERM code X'43E' is passed to retry routine IGC062R3. If end-of-task resource manager processing had not yet completed processing the end-of-task ECB or EXTR, ABTERM code X'53E' is passed to the retry routine. Otherwise, no ABTERM is indicated for this routine. The DETACH FRR gives R/TM control with a return code of 0 for SVC entries or 4 for branch entries. The SDWA contains recording information. Field SDWAVRA contains recording information as set by routine IGC044R2. (See extended description of CHAP SVC for description of this information.) Also, field SDWARECP is set to module name (IEAVEED0), CSECT name (IGC062), and FRR name (IGC062R2). For branch entries, general register 0 contains a completion code of X'43E' to indicate a validity check error, a X'53E' to indicate an error in end-of-task processing, or 0 for no error. Control goes to R/TM.		
14 The DETACH recovery retry routine checks register 0 for a completion code. If register 0 does not contain a completion code, control goes to step 16. Otherwise, processing continues.		IGC062R3
15 If a completion code exists, the DETACH recovery retry routine terminates the originating task by giving control to R/TM.		
16 Control returns to the caller that entered DETACH via a branch.		

Diagram 20-3. CHAP Processing (IEAVECH0) (Part 1 of 6)

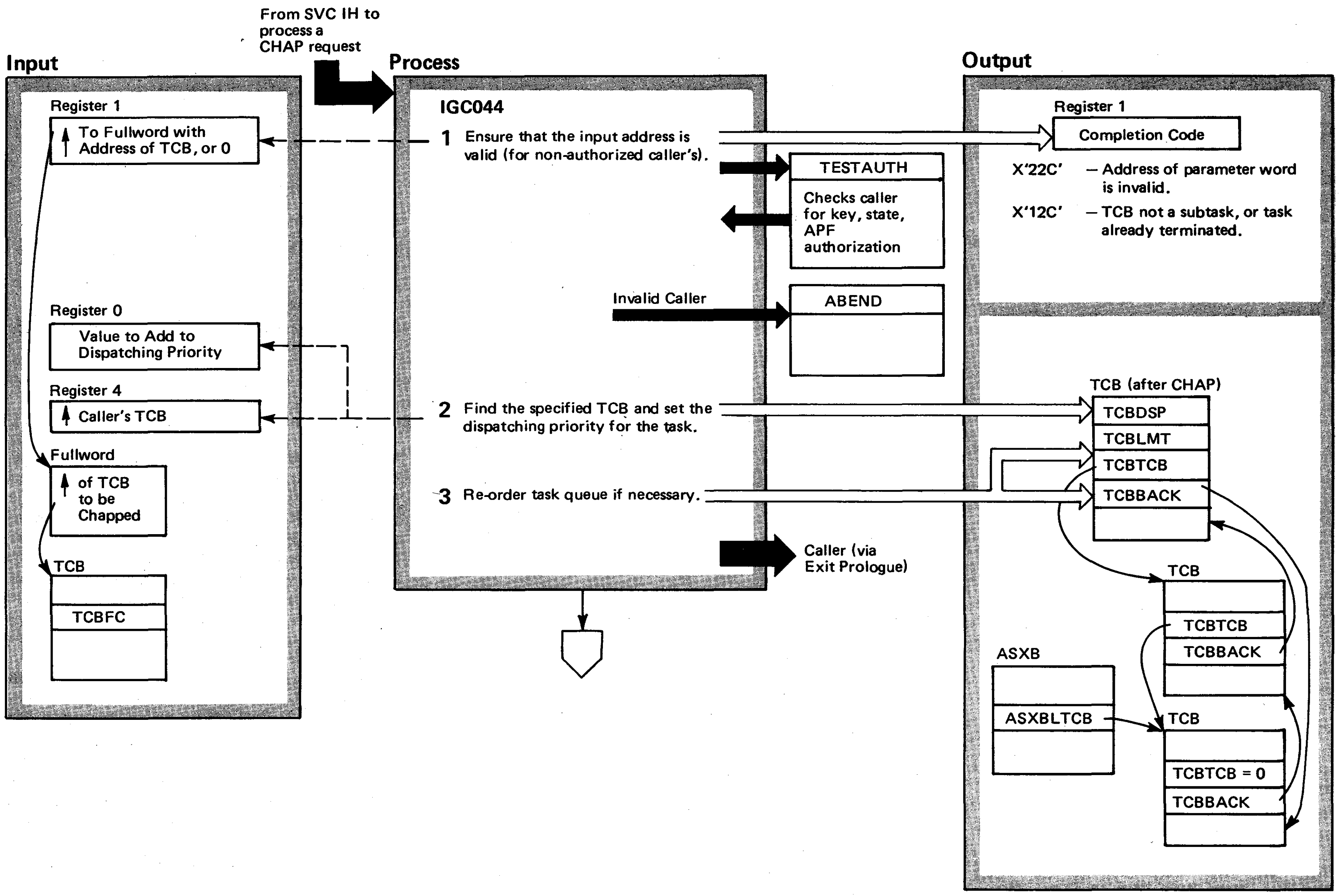

From SVC IH to process a CHAP request
Input
Register 1
To Fullword with Address of TCB, or 0
Register 0
Value to Add to Dispatching Priority
Register 4
Caller's TCB
Fullword
of TCB to be Chapped
TCB
TCBFC
Process
IGC044
1 Ensure that the input address is valid (for non-authorized caller's).
TESTAUTH
Checks caller for key, state, APF authorization
Invalid Caller
ABEND
2 Find the specified TCB and set the dispatching priority for the task.
3 Re-order task queue if necessary.
Caller (via Exit Prologue)
Output
Register 1
Completion Code
X'22C' — Address of parameter word is invalid.
X'12C' — TCB not a subtask, or task already terminated.
TCB (after CHAP)
TCBDSP
TCBLMT
TCBTCB
TCBBACK
TCB
TCBTCB
TCBBACK
ASXB
ASXBLTCB
TCB
TCBTCB = 0
TCBBACK

Diagram 20-3. CHAP Processing (IEAVECH0) (Part 2 of 6)

Extended Description	Module	Label
The CHAP routine permits a problem program or system program to alter its dispatching priority or the dispatching priority of one of its subtasks. The subtask must belong to the issuer; that is, the subtask must have been attached by a routine belonging to the caller's task, and its TCB must therefore be on the caller's subtask queue. In addition, an authorized caller can change the dispatching priority of any task in the address space.		IGC044
A program issuing the CHAP macro instruction may change the dispatching priority of a specified task to any value between 0 and the issuer's limit priority. The distinction between dispatching and limit priorities follows in the next paragraphs.		
Although both priorities are specified as parameters of the ATTACH macro instruction, they serve different functions. The dispatching priority determines the appropriate position of a TCB in the task queue, and also the next routine to be placed in execution by the dispatcher. The dispatcher gives control to the ready TCB with the highest dispatching priority.		
In contrast, the limit priority is used by the CHAP routine to determine the maximum value to which it may increase the dispatching priority of the task.		

Extended Description	Module	Label
1 If 0 is supplied in register 1, the dispatching priority of the caller is to be changed. The address of the caller's TCB was placed in register 4 by the SVC Interruption Handler (IH), and no validity check of the address is required. The CHAP routine holds the local lock.	IEAVECH0	
2 If a valid address is supplied in register 1 and if the caller is not authorized, CHAP compares the specified TCB address with the addresses of the TCBs that represent the caller's subtasks. If the subtask is not found, CHAP abnormally terminates the caller.		
The CHAP routine does not make this test if the caller's TCB (address in register 4) is the subject.		
The dispatching priority is in field TCBDSP, and the limit priority is in field TCBLMP.		
3 CHAP queues the TCB according to its dispatching priority, but at the end of the group with the same priority level.		

Diagram 20-3. CHAP Processing (IEAVECH0) (Part 3 of 6)

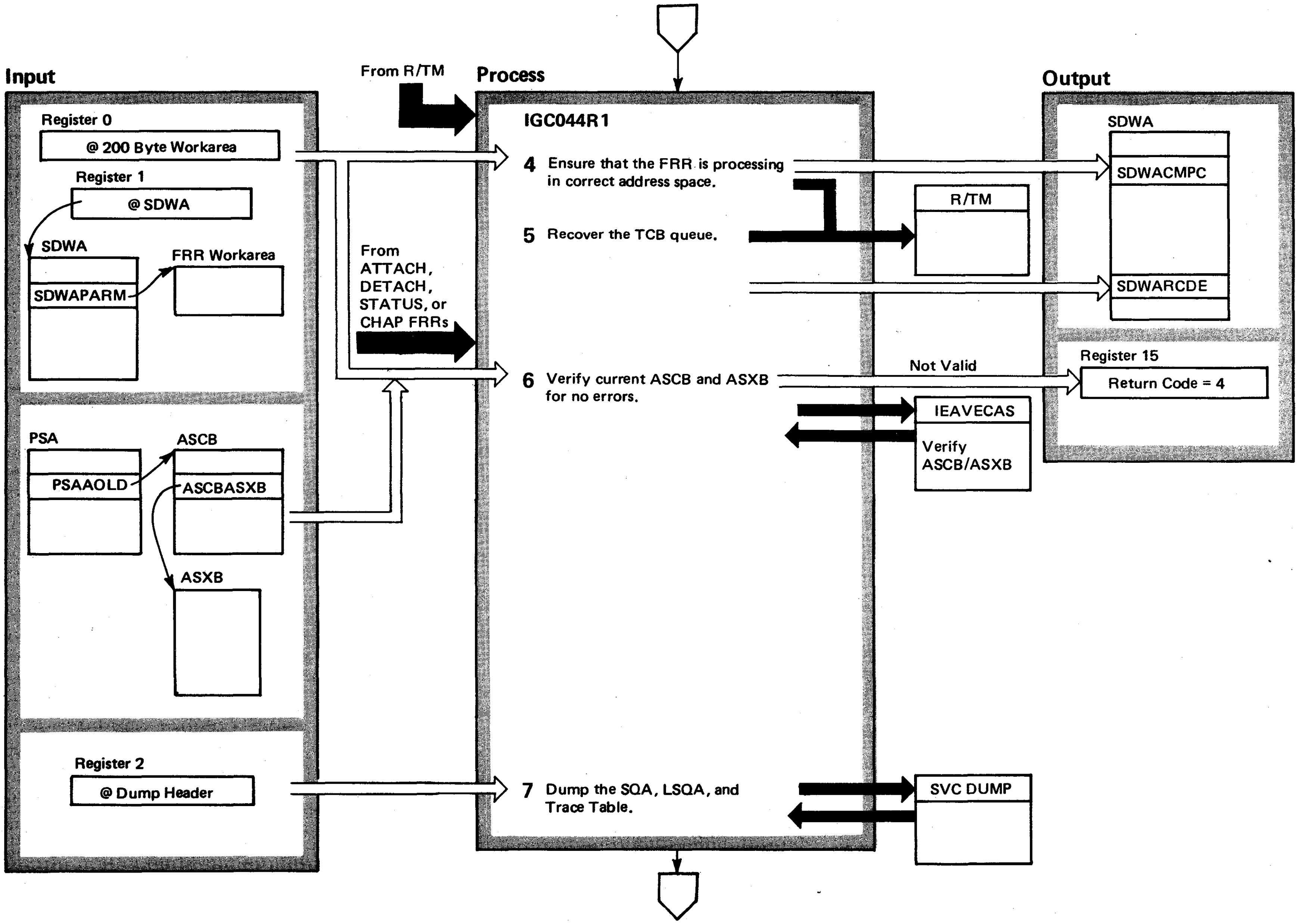
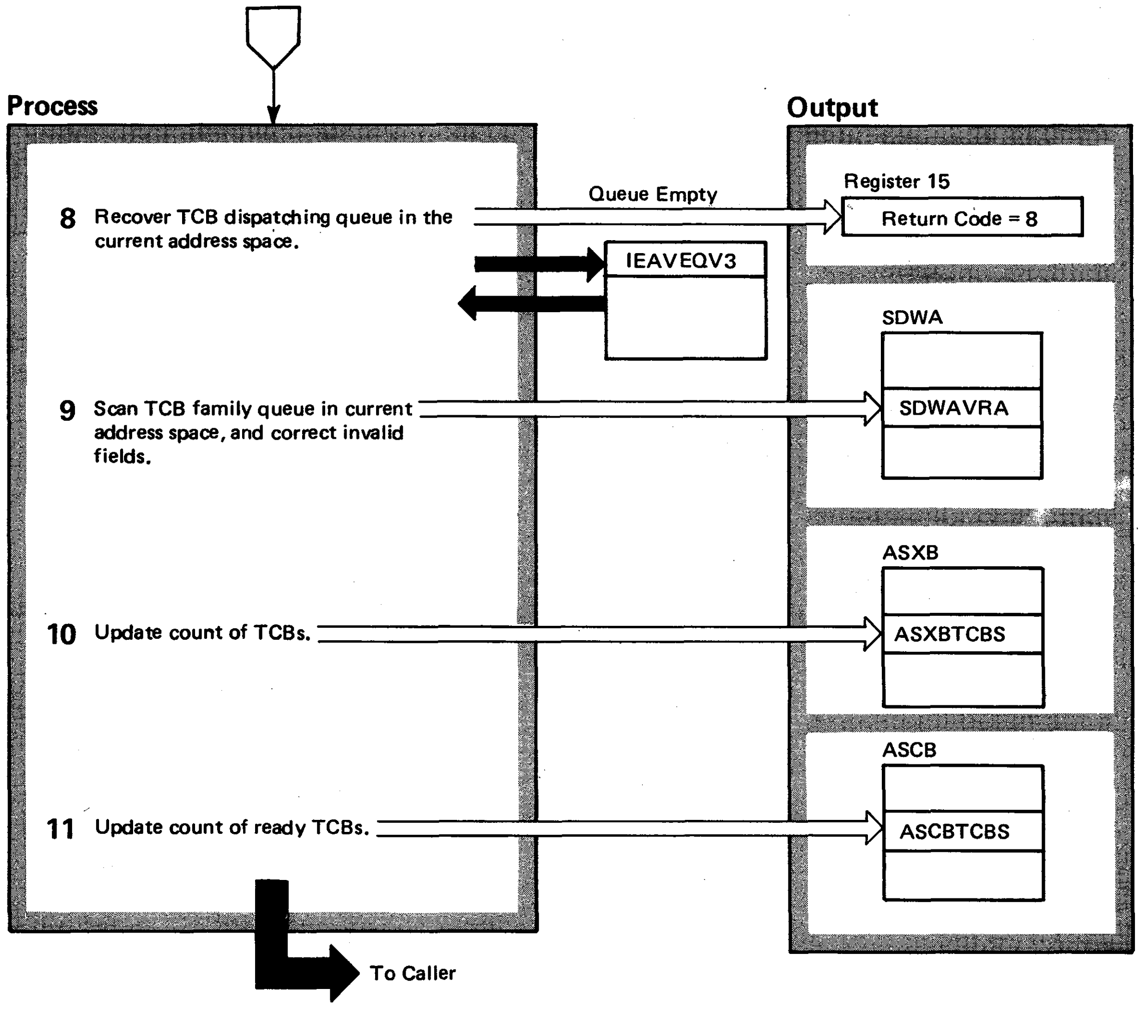


Diagram 20-3. CHAP Processing (IEAVECH0) (Part 4 of 6)

Extended Description	Module	Label
4 After receiving control from R/TM, the CHAP FRR ensures that it is operating in the address space used by the CHAP routine. If the FRR is processing in the wrong address space, control goes to R/TM. If an invalid parameter is detected, the CHAP FRR sets the SDWACMPC field of the SDWA to a X'22C' completion code and control goes to R/TM. If "percolation" has occurred, the CHAP FRR specifies 'no recording' and control goes to R/TM. The return code is zero for all three cases.		
5 Then, the CHAP FRR calls routine IGC044R2 (an external entry) to recover the TCB dispatching queue, sets a 0 return code, and gives control to R/TM. Recording information has been set in field SDWAVRA by routine IGC044R2. In addition, field SDWARECP contains the module name (IEAVECH0), the CSECT name (IGC044), and the FRR name (IGC044R1).		
6 The CHAP TCB queues recovery routine verifies the accuracy of the current ASCB and ASXB by going to IEAVECAS. If the ASCB and ASXB are not valid, register 15 contains a return code of 4.		
7 SVC DUMP dumps the LSQA, SQA, and trace table, via the SDUMP macro instruction.		

Diagram 20-3. CHAP Processing (IEAVECH0) (Part 5 of 6)



Process
Output
8 Recover TCB dispatching queue in the current address space.
Queue Empty
Register 15
Return Code = 8
IEAVEQV3
SDWA
9 Scan TCB family queue in current address space, and correct invalid fields.
SDWAVRA
ASXB
10 Update count of TCBs.
ASXBTCBS
ASCB
11 Update count of ready TCBs.
ASCBTCBS
To Caller

Diagram 20-3. CHAP Processing (IEAVECH0) (Part 6 of 6)

Extended Description	Module	Label

8 The IEAVEQV0 routine (entry point IEAVEQV3) recovers the TCB dispatching queue. If the queue is empty, register 15 contains a return code of 8.

9 The CHAP queues recovery routine corrects any invalid fields in the TCB family queue. The SDWA contains descriptive information about the errors found and corrected. Field SDWAVRA contains the queue verify routine name, IEAVEQV3, followed by recording information supplied by that routine. Following this recording information is the name IGC044R2 and a four-byte descriptor field. The format of the descriptor follows:

Byte	Bit	Description
1	0	Set to 1: errors were detected but not recorded. Set to 0: all errors detected were recorded.
	1-7	Reserved
2	0-7	Number of errors recorded
3	0-7	Number of errors detected.
4	0-7	Return code from IGC044R2.

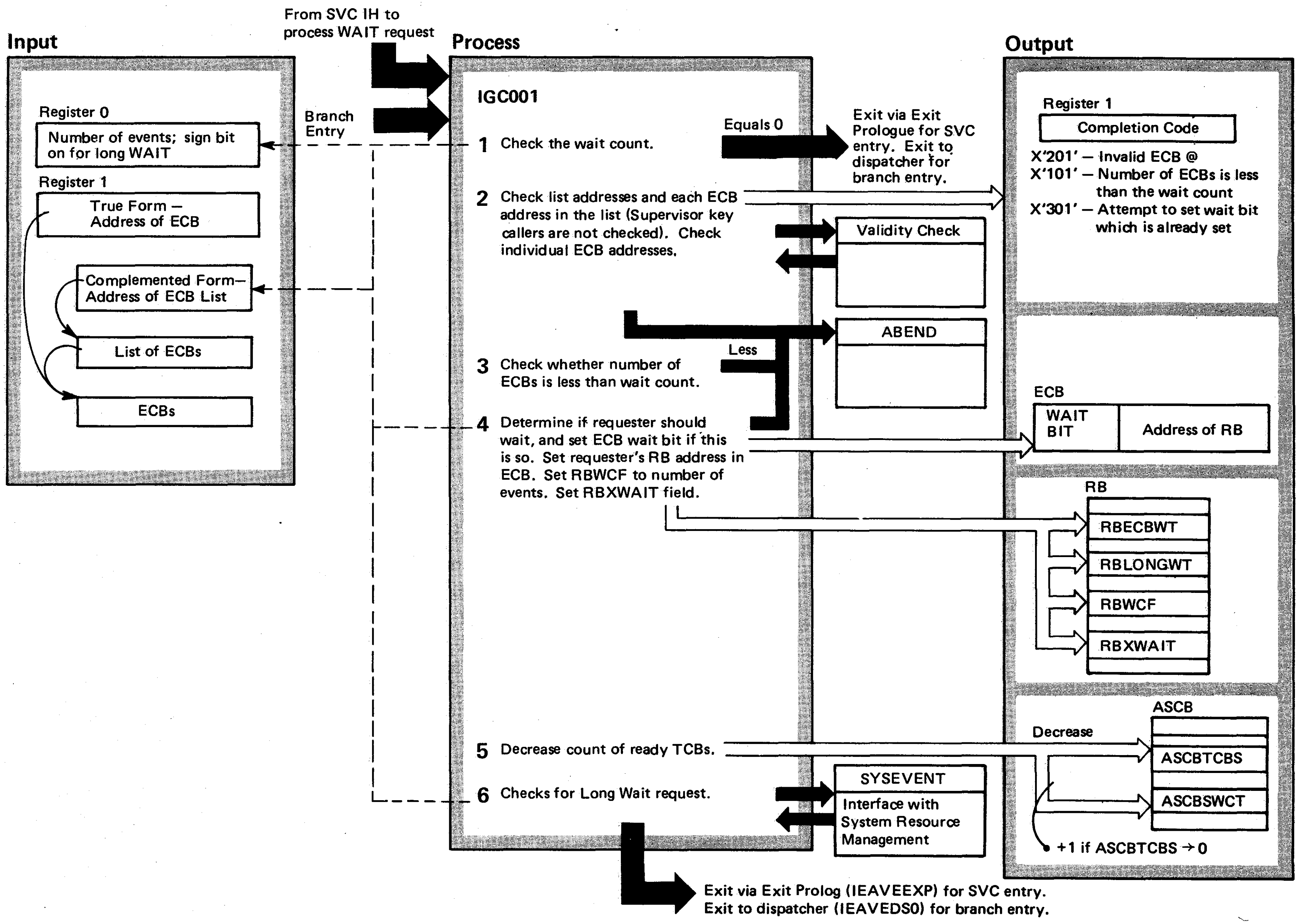
Following the descriptor is a 16-byte entry for each error detected. The entry format follows:

Bytes	Description
1-4	NTCB̸ or LTCB̸ to indicate whether TCBNTC or TCBLTC was updated.
5-8	Address of TCB with invalid field.
9-12	Contents of the invalid field.
13-16	The replacement address (new contents for that field).

Recording terminates whenever SDWAVRA becomes filled (indicated bv fields SDWAVRAL and SDWAURAL).

10 The count of ready TCBs in the ASCB (ASCBTCBS) is updated to reflect the TCBs on the dispatching queue. The total number of TCBs on the dispatching queue is updated in the ASXB (ASXBTCBS). Control then returns to the caller. CHAP itself, ATTACH, DETACH, and STATUS all call routine IGC044R2 to recover the TCB queue.

Diagram 20-4. WAIT Processing (IEAVSY50) (Part 1 of 2)


Input
Register 0
Number of events; sign bit on for long WAIT
Register 1
True Form – Address of ECB
Complemented Form– Address of ECB List
List of ECBs
ECBs
From SVC IH to process WAIT request
Branch Entry
Process
IGC001
Equals 0
Exit via Exit Prologue for SVC entry. Exit to dispatcher for branch entry.
Validity Check
ABEND
Less
SYSEVENT
Interface with System Resource Management
Output
Register 1
Completion Code
X'201' – Invalid ECB @
X'101' – Number of ECBs is less than the wait count
X'301' – Attempt to set wait bit which is already set
ECB
WAIT BIT
Address of RB
RB
RBECBWT
RBLONGWT
RBWCF
RBXWAIT
ASCB
Decrease
ASCBTCBS
ASCBSWCT
+1 if ASCBTCBS → 0
Exit via Exit Prolog (IEAVEEXP) for SVC entry.
Exit to dispatcher (IEAVEDS0) for branch entry.

1 Check the wait count.

2 Check list addresses and each ECB address in the list (Supervisor key callers are not checked). Check individual ECB addresses.

3 Check whether number of ECBs is less than wait count.

4 Determine if requester should wait, and set ECB wait bit if this is so. Set requester's RB address in ECB. Set RBWCF to number of events. Set RBXWAIT field.

5 Decrease count of ready TCBs.

6 Checks for Long Wait request.

Diagram 20-4. WAIT Processing (IEAVSY50) (Part 2 of 2)

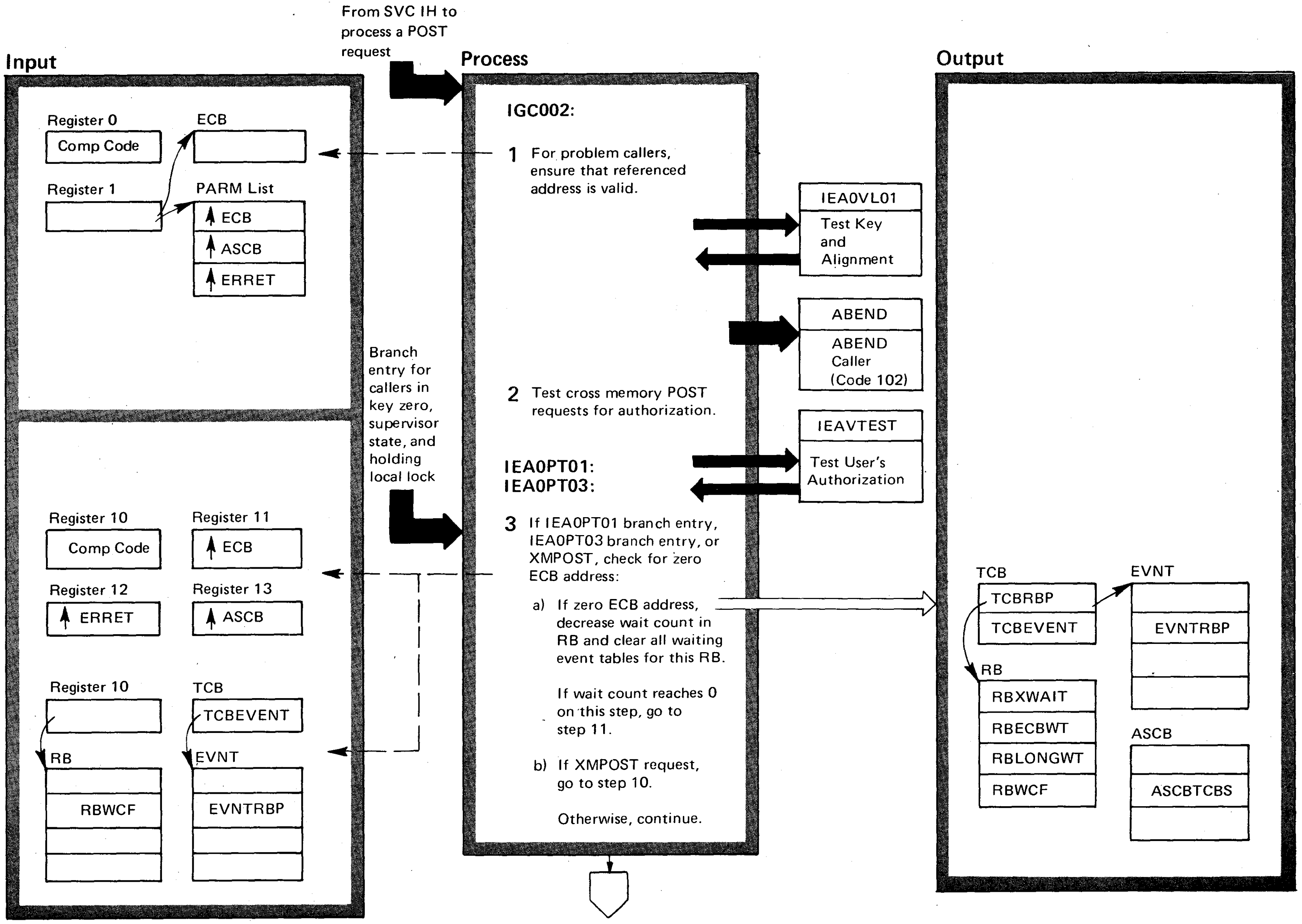
Extended Description	Module	Label
WAIT processing permits a problem program or system program to stop its execution until a specified number of events have occurred, such as the completion of one or more I/O operations. When the specified events have occurred, the POST routine indicates the occurrence of the awaited event or events via the CS (compare and swap) instruction, and makes the program ready (no longer waiting), so that its execution can continue.		
1 Control returns to the caller if the events waited on have already occurred (wait count = 0).	IEAVSY50	ENDWTCT
2 WAIT checks the specified ECB address or the list addresses and ECB addresses only for non supervisor key callers.		ADDROK
3 WAIT sets the RBECBWT bit in the caller's RB when the caller specifies a wait count less than number of ECBs. This means that the caller awaits fewer events than the maximum number that can occur. For example, if a WAIT request is fulfilled by the completion of one of three possible I/O operations, the wait bit set in each of the two ECBs not yet posted is now misleading. If the RBECBWT bit is set, the POST routine clears the wait bit in each of the ECBs not yet posted, and also clears the RBECBWT bit. This removes the misleading indicators.		ECBWT

4

Event Complete?	Wait Flag Set	Decrease Wait Count	Action
Yes	NA	=0	• Reset RBECBWT=0 and clear wait flags in any ECBs in the list.
Yes	NA	≠0	• Continue processing ECBs. • Go to Step 5 if this is the last ECB.
No	Yes	NA	• ABEND – code X'301'.
No	No	NA	• Set ECB wait flag. • Put address of caller's RB in ECB for POST. • Continue processing ECBs. If this is the last ECB, store wait count in RBWCF and go to Step 5.

Extended Description	Module	Label
5 Decrease ASCBTCBS count (ready TCBs); if the ASCBTCBS count reaches 0, WAIT increases the ASCBSWCT by 1.		CSTCBSDN
6 If the user issued a long wait request, the long wait bit is set, and if all the TCBs in the address space are either in a wait condition or nondispatchable, then go to System Resource Manager.		LONGRBCK

Diagram 20-5. POST Processing (IEAVSY50) (Part 1 of 12)



IGC002:

1 For problem callers, ensure that referenced address is valid.

2 Test cross memory POST requests for authorization.

IEA0PT01:
IEA0PT03:

3 If IEA0PT01 branch entry, IEA0PT03 branch entry, or XMPOST, check for zero ECB address:

a) If zero ECB address, decrease wait count in RB and clear all waiting event tables for this RB.

If wait count reaches 0 on this step, go to step 11.

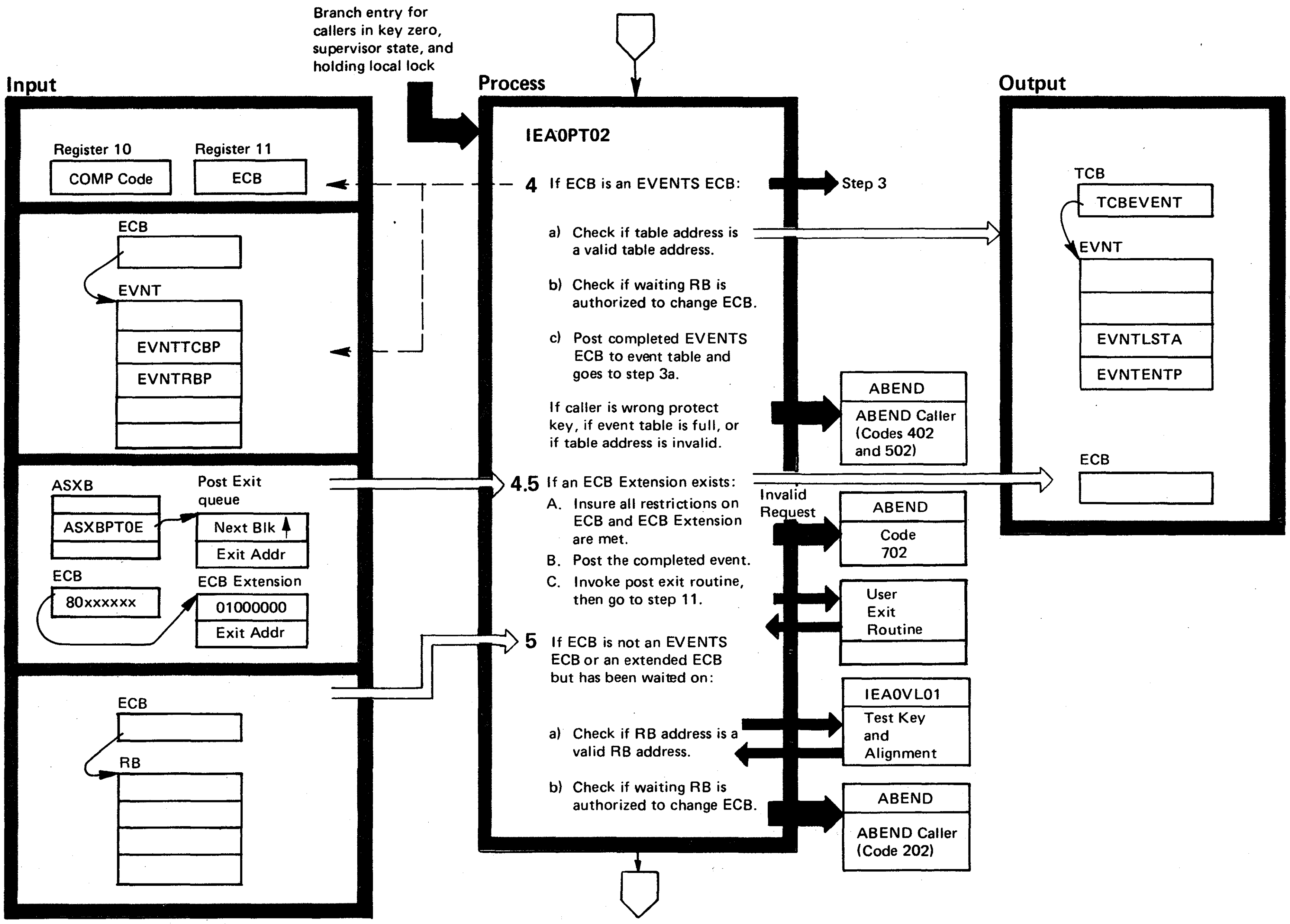
b) If XMPOST request, go to step 10.

Otherwise, continue.

Diagram 20-5. POST Processing (IEAVSY50) (Part 2 of 12)

Extended Description	Module	Label
POST processing signals to a waiting program the occurrence of an expected event (such as the completion of an I/O operation). To signal the event occurring, POST changes an indicator in an ECB (event control block) via the CS (compare and swap) instruction. The program issuing the POST and the waiting program share the same ECB.	IEAVSY50	
POST places a code in the ECB, as specified by the issuer. The waiting program inspects the code to determine the type of event that occurred.		
POST also determines whether the waiting program can be dispatched.		
If the ECB was an event type ECB, POST places the address of the completed ECB at the end of the event table and moves the end-of-list indicator.		
If the ECB was an extended ECB that identifies a valid POST exit routine, POST will route control to the identified exit routine.		
1 POST ensures that the ECB address referenced is valid for problem program callers.		ECBVALID
2 Only authorized users can use the cross-memory post service.		AUTHXMP
3 If the caller specifies an ECB address of 0, register 10 contains the RB address. The RB wait count field (RBWCF) is decreased by 1. Also, if any event tables are waiting on this RB, they are taken out of the wait state (EVNTRBP=0).		
a) If the RB wait count goes to zero, the RBXWAIT, RBECBWT, RBLONGWT wait bits are reset to make the RB ready. If the TCB is made ready, the count of ready TCBs (ASCBTCBS) count is updated for use by the Dispatcher.		
b) A XMPOST request is determined by testing bit 0, register 11. If it is one, or a parm list was specified on the SVC entry, an XMPOST is requested. Go to step 10 to schedule an SRB in the specified address space.		

Diagram 20-5. POST Processing (IEAVSY50) (Part 3 of 12)



IEA0PT02

4 If ECB is an EVENTS ECB:

a) Check if table address is a valid table address.

b) Check if waiting RB is authorized to change ECB.

c) Post completed EVENTS ECB to event table and goes to step 3a.

If caller is wrong protect key, if event table is full, or if table address is invalid.

4.5 If an ECB Extension exists:

A. Insure all restrictions on ECB and ECB Extension are met.

B. Post the completed event.

C. Invoke post exit routine, then go to step 11.

5 If ECB is not an EVENTS ECB or an extended ECB but has been waited on:

a) Check if RB address is a valid RB address.

b) Check if waiting RB is authorized to change ECB.

Extended Description	Module	Label
4 An EVENTS ECB is determined by checking the low order bit of a waited on ECB. If that bit is on, the ECB is assumed to be an EVENTS ECB.	IEAVSY50	RBCHECK
a) The event table address is taken from the ECB, and the TCB address is gotten from the event table. The TCB ready queue is searched for this TCB. When it is found, the event table queue is searched for the event table. If it is found, the table address is valid.		
b) If an RB is waiting on this event table, it is checked for problem key (keys 8-15). If it is problem key, the ECB is referenced in the key of the waiting TCB. If the waiting TCB was not in the proper key, the EVENT FRR will receive control, change the completion code to '402', and percolate.		
c) The completed event is added to the event table, unless the table is full, in which case the user will be abended with '502'. The complete bit and the completion code are stored in the ECB, and control is given to step 3a.		
4.5 It is assumed that an ECB extension exists when the low order two bits of a waited-on ECB are on.	IEAVSY50	EXTECB
a) The ECB and ECB extension are checked to ensure that they pass all post restrictions. Failure to pass a restriction results in a 702 ABEND. The reason code associated with the abend identifies the cause of the abend.		
b) The completion bit and completion code are stored into the ECB.		
c) The exit routine identified in the ECB extension is invoked via a branch. This routine executes as a closed subroutine of post. The interface to the exit routine is described in the *OS/VS2 System Programming Library: Supervisor.* Upon return, control is given to step 11.		

Extended Description	Module	Label
5 If the ECB is waited on and the low order bit is not on, the ECB is a standard ECB.		NOEVENTS
a) The TCB RB queue is searched, comparing the RB address in the ECB to that of the RB in the address space. If an equal compare is made, the ECB is valid.		
b) If the waiting RB is problem key (key 8-15), the ECB address is passed to validity check to verify the waiting TCB's authorization to change that ECB. If validity check fails, the caller will be abended with '202'.		

Diagram 20-5. POST Processing (IEAVSY50) (Part 5 of 12)

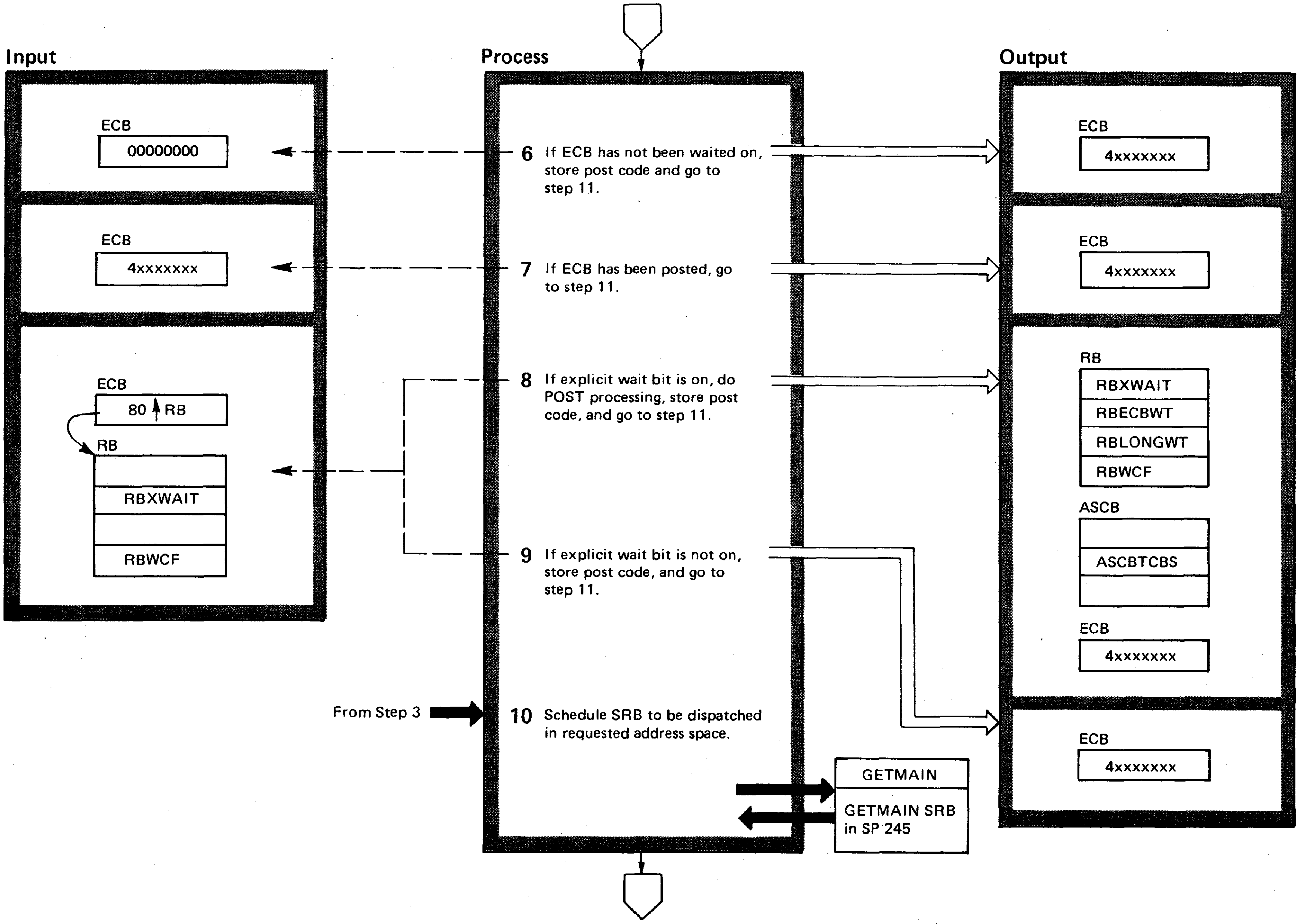
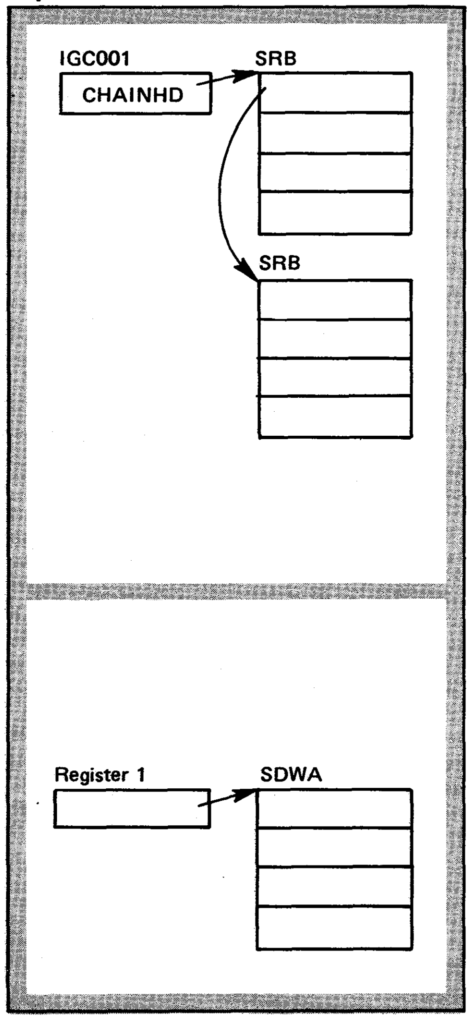

Input
ECB
00000000
ECB
4xxxxxxx
ECB
80 RB
RB
RBXWAIT
RBWCF
Process
6 If ECB has not been waited on, store post code and go to step 11.
7 If ECB has been posted, go to step 11.
8 If explicit wait bit is on, do POST processing, store post code, and go to step 11.
9 If explicit wait bit is not on, store post code, and go to step 11.
From Step 3
10 Schedule SRB to be dispatched in requested address space.
GETMAIN
GETMAIN SRB in SP 245
Output
ECB
4xxxxxxx
ECB
4xxxxxxx
RB
RBXWAIT
RBECBWT
RBLONGWT
RBWCF
ASCB
ASCBTCBS
ECB
4xxxxxxx
ECB
4xxxxxxx

Diagram 20-5. POST Processing (IEAVSY50) (Part 6 of 12)

Extended Description	Module	Label
6 If POST finds neither the wait bit nor complete bit set in the ECB, POST updates the ECB with the post code specified by the caller, sets the complete bit, and goes to step 11.	IEAVSY50	POSTTEST
7 If the ECB has been posted already, no processing is necessary. Go to step 11.		POSTTEST
8 If POST finds the ECB waited on and the explicit wait bit on (RBXWAIT), it will decrease the wait count, set the complete bit and post code. If the wait count is not equal to zero, go to step 11. If the wait count goes to zero, POST resets the wait bit in all the ECBs, if RBECBWT is set (the wait was on a list and the wait count is less than the number of ECBs). POST increases the number of ready TCBs (ASCBTCBS) in the ASCB for use by the Dispatcher, if the post makes the task ready. POST also resets the RBECBWT, RBLONGWT, and RBXWAIT bits to make the RB ready, then goes to step 11.		
9 If POST determines that the ECB-specified RB is not in an explicit wait (RBXWAIT), it posts the ECB as if the wait bit were off, and goes to step 11.		SUPKEY
10 The POST routine gives control to GETMAIN to get the storage for an SRB, if necessary. (*Note:* POST maintains a queue of available SRBs and usually uses these. POST uses GETMAIN only when no storage blocks exist in the queue, but does not use FREEMAIN to free these blocks.) Also, POST puts the ECB address, completion code, and the ERRET address into the SRB parameter list. The ASCB cross memory post queue (XMPQ) is updated (see description of step 20). Then, the SRB is scheduled to the specified address space. The caller gets control back from POST after POST schedules the SRB. The SRB eventually receives control from the Dispatcher to perform the POST request in the specified address space.		QJ12

Diagram 20-5. POST Processing (IEAVSY50) (Part 7 of 12)

Input



Process

11 If not branch entry, return to exit prologue.

To EXIT Prolog (IEAVEEXP)

12 If running under XMPOST SRB, chain the SRB to the XMPOST SRB available queue for XMPOST requests.

Return to caller which entered branch entry (IEA0PT01) or to the Dispatcher (IEAVEDS0)

SRB entry from the Dispatcher (IEAVEDS0)

13 **SRBPOST:**

Execute POST request in the requested address space.

Step 3

From R/TM

14 **SRBFRR:**

Indicate to RTM that retry is to be at SRBRETRY (Step 15).

Return to R/TM

Output

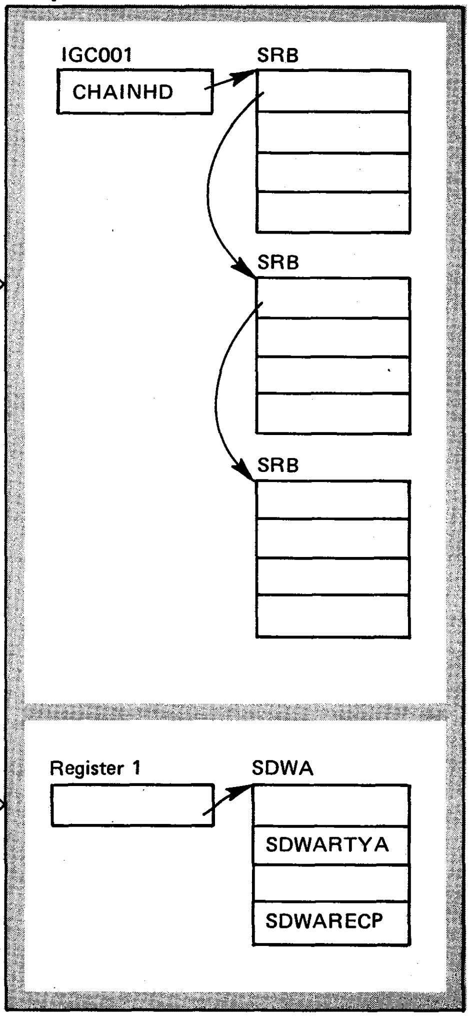
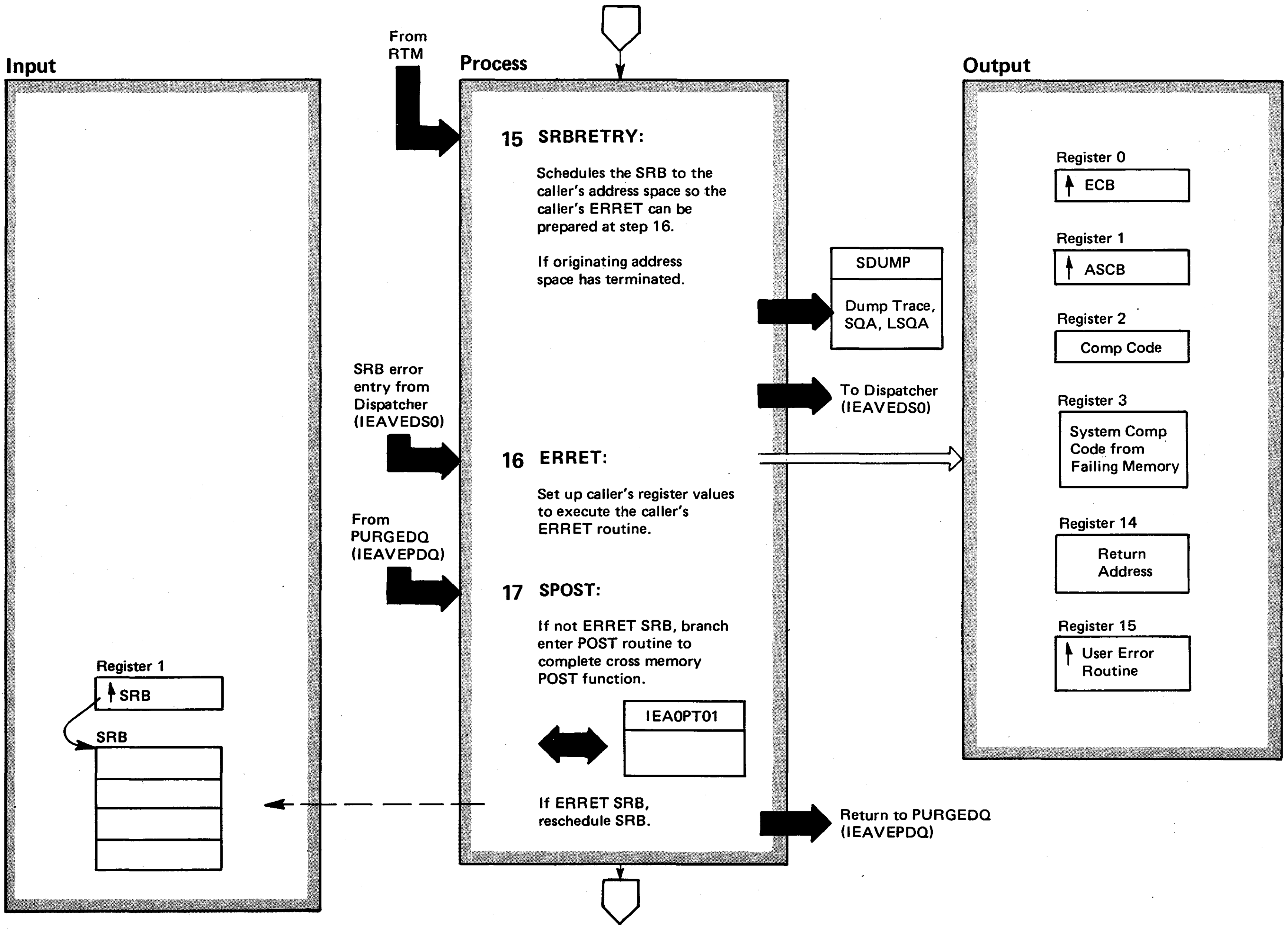


Diagram 20-5. POST Processing (IEAVSY50) (Part 8 of 12)

Extended Description	Module	Label
11 If this was an SVC entry, control is returned to exit prologue.	IEAVSY50	
12 If code was running under XMPOST's SRB, the SRB is put back on the XMPOST available queue. The XMPOST available queue consists of the available SRBs to be used to schedule XMPOST requests. Return to caller.		
13 The scheduled SRB enters POST at this point. XMPOST gets the local lock and extracts the necessary information from the SRB parameter list. Control goes to step 3 at ECBCHECK. Normal POST processing follows, as in a local POST service request (to be executed in the caller's address space). The only differences are that the POST request occurs in an address space other than the one that issued the POST, and that an FRR covers the processing. (Local POST does not have an FRR, except for the EVENTS ECB processing.)		SRBPOST
14 The XMPOST FRR specifies retry at SRB retry (step 15) and gives control to RTM. The XMPOST FRR records the following information: IEAVSY50 IGC001 IGC002		SRBFRR

Diagram 20-5. POST Processing (IEAVSY50) (Part 9 of 12)



15 SRBRETRY:

Schedules the SRB to the caller's address space so the caller's ERRET can be prepared at step 16.

If originating address space has terminated.

16 ERRET:

Set up caller's register values to execute the caller's ERRET routine.

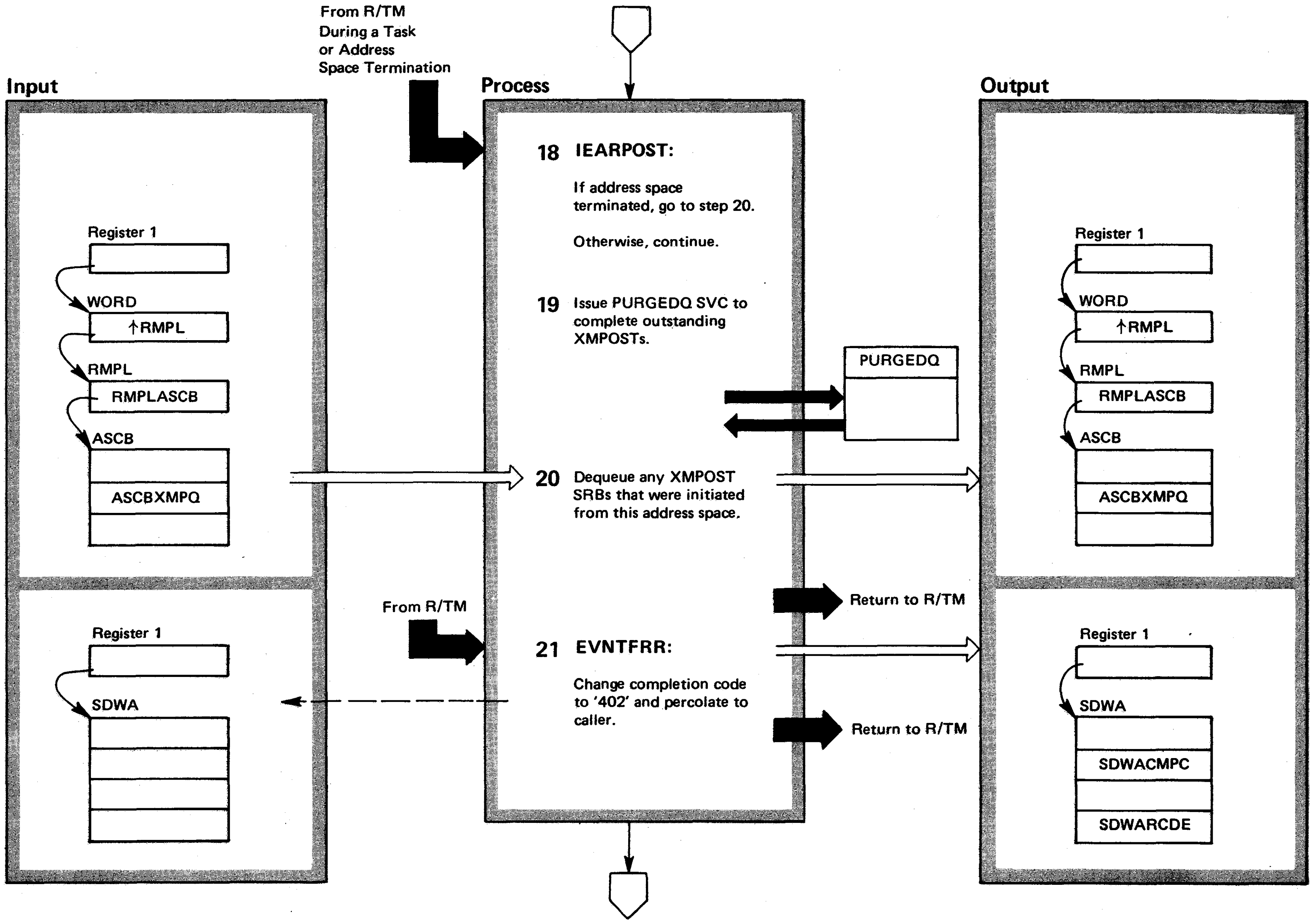
17 SPOST:

If not ERRET SRB, branch enter POST routine to complete cross memory POST function.

If ERRET SRB, reschedule SRB.

Extended Description	Module	Label
15 The POST FRR retry routine schedules an SRB to execute in the caller's address space, unless bit 0 of the ERRET address was on. In this case, the SRB will be scheduled to the master's address space, where the user's error routine (ERRET) will be executed. Control goes to the dispatcher. The dispatcher will dispatch the scheduled SRB to execute at step 16. If the originating address space has gone through termination, and thus no ERRET routine is available, XMPOST branch enters SDUMP to dump trace table, LSQA, and SQA. (See step 20 explanation.) The DUMP header is: IEAVSY50 IGC001 IGC002 XMPOST FAIL – NO ERRET	IEAVSY50	SRBRETRY
16 After the SRB has been put back on the available queue, POST branches, in SRB mode, to the user's error routine with the registers as indicated.		ERRET
17 The XMPOST resource manager termination routine (RMTR) is called by PURGEDQ. The XMPOST RMTR attempts to complete the cross-memory operation. This operation can be the result of the issuance of an SPOST macro or of the issuance of PURGEDQ in IEARPOST task termination.		

Diagram 20-5. POST Processing (IEAVSY50) (Part 11 of 12)



18 IEARPOST:

If address space terminated, go to step 20.

Otherwise, continue.

19 Issue PURGEDQ SVC to complete outstanding XMPOSTs.

20 Dequeue any XMPOST SRBs that were initiated from this address space.

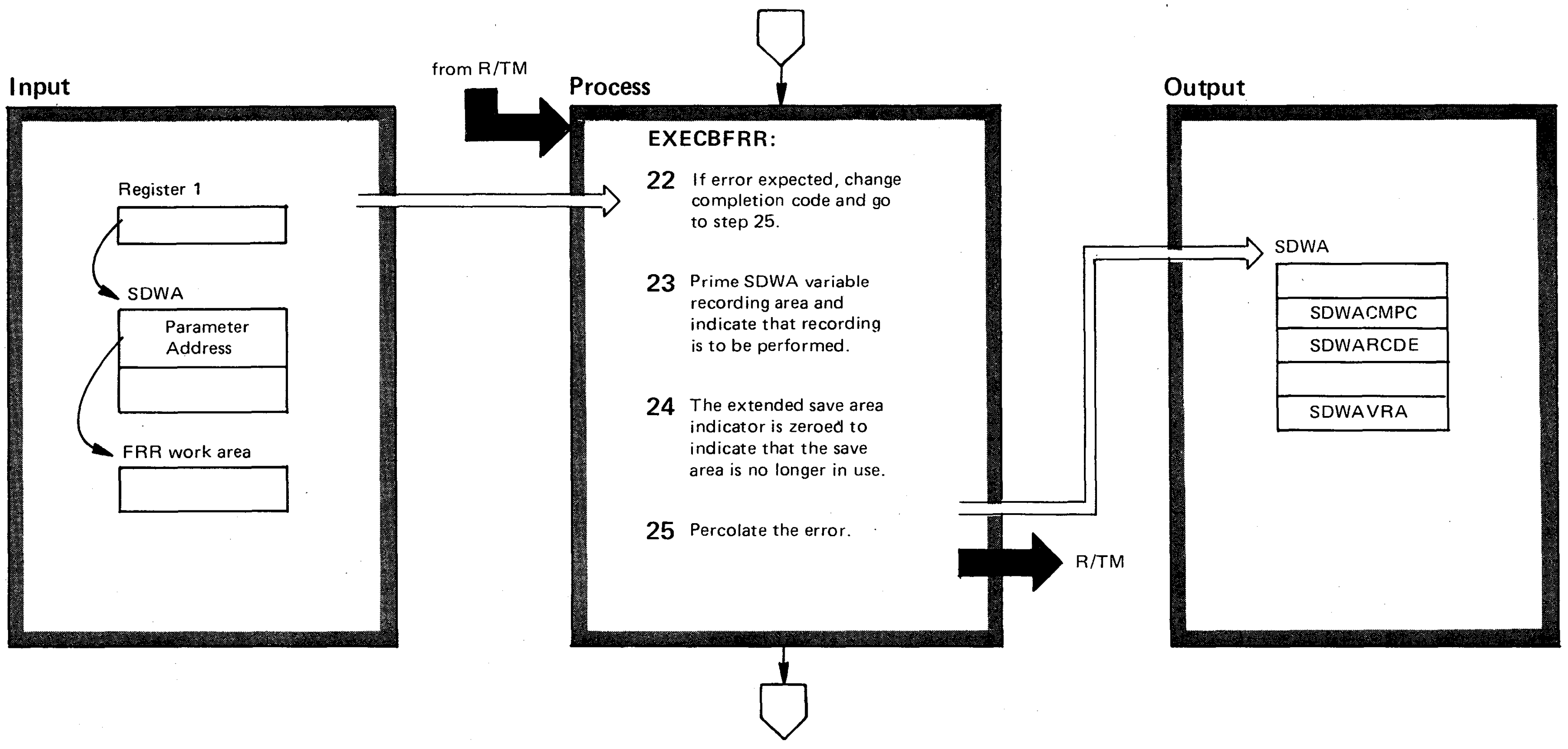
21 EVNTFRR:

Change completion code to '402' and percolate to caller.

Diagram 20-5. POST Processing (IEAVSY50) (Part 12 of 12)

Extended Description	Module	Label
18 For an address space termination, go to step 20. Otherwise, continue.	IEAVSY50	IEARPOST
19 The Task Resource Manager issues a PURGEDQ SVC to complete any outstanding XMPOST requests.		
20 XMPOST maintains a list of XMPOST SRBs called the XMPQ (cross memory post queue) anchored at ASCBXMPQ. These SRBs have originated from this address space. On address space termination, the SRBs are marked as not having an address space to schedule an ERRET SRB to. If ERRET scheduling is attempted, the XMPOST FRR issues an SDUMP to dump the trace table, LSQA, and SQA.		
21 Entry is because failure occurred during EVENTS processing.		EVNTFRR

Diagram 20-5. POST Processing (IEAVSY50) (Part 12.0 of 12)

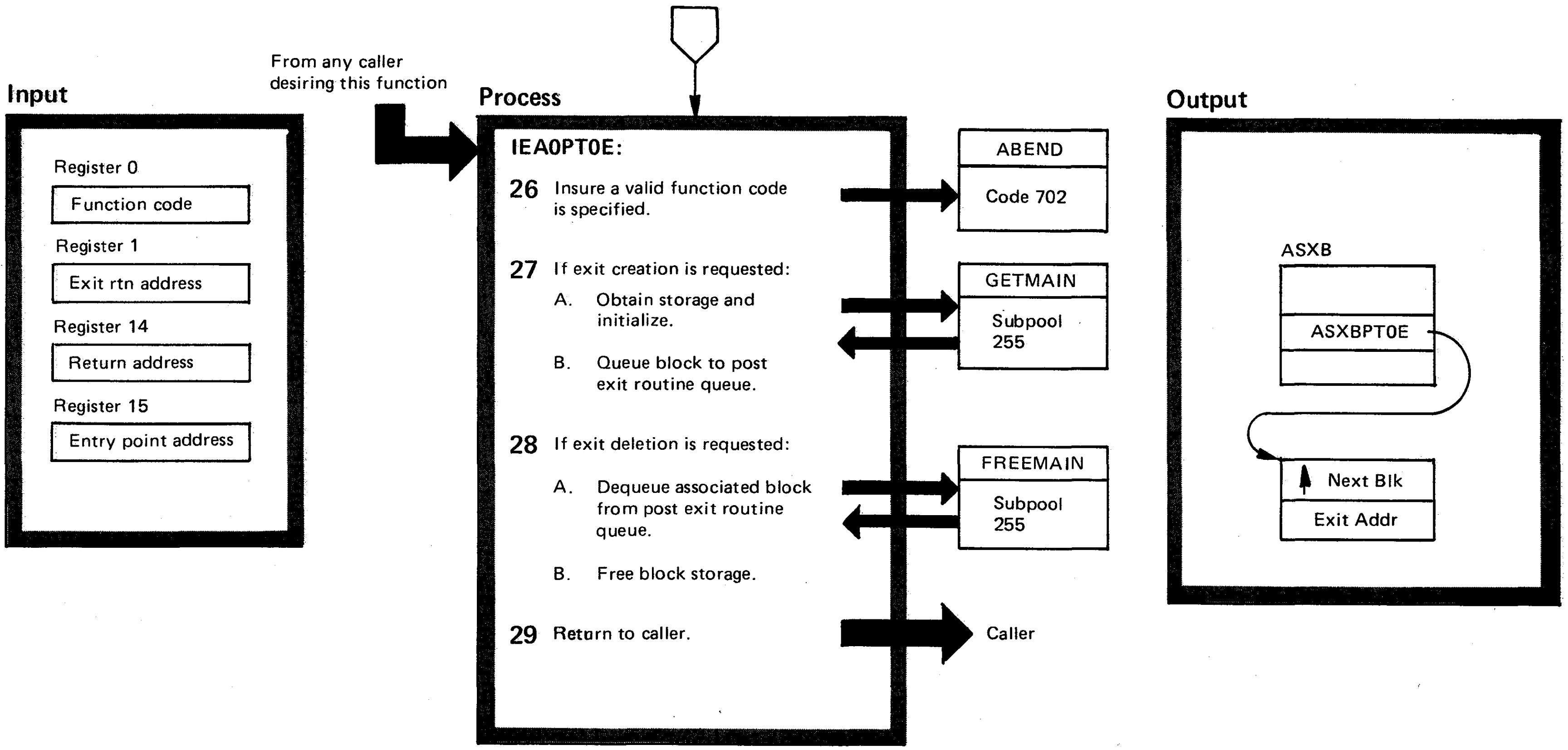


Input
Register 1
SDWA
Parameter Address
FRR work area
from R/TM
Process
EXECBFRR:
22 If error expected, change completion code and go to step 25.
23 Prime SDWA variable recording area and indicate that recording is to be performed.
24 The extended save area indicator is zeroed to indicate that the save area is no longer in use.
25 Percolate the error.
R/TM
Output
SDWA
SDWACMPC
SDWARCDE
SDWAVRA

Diagram 20-5. POST Processing (IEAVSY50) (Part 12.1 of 12)

Extended Description	Module	Label
22 The FRR work area is tested to determine if the error was expected. If value tested is zero, the error was not expected. If non-zero, the completion code is changed to 702 and the reason code is set to X'14'. The error is percolated. If the error was not expected, continue.	IEAVSY50	EXECBFRR
23 The SDWA variable recording area is primed to indicate the associated exit routine and the record indicator is turned on in the SDWA.		
24 The extended save area in-use indicators (located in the POST save area) are zeroed to indicate that the extended save areas are no longer in-use.		
25 Issue SETRP to percolate error.		

Diagram 20-5. POST Processing (IEAVSY50) (Part 12.2 of 12)



IEAOPTOE:

26 Insure a valid function code is specified.

27 If exit creation is requested:

A. Obtain storage and initialize.

B. Queue block to post exit routine queue.

28 If exit deletion is requested:

A. Dequeue associated block from post exit routine queue.

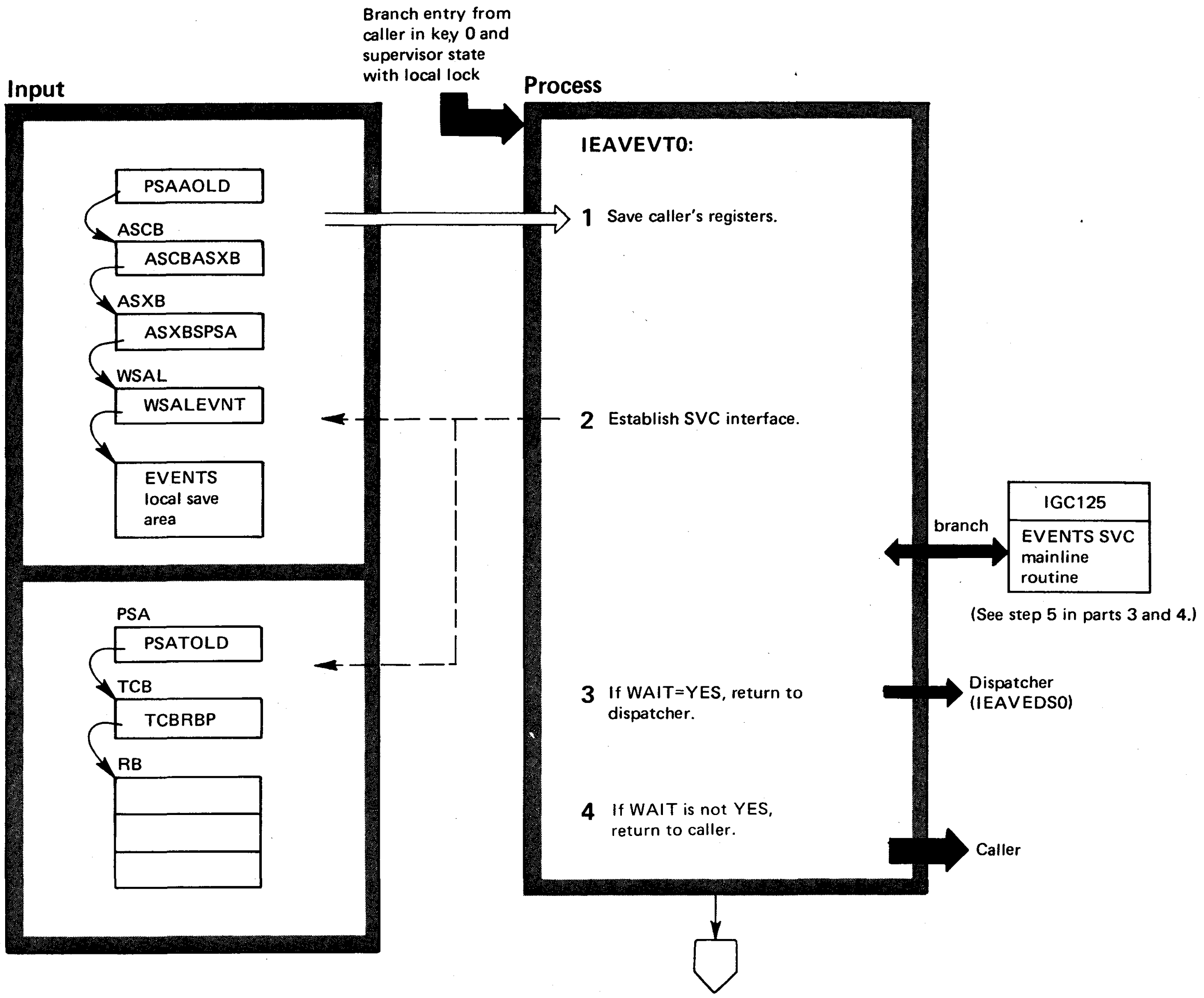
B. Free block storage.

29 Return to caller.

Diagram 20-5. POST Processing (IEAVSY50) (Part 12.3 of 12)

Extended Description	Module	Label
26 This entry point to post currently identifies/deletes exit routine addresses used by the post exit function. As input, register 0 contains the requested function code. A request specifying an undefined function code results in a 702-0 ABEND.	IEAVSY50	IEAOPTOE
27 A function code of 4 is an exit creation request.		
A. GETMAIN is invoked for 8 bytes of LSQA (subpool 255). This block is initialized using the first word as a queue chaining field, and the second word contains the new exit routine address.		
B. The initialized block is placed on the post exit routine queue in a pushdown (last-in, first-out) manner. The header of the post exit routine queue is located in the ASXB (ASXBPTOE).		
28 A function code of is an exit deletion request.		
A. The block containing the associate post exit routine address is removed from the post exit routine queue. Failure to find the block results in a 702-4 ABEND.		
B. FREEMAIN is invoked to free the block storage.		
29 Control is returned to the caller via a branch.		

Diagram 20-6. EVENTS Processing (IEAVEVT0) (Part 1 of 8)



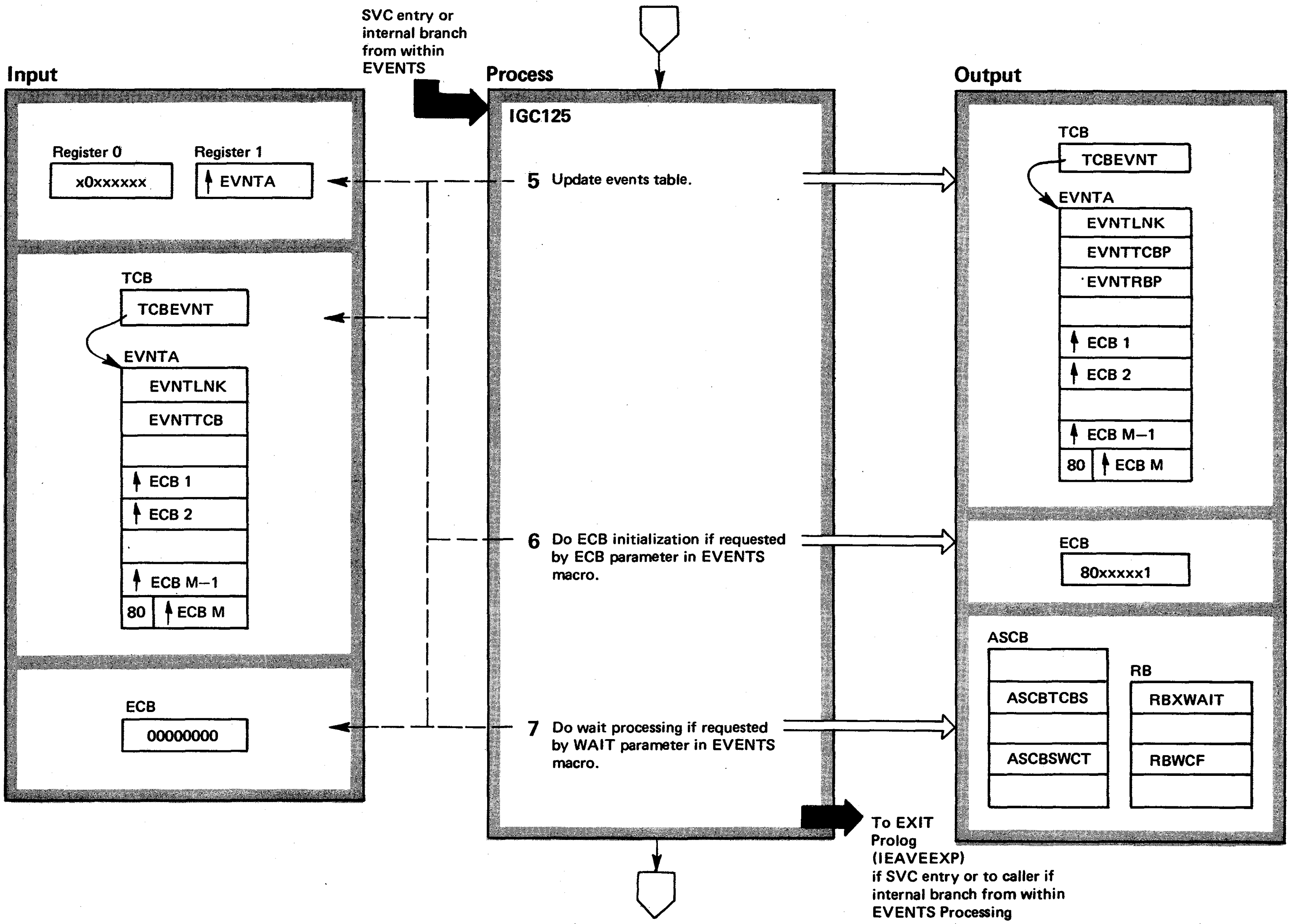
Branch entry from caller in key 0 and supervisor state with local lock
Input
PSAAOLD
ASCB
ASCBASXB
ASXB
ASXBSPSA
WSAL
WSALEVNT
EVENTS local save area
PSA
PSATOLD
TCB
TCBRBP
RB
Process
IEAVEVT0:
1 Save caller's registers.
2 Establish SVC interface.
3 If WAIT=YES, return to dispatcher.
4 If WAIT is not YES, return to caller.
branch
IGC125
EVENTS SVC mainline routine
(See step 5 in parts 3 and 4.)
Dispatcher (IEAVEDS0)
Caller

Diagram 20-6. EVENTS Processing (IEAVEVT0) (Part 2 of 8)

Extended Description	Module	Label
The EVENTS facility allows a user to WAIT on the completion of one of n events and be directly informed by the system which event or events have completed. This is a functional specialization of the current WAIT multiple facility. The EVENTS macro will provide for the creation and deletion of the event table. After the user routine has issued the EVENT macro and the address of the created event table has been returned, the user routine must initialize the ECBs that are to be posted to that table, so that the user routine can be informed of the completion of those events. Each ECB must be initialized by EVENTS in the following manner, so that POST will be able to determine that the ECB is an EVENTS ECB. The high-order byte position will be marked with a X'80' (previously used to indicate a waited-on ECB), and the post code field of the ECB will be initialized with the event table address (previously initialized by WAIT to the waiting RB address). The address will be used to locate an event table which will contain a list of pointers to posted ECBs. Bit 31 of the ECB will be turned on to indicate an EVENTS ECB. Completion of events represented by initialized ECBs is accomplished by the existing system POST facility. Completed events are processed in POST-occurrence order through issuance of the EVENTS macro to the appropriate event table. When the user routine regains control after issuing the EVENTS macro with the WAIT operand, register one points to a list of pointers to posted ECBs. The posted ECBs retain the current format (i.e., the high-order byte contains a hex '40' and the low-order 30 bits contain the completion code).		

Extended Description	Module	Label
1 The caller's registers are saved in the EVENTS local save area (WSALEVNT).	IEAVEVT0	IEAVEVT0
2 The registers are initialized to provide the standard SVC interface, and the type-one SVC mainline (IGC125) is called to do the processing requested by the caller of EVENTS. (See processing that begins at step 5.) Control returns to step 3 from IGC125.		
3 On return from the type-one mainline, a check is made for a WAIT=YES request. If YES, the caller's resume environment is in the caller's RB/TCB. EVENTS will then store register one in the TCB register one save area, purge the FRR stack, disable, free the local lock, and branch enter the dispatcher. Another task can then be dispatched, since the caller's RB is in a wait condition.		
4 If the caller did not specify WAIT=YES, EVENTS will restore registers 2-14 and return to the caller. (Caller's parameters have been processed.)		

Diagram 20-6. EVENTS Processing (IEAVEVT0) (Part 3 of 8)



SVC entry or internal branch from within EVENTS
Input
Register 0
x0xxxxxx
Register 1
EVNTA
TCB
TCBEVNT
EVNTA
EVNTLNK
EVNTTCB
ECB 1
ECB 2
ECB M–1
80
ECB M
ECB
00000000
Process
IGC125
5 Update events table.
6 Do ECB initialization if requested by ECB parameter in EVENTS macro.
7 Do wait processing if requested by WAIT parameter in EVENTS macro.
To EXIT Prolog (IEAVEEXP) if SVC entry or to caller if internal branch from within EVENTS Processing
Output
TCB
TCBEVNT
EVNTA
EVNTLNK
EVNTTCBP
EVNTRBP
ECB 1
ECB 2
ECB M–1
80
ECB M
ECB
80xxxxx1
ASCB
ASCBTCBS
ASCBSWCT
RB
RBXWAIT
RBWCF

Diagram 20-6. EVENTS Processing (IEAVEVT0) (Part 4 of 8)

Extended Description	Module	Label
5 If WAIT=YES or WAIT=NO (bits 0 and 1 of register 0) are specified, bits 8-31 of register 0 (if bit 2 is off) point to the last event entry that the caller has specified. (If LAST parameter was not specified, the caller has processed only one entry.) In either case, EVENTS moves all unprocessed event entries to the top of the event table. The assumption is that the top entry, or all entries up to the last specified by the caller, have been processed by the caller.	IEAVEVT0	IGC125
Entries are placed in the event table, as ECBs complete, by Post if the ECB has been initialized to the EVENTS format (X'80' in the high order byte and the event table address plus one in the low order three bytes). Entries are also placed in the event table by EVENTS to initialize a posted ECB if the ECB parameter was specified in the EVENTS macro. The event entries are added to the event table in FIFO order and the end of list indicator is moved as events are added.		
6 If ECB= is specified (bit 2, register 0 is on), the ECB pointed to by bits 8-31 of register 0 is initialized to the EVENTS format, unless the ECB has already been posted, in which case the address of the ECB will be added to the list of completed events.		

Extended Description	Module	Label
7 If WAIT=YES or WAIT=NO have been specified, EVENTS will check if there are any completed events in the event table. If there are, the address of the first completed event entry will be placed in register 1 as return information for the caller. If WAIT=NO was specified and there are no completed events in the table, register 1 will contain binary zeros. Control will be returned to the caller.		
If WAIT=YES was specified and there are no completed events in the table, EVENTS will set the RB wait count (RBWCF) to one to cause the caller to wait, store the RB address in the event table (EVNTRBP) to indicate that this table is waiting for an event to complete, and decrement the count of ready TCBs (ASCBTCBS). If the count of ready TCBs goes to zero, EVENTS will increment the short wait count (ASCBSWCT). This latter processing is for use by the System Resource Manager (SRM).		

Diagram 20-6. EVENTS Processing (IEAVEVT0) (Part 5 of 8)

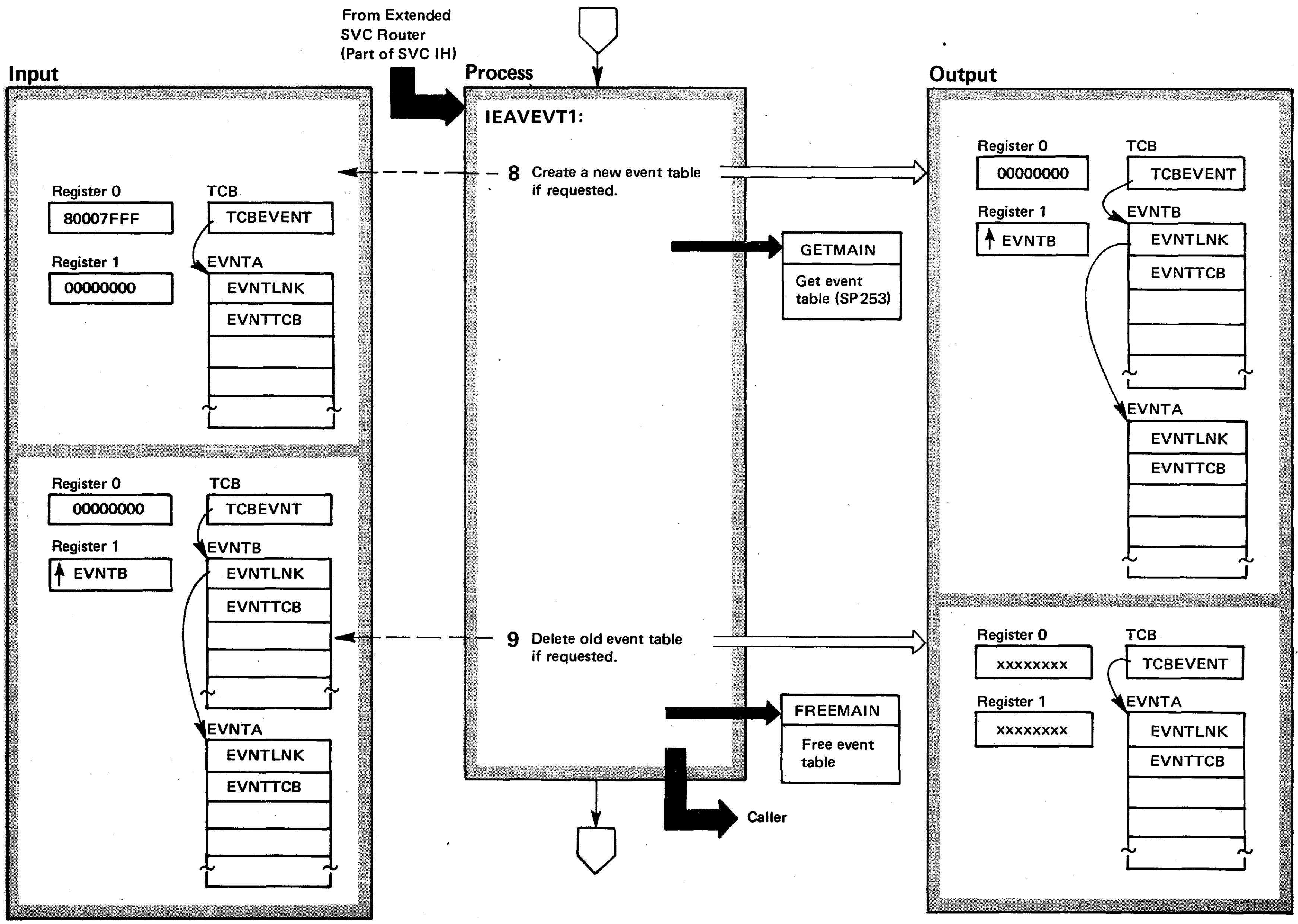
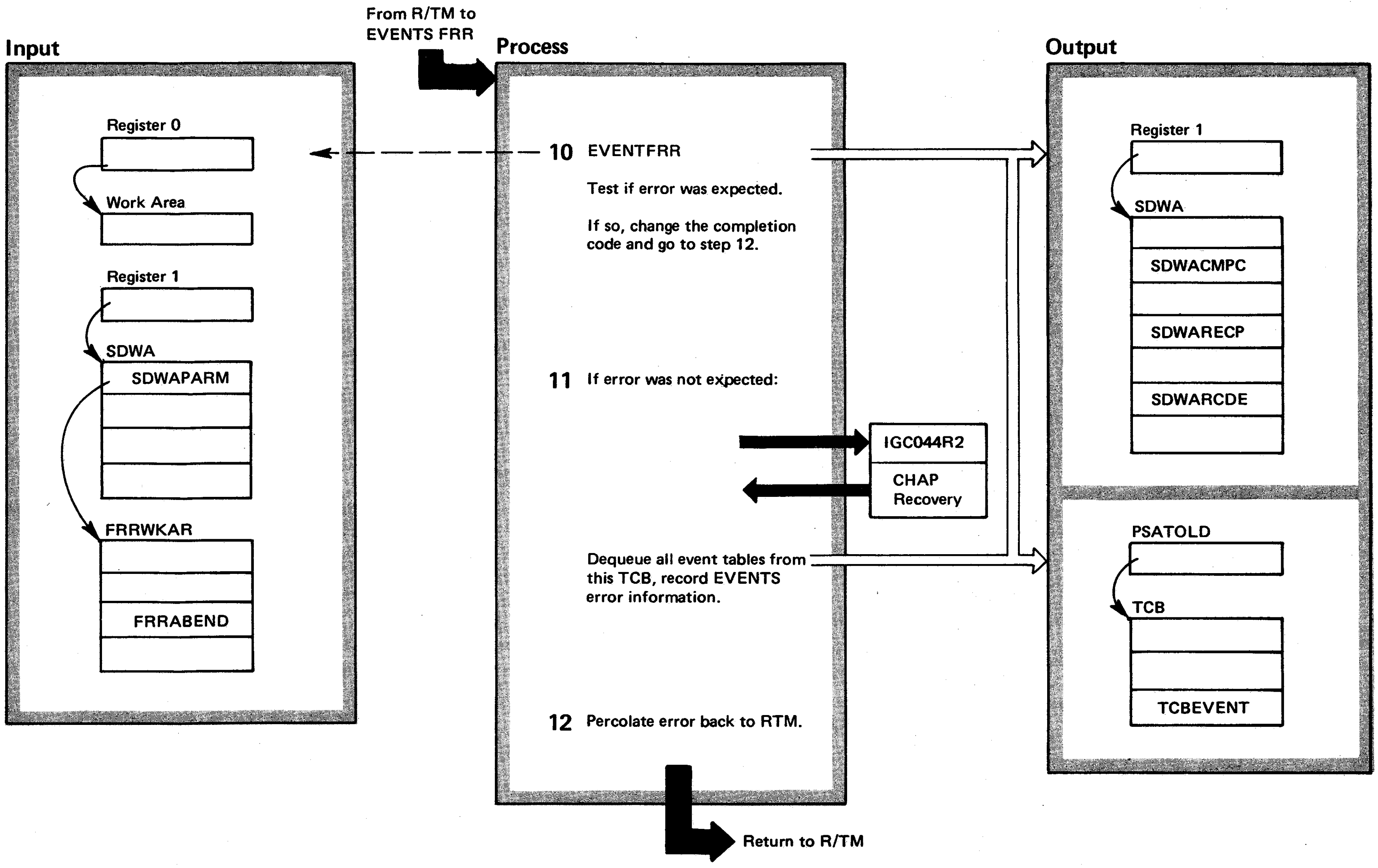


Diagram 20-6. EVENTS Processing (IEAVEVT0) (Part 6 of 8)

Extended Description	Module	Label
8 If register 0 bit 1 is on, a table create has been requested. (EVENTS ENTRIES=n was specified.) Events will compute the table size based on the number of entries requested. (There can be a maximum of 32,767 entries in an event table.) Get the table from SP 253 (task related storage), initialize the event table header to contain a pointer to the requesting TCB, set pointer to the first valid event table entry, to the last valid event table entry, and to the last active event table entry. EVENTS will queue the new event table to the top of the event table queue for the requesting TCB, and return to the caller with the table address in register one.	IEAVEVT0	IEAVEVT1
9 If register 0 is zero and register one contains a table address, the EVENTS macro specified ENTRIES=DELETE. (The caller wants to delete the event table.) EVENTS will locate the table on the requesting TCB's event table queue, dequeue the event table, and free the event table.		

Diagram 20-6. EVENTS Processing (IEAVEVT0) (Part 7 of 8)



10 EVENTFRR

Test if error was expected.

If so, change the completion code and go to step 12.

11 If error was not expected:

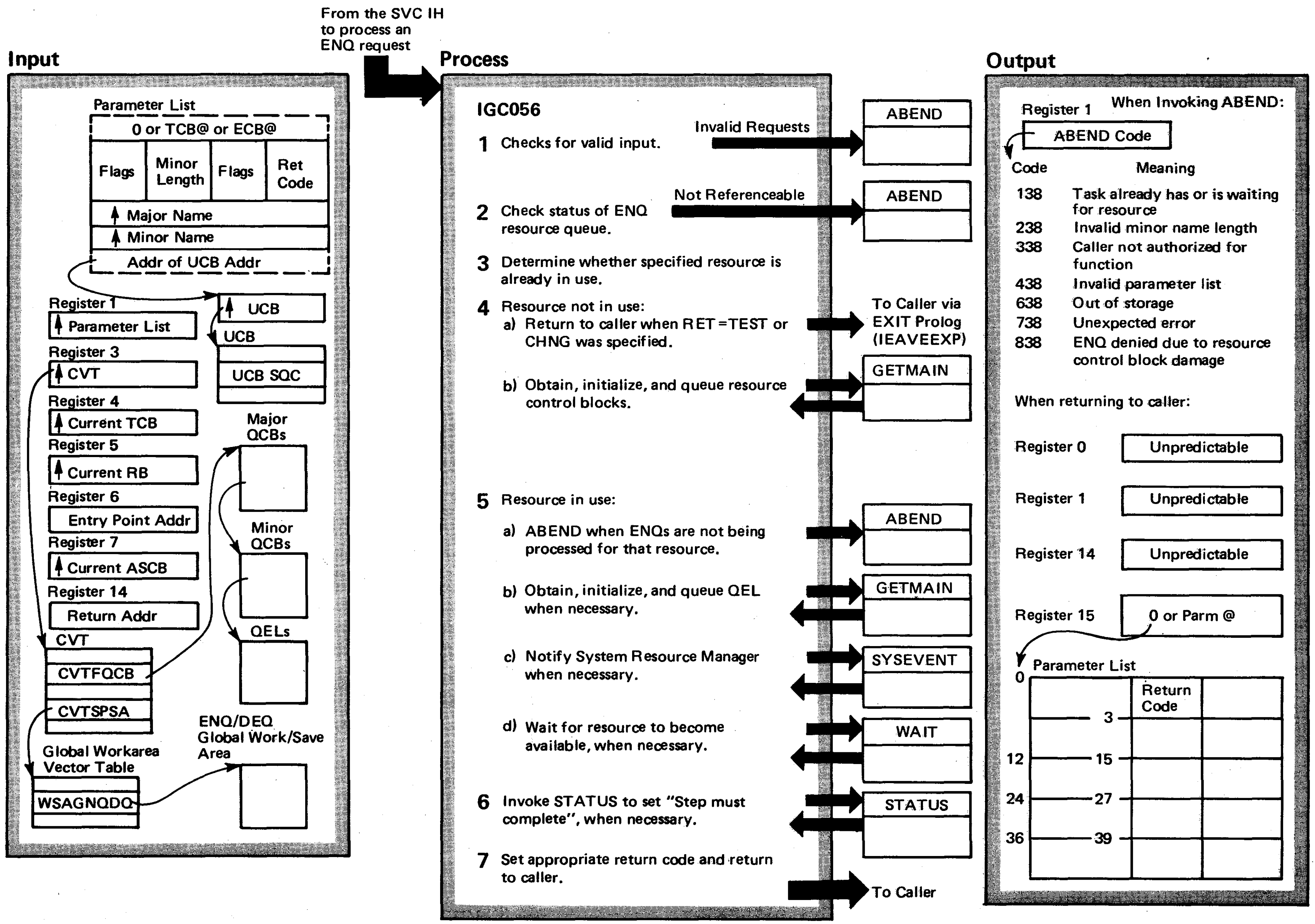
Dequeue all event tables from this TCB, record EVENTS error information.

12 Percolate error back to RTM.

Diagram 20-6. EVENTS Processing (IEAVEVT0) (Part 8 of 8)

Extended Description	Module	Label
10 The FRR work area (FRRABEND) is tested. If it was not zero, the completion code is changed to the contents of this routine's work area (FRRABEND) and the error is percolated back to R/TM. This error can occur when EVENTS tries to store in the ECB in the user's key.	IEAVEVT0	EVENTFRR
11 If the error was not expected (FRRABEND=0), CHAP Error Recovery is called to verify the dispatchability of the TCBs in this address space and to verify the ASCBTCBS count in the ASCB. This action ensures that the rest of the address space is dispatchable in case the error occurred while EVENTS was manipulating TCB dispatchability or updating the count of ready TCBs. When control is received back from CHAP Recovery, EVENTS will dequeue all events tables from this TCB.		
12 Return to R/TM, indicating "continue with percolation".		

Diagram 20-7. ENQ/RESERVE Processing (IEAVENQ1) (Part 1 of 4)



IGC056

1 Checks for valid input.

2 Check status of ENQ resource queue.

3 Determine whether specified resource is already in use.

4 Resource not in use:
   a) Return to caller when RET=TEST or CHNG was specified.
   b) Obtain, initialize, and queue resource control blocks.

5 Resource in use:
   a) ABEND when ENQs are not being processed for that resource.
   b) Obtain, initialize, and queue QEL when necessary.
   c) Notify System Resource Manager when necessary.
   d) Wait for resource to become available, when necessary.

6 Invoke STATUS to set "Step must complete", when necessary.

7 Set appropriate return code and return to caller.

Code	Meaning
138	Task already has or is waiting for resource
238	Invalid minor name length
338	Caller not authorized for function
438	Invalid parameter list
638	Out of storage
738	Unexpected error
838	ENQ denied due to resource control block damage

Diagram 20-7. ENQ/RESERVE Processing (IEAVENQ1) (Part 2 of 4)

The ENQ routine, working with the DEQ routine, permits programs issuing the ENQ macro instruction or the RESERVE macro instruction to gain control of a resource or set of resources. The requested resource may be one or more data sets, records within a data set, programs, or work areas within main storage. ENQ uses the symbolic name of the resource to control access to the resource.

The ENQ routine places in a resource queue all resource requests specified in the caller's macro instruction. If no other ENQ-issuing program is using any of the requested resources, the ENQ routine, via the Exit Prolog routine and the dispatcher, returns control to the caller, and the caller is the owner of the resource(s). But if any requested resource is already in use by another ENQ-issuing program, the ENQ routine may place the caller in a wait condition until the resource becomes available.

Extended Description	Module	Label
1 ENQ passes control to ABEND when the caller issues invalid input. Required authorization is verified, when necessary, by invoking the TESTAUTH macro.	IEAVENQ1	
2 When the ENQ resource queue is not referenceable (indicated by a flag in the ENQ/DEQ Global Work/Save area), control is passed to ABEND. This indicator is set during ENQ/DEQ recovery when either the major or minor QCB queue could not be repaired.		ENQID
3 ENQ searches the resource queues to determine whether the requested resource is already in use. ENQ searches the major QCB queue for a major QCB that contains the specified qname. If it finds the qname, at least one resource in the set of resources is in use, and the routine then searches the associated minor QCB queue for the rname and scope.		XFINDMAJ XFINDMIN
4 The absence of QCBs with the specified qname-rname-scope attributes indicates that the requested resource is not in use. If RESERVE was requested, and the device obtained via the UCB keyword is a shareable direct access device, and the requester has control of the resource, ENQ will increase the UCBSQC count. This causes the I/O Supervisor to "reserve" the device when a user issues I/O to that device.		
a) When RET=CHNG or TEST was specified and the resource was not in use, control is returned to the caller with the appropriate return code (8 or 0 respectively).		
b) A QEL, minor QCB, and major QCB or a QEL and a minor QCB are obtained, initialized, and queued to the appropriate queues. When a major QCB already exists for this resource, one does not need to be obtained.		XGETQEL XGETMIN XGETMAJ

Extended Description	Module	Label
These control blocks are obtained either from storage previously used (and saved) by ENQ or by invoking GETMAIN.		
5 Another requester has access to the resource, as indicated by a major and minor QCB containing the resource names and scope:		
a) When ENQs are being stopped for the specified resource (MINNOENQ on in minor QCB), control is passed to ABEND. This indicator is set during ENQ/DEQ recovery when the QEL queue for this resource could not be repaired.		ENQYMIN
b) This processing depends on the particular RET option that the caller has specified, on the type of request — shared (S) or exclusive (E) — and on the types of QELs already on the queue. When the caller desires to be placed in the queue for the specified resource, a QEL is obtained, initialized, and placed on the QEL queue for that resource. The QEL is obtained either from storage previously used (and saved) by ENQ or by invoking GETMAIN. When all previous QELs on the queue and the present QEL request are both for "shared" control of the resource, the new requester and the previous requesters may simultaneously share the resource. Thus, a requester need not have its QEL at the top of the "shared" group of QELs and still be permitted to access the resource.		XGETQEL
c) When this occurs and the scope of the resource is SYSTEM or SYSTEMS and the current requester is the first to wait for the resource, the Systems Resource Manager is notified, by issuing a SYSEVENT.		XHOLD
d) The requester's willingness to wait for the resource is indicated by a RET option of HAVE, NONE, or the omission of the RET operand. The RET option of TEST never causes creation of a QEL. If RET is USE, a QEL is created only if the requester can have immediate access to the resource.		ENQYEOL
6 When the caller has specified 'SMC=STEP', ENQ will invoke STATUS to perform the "step must complete" function.		XENDUP
7 The appropriate return code is set and control is returned to the caller.		

Diagram 20-7. ENQ/RESERVE Processing (IEAVENQ1) (Part 3 of 4)

ENQ Return Codes

Hexadecimal Code	Meaning
0	For RET=TEST, the resource was immediately available. For RET=USE, RET=HAVE, or ECB=, control of the resource has been assigned to the active task. For RET=CHNG, the status of the resource has been changed to exclusive.
4	For RET=TEST or RET=USE, the resource is not immediately available. For RET=CHNG, the status cannot be changed to shared. For ECB=, the ECB will be posted when available.
8	For RET=TEST, RET=USE, RET=HAVE, or ECB=, a previous request for control of the same resource has been made for the same task. Task has control of resource. For RET=CHNG, the resource has not been queued. If bit 3 is on — shared control of resource; if bit 3 is off — exclusive control.
20	A previous request for control of the same resource has been made for the same task. Task does not have control of resource.

RESERVE Return Codes

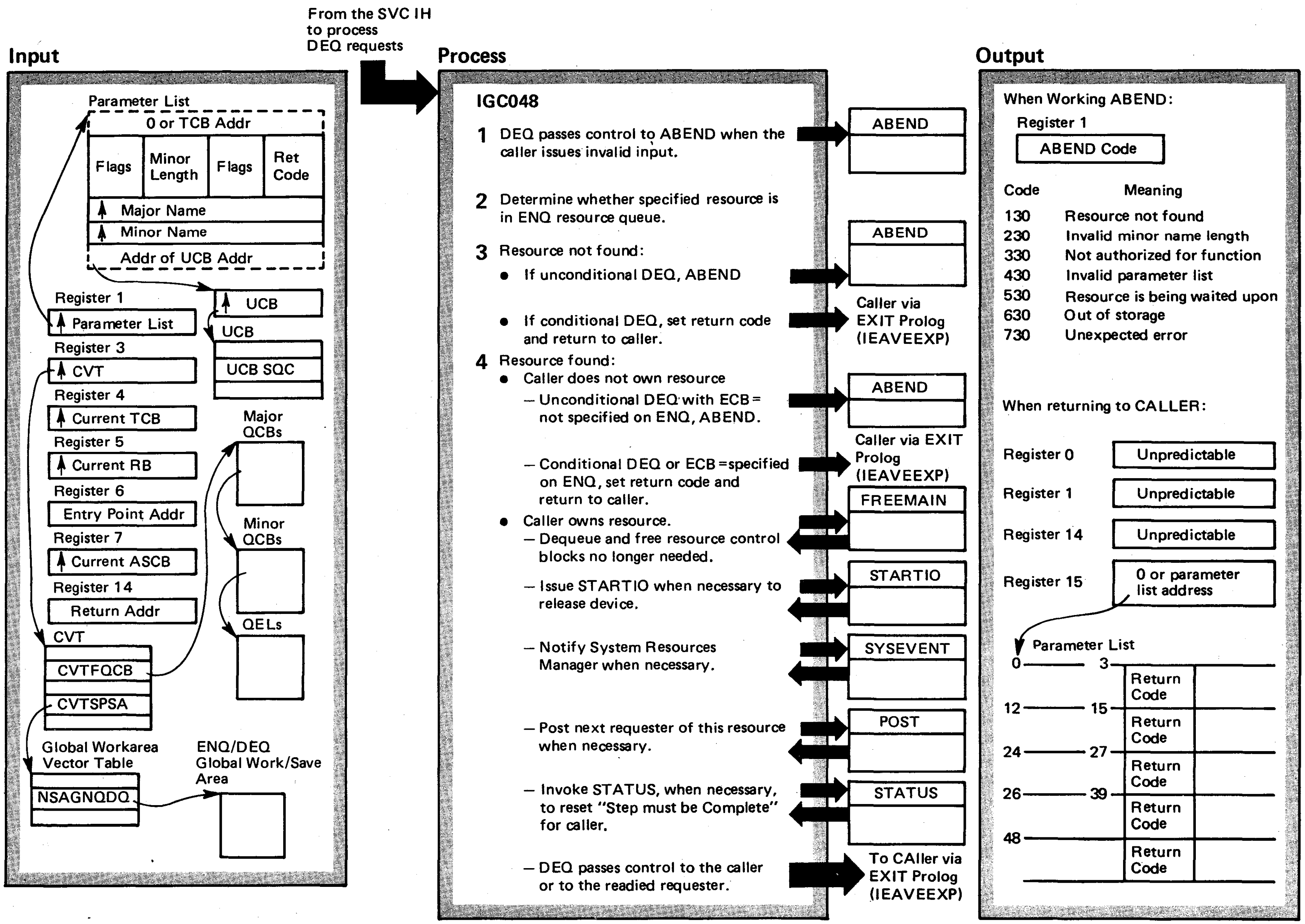
Hexadecimal Code	Meaning
0	For RET=TEST, the resource was immediately available. For RET=USE, RET=HAVE, or ECB, control of the resource has been assigned to the active task.
4	For RET=TEST or RET=USE, the resource is not immediately available. For ECB=, the ECB will be posted when available.
8	A previous request for control of the same resource has been made for the same task. Task has control of resource. If bit 3 is on — shared control of resource; if bit 3 is off — exclusive control.
20	A previous request for control of the same resource has been made for the same task. Task does not have control of resource.

Diagram 20-7. ENQ/RESERVE Processing (IEAVENQ1) (Part 4 of 4)

DEQ Return Codes

Hexadecimal Code	Meaning
0	The resource has been released.
4	The resource has been requested for the task, but the task has not been assigned control. The task is not removed from the wait condition. (This return code could result if DEQ is issued within an exit routine which was given control because of an interruption.)
8	Control of the resource has not been requested by the active task, or the resource has already been released.

Diagram 20-8. DEQ Processing (IEAVENQ1) (Part 1 of 2)



IGC048

1 DEQ passes control to ABEND when the caller issues invalid input.

2 Determine whether specified resource is in ENQ resource queue.

3 Resource not found:
- If unconditional DEQ, ABEND
- If conditional DEQ, set return code and return to caller.

4 Resource found:
- Caller does not own resource
  - Unconditional DEQ with ECB = not specified on ENQ, ABEND.
  - Conditional DEQ or ECB =specified on ENQ, set return code and return to caller.
- Caller owns resource.
  - Dequeue and free resource control blocks no longer needed.
  - Issue STARTIO when necessary to release device.
  - Notify System Resources Manager when necessary.
  - Post next requester of this resource when necessary.
  - Invoke STATUS, when necessary, to reset "Step must be Complete" for caller.
  - DEQ passes control to the caller or to the readied requester.

When Working ABEND:

Register 1: ABEND Code

Code	Meaning
130	Resource not found
230	Invalid minor name length
330	Not authorized for function
430	Invalid parameter list
530	Resource is being waited upon
630	Out of storage
730	Unexpected error

When returning to CALLER:

Register 0	Unpredictable
Register 1	Unpredictable
Register 14	Unpredictable
Register 15	0 or parameter list address

Parameter List

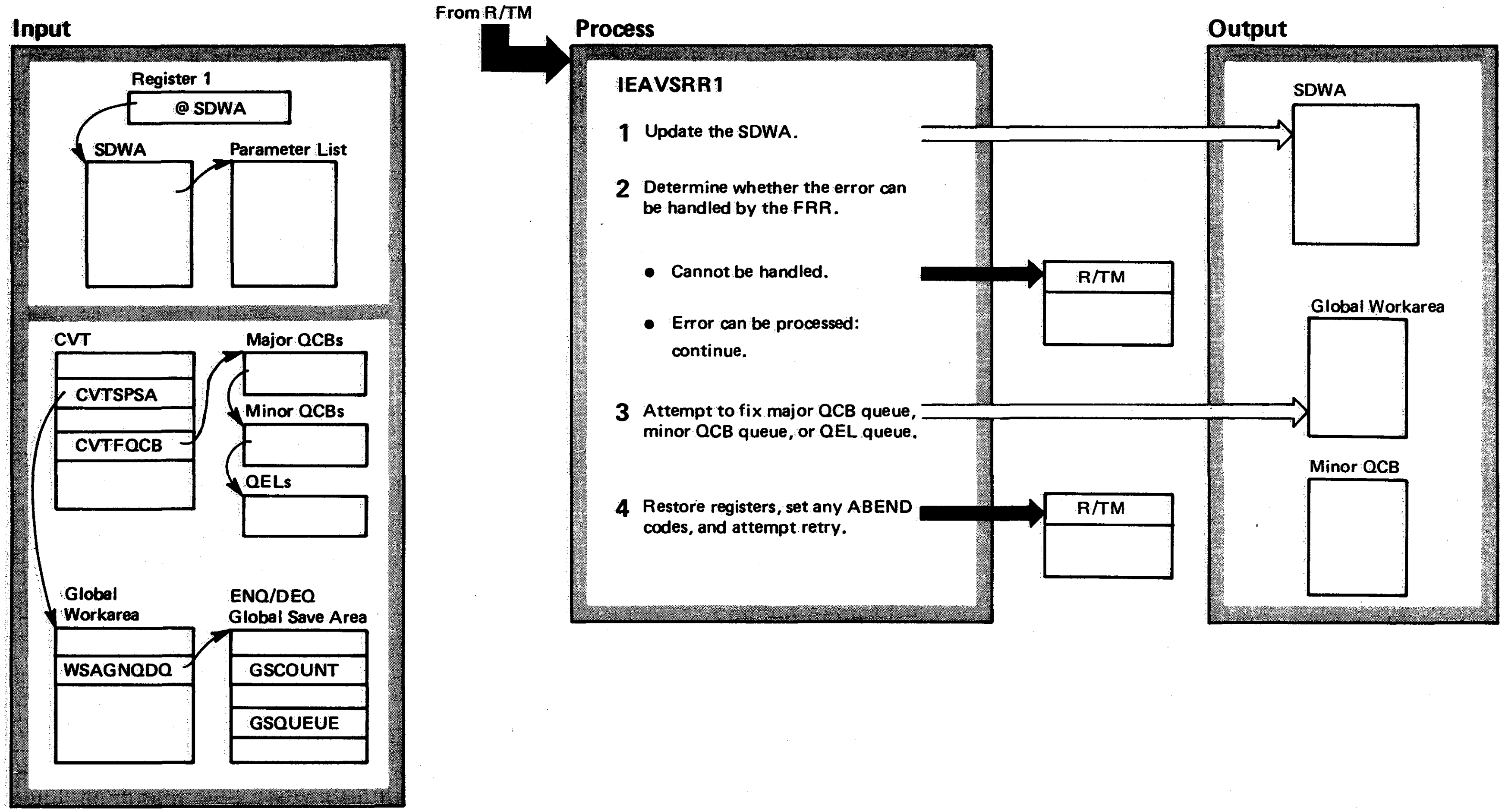
0 — 3	Return Code
12 — 15	Return Code
24 — 27	Return Code
26 — 39	Return Code
48	Return Code

Diagram 20-8. DEQ Processing (IEAVENQ1) (Part 2 of 2)

When the program finishes using the resource(s), it issues a DEQ macro instruction, which causes the DEQ routine to remove one or more elements from the request queue. This may cause other waiting requests to gain control via the POST routine.

Extended Description	Module	Label
1 DEQ passes control to ABEND when the caller issues invalid input.		
2 DEQ searches for the QEL that represents a request that should now be dequeued. It first finds both a major QCB and a minor QCB containing the specified resource names and scope. DEQ then examines the QEL queue associated with the specified resource. If the caller's TCB address matches that stored in one of the QELs, the caller has issued an ENQ for that resource.	IEAVENQ1	XFINDMAJ XFINDMIN
3 When the specified resource request (QEL) is not found, the caller is attempting to DEQ a resource that he is not ENQed on.		DEQNQEL
a) When the caller has requested an unconditional DEQ (RET=NONE), control is passed to ABEND.		
b) When a conditional DEQ was requested, the appropriate return code is set and control returns to the caller.		
4 When the specified resource request (QEL) is found, this indicates that the caller does indeed have an ENQ outstanding for this resource. DEQ scans the QEL queue to determine whether the caller currently owns or shares the resource.		DEQNGENR
a) When the caller does not own or share the resource, the input parameters are checked to determine the action to be taken.		DEQNDEQ1
When an unconditional DEQ is requested (RET=NONE) and the original ENQ did not specify the ECB parameter, control is passed to ABEND.		DEQPART2
When a conditional DEQ was specified or the original ENQ specified the ECB parameter, the appropriate return code is set and control returns to the caller.		DEQPART2
b) When the caller owns or shares the resource, the QEL is dequeued and that storage is saved for future use or freed by invoking FREEMAIN.		XUNCHAIN XFREEQEL
DEQ examines the QCB queues to determine if any QCB may be released. If there are no more QELs queued to the minor QCB, the minor QCB can be released. In this case, DEQ removes the minor QCB from its queue and frees or saves the space it occupies. It then examines the minor QCB queue to decide whether the major QCB is needed and can be similarly eliminated. If there are no minor QCBs queued to the major QCB, DEQ removes the major QCB from its queue and frees or saves its space. DEQ then processes in a similar manner any other input parameters that represent QELs to be dequeued.		XFREEMIN XFREEMAJ
"Reserved" QELs being dequeued from an owning group will cause the UCBSQC count to be decreased. When the count reaches zero, the DEQ routine issues a "STARTIO" instruction. This causes the I/O Supervisor to "release" the shared direct access device.		XDEQQEL
When the scope of the resource being DEQed is SYSTEM or SYSTEMS, System Resources Manager (SRM) is notified that the resource is being released. If subsequently that resource has other requesters, the SRM is notified that once again the resource is being held. Communication to the SRM is via a SYSEVENT.		XRLSE XHOLD
When additional requests are outstanding for the resource being DEQed and the resource is available for use, POST is invoked to notify the appropriate requester(s) that they own the resource.		XPOST
DEQ increases the UCBSQC count for all reserved QELs going from a non-owning group to an owning group. This causes the I/O Supervisor to "reserve" the shared direct access device when a user issues I/O to that device.		
STATUS is invoked to reset "step must complete", when the "RMC=STEP" parameter is specified. DEQ passes control to the caller or to the readied requester.		XENDUP

Diagram 20-9. ENQ/DEQ/RESERVE Recovery (IEAVENQ1) (Part 1 of 2)



IEAVSRR1

1 Update the SDWA.

2 Determine whether the error can be handled by the FRR.

- Cannot be handled.
- Error can be processed: continue.

3 Attempt to fix major QCB queue, minor QCB queue, or QEL queue.

4 Restore registers, set any ABEND codes, and attempt retry.

Diagram 20-9. ENQ/DEQ/RESERVE Recovery (IEAVENQ1) (Part 2 of 2)

Extended Description	Module	Label
1 The ENQ/DEQ FRR updates the SDWA with diagnostic-type information (IEAVENQ1, IGC048, IEASRR1).	IEAVENQ1	

2 The ENQ/DEQ FRR does not attempt to verify the resource queues when:

- The LSQA cannot be addressed.
- The CMS lock was not held at the time of the error.
- The user passed an invalid parameter list.

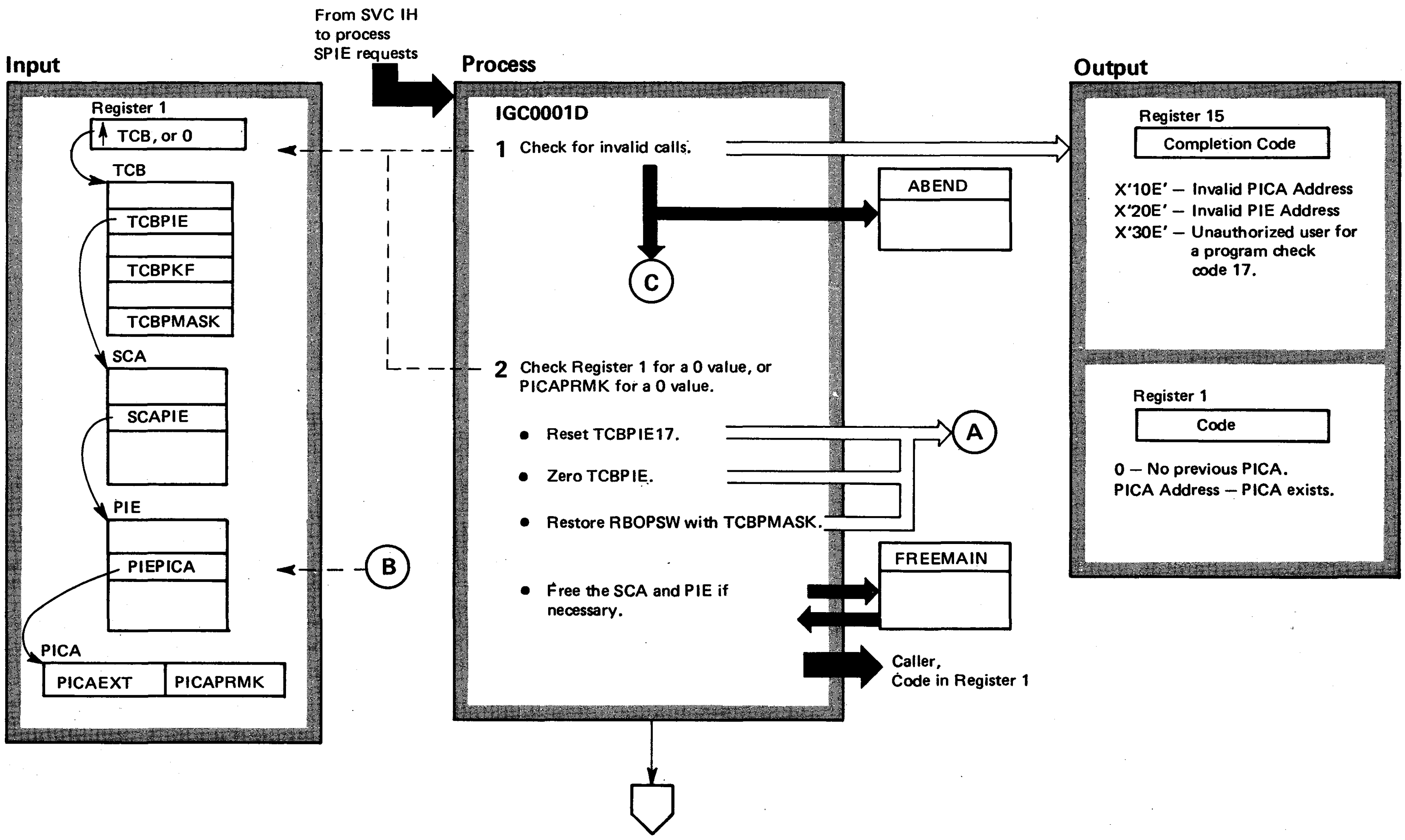
3 The ENQ/DEQ FRR attempts to fix the major QCB queue, minor QCB queue, or QEL queue, if necessary. If the major QCB or minor QCB queue cannot be fixed, the FRR indicates a serious error condition in the GSNOENQ area of the global save area. If the QEL queue cannot be fixed, the FRR indicates a serious error condition in the MINNOENQ field of that minor QCB.

The variable area of the SDWA (SDWAVRA) is updated to reflect the changes made to the resource queues as follows:

Hex Displ	Contents
+0	Count of number of corrections made to queues
+4	Address of ENQ/DEQ module
+8	Address resulting from last BAL instruction of the ENQ/DEQ FRR
+C	Type of control block damaged X'10' – QEL X'14' – minor QCB X'18' – major QCB
+10	Beginning address of invalid address range
+14	Ending address of invalid address range
+18	Image of data contained within invalid address range

4 The ENQ/DEQ FRR restores the values and gives control to R/TM to attempt retry in ENQ or DEQ. The retry will cause the user to ABEND with either a X'738' or X'730' system ABEND code, indicating that an unexpected error was encountered by ENQ or DEQ, respectively.

Diagram 20-10. SPIE Processing (IEAVTB00) (Part 1 of 4)


From SVC IH to process SPIE requests
Input
Register 1
TCB, or 0
TCB
TCBPIE
TCBPKF
TCBPMASK
SCA
SCAPIE
PIE
PIEPICA
PICA
PICAEXT
PICAPRMK
B
Process
IGC0001D
1 Check for invalid calls.
C
ABEND
2 Check Register 1 for a 0 value, or PICAPRMK for a 0 value.
● Reset TCBPIE17.
● Zero TCBPIE.
● Restore RBOPSW with TCBPMASK.
A
FREEMAIN
● Free the SCA and PIE if necessary.
Caller, Code in Register 1
Output
Register 15
Completion Code
X'10E' – Invalid PICA Address
X'20E' – Invalid PIE Address
X'30E' – Unauthorized user for a program check code 17.
Register 1
Code
0 – No previous PICA.
PICA Address – PICA exists.

Extended Description	Module	Label
SPIE processing completes the processing needed for a user to specify a program interruption exit routine. The initial processing – creating and initializing the fields of a PICA (program interruption control area) – is performed by executable coding produced by the expansion of the SPIE macro. This processing places a program mask, the address of the user's program-interruption exit routine, and an interruption mask in the fields of the PICA.		
If, after the execution of the SPIE routine, a program-check interruption occurs in a program being executed for the issuer's task, the user's exit routine processes the program interruption according to the information in the PICA.		
If an interruption occurs, the interruption supervisor stores in the PIE the information needed by the user's exit routine to handle the interruption. This information includes the program check old PSW and registers 14-2.		
For the interruption supervisor to pass control to the correct error handling routine, it must be able to test for the existence of a user routine. The main function of the SPIE routine is to place in the TCB of the macro-issuing program an indirect pointer to the user routine. If, after a program-check interruption has occurred, the supervisor finds an address in the pointer field, it passes control to the user routine to handle the interruption. Otherwise, the supervisor's Program Check IH schedules abnormal termination of the task whose error caused the program interruption.		
The SCA (SPIE control area) contains the SRB the program check IH needs to schedule the user routine.		
SPIE always refers to the PIE and PICA in the key of the caller. Violations will result in a program check error. The SPIE FRR (functional recovery routine) will convert the program check to either a X'10E' or X'20E' ABEND code.		
1 If the caller is in supervisor state, or is in a key other than that indicated in the TCBPKF field, he cannot use SPIE.	IEAVTB00	TEST1
2 Whenever a caller issues a SPIE macro with no operands, a zero PICA (in register 1) results from the macro expansion. The saved program mask (TCBPMASK) is used to set the program mask in the caller's PSW (RBOPSW). Thus, a SPIE macro with no operands cancels the effect of a previous SPIE macro.		SPCANCEL

Diagram 20-10. SPIE Processing (IEAVTB00) (Part 3 of 4)

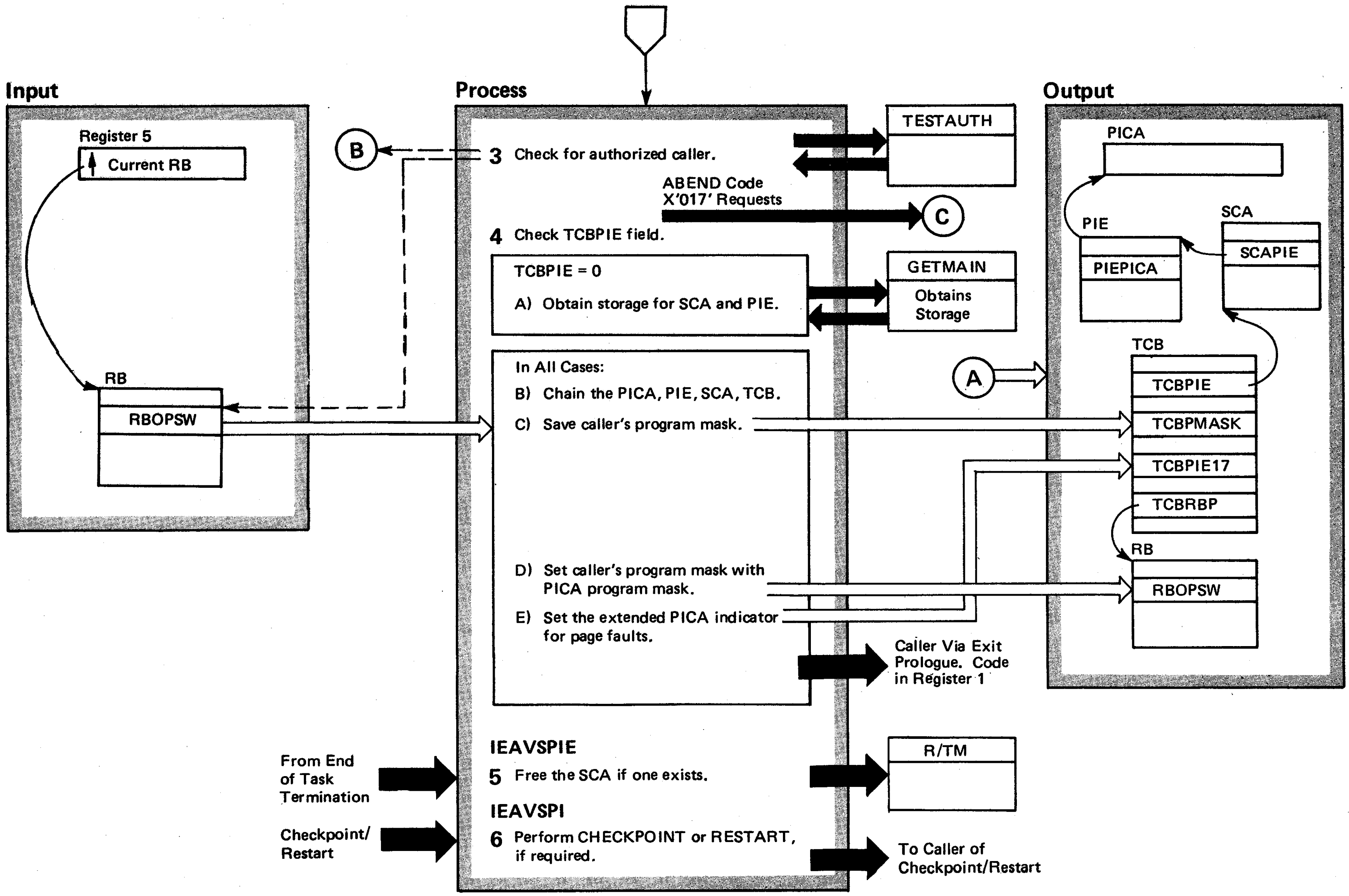
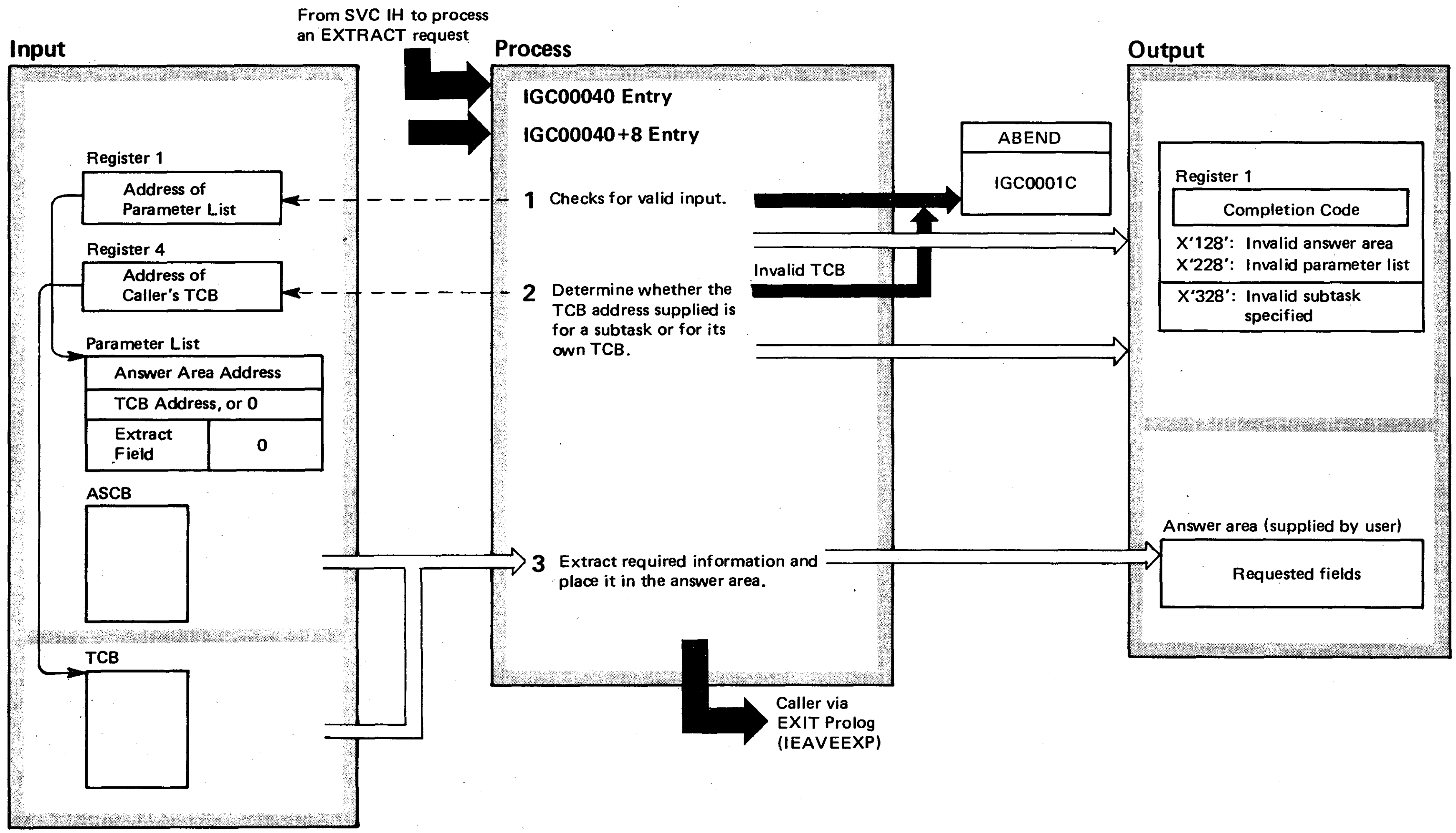

Input
Register 5
Current RB
RB
RBOPSW
Process
3 Check for authorized caller.
B
TESTAUTH
ABEND Code X'017' Requests
C
4 Check TCBPIE field.
TCBPIE = 0
A) Obtain storage for SCA and PIE.
GETMAIN
Obtains Storage
In All Cases:
B) Chain the PICA, PIE, SCA, TCB.
C) Save caller's program mask.
D) Set caller's program mask with PICA program mask.
E) Set the extended PICA indicator for page faults.
A
Caller Via Exit Prologue. Code in Register 1
From End of Task Termination
IEAVSPIE
5 Free the SCA if one exists.
R/TM
Checkpoint/ Restart
IEAVSPI
6 Perform CHECKPOINT or RESTART, if required.
To Caller of Checkpoint/Restart
Output
PICA
PIE
PIEPICA
SCA
SCAPIE
TCB
TCBPIE
TCBPMASK
TCBPIE17
TCBRBP
RB
RBOPSW

Diagram 20-10. SPIE Processing (IEAVTB00) (Part 4 of 4)

Extended Description	Module	Label
3 If PICAEXT does not equal zero, the TCBPIE17 bit is set equal to 1 if the user is authorized. The TCBPIE17 bit makes it possible to avoid inspection of the PIE and PICA every time a missing page interruption occurs. The TCBPIE17 bit equals 1 if the user has provided an exit routine for this type of interruption.		TESTIE
4 If the TCBPIE field equals 0, this is the first time that the caller has issued a SPIE macro. A new SCA and PIE must be built. If the TCBPIE field does not equal 0, a PIE exists from an earlier execution of the SPIE macro. The SPIE routine sets various fields, and control returns to the caller. Register 1 contains the address of the PICA.		SPN017
5 SPIE's resource management gets called at end-of-task. If an SCA exists for the terminating task, it is freed at this time. Control goes to R/TM.		IEAVSPIE
6 SPIE is called by CHECKPOINT/RESTART to save or restore the status of the user's SPIE exit routines.		IEAVSPI

Diagram 20-11. EXTRACT Processing (IEAVTB00) (Part 1 of 2)

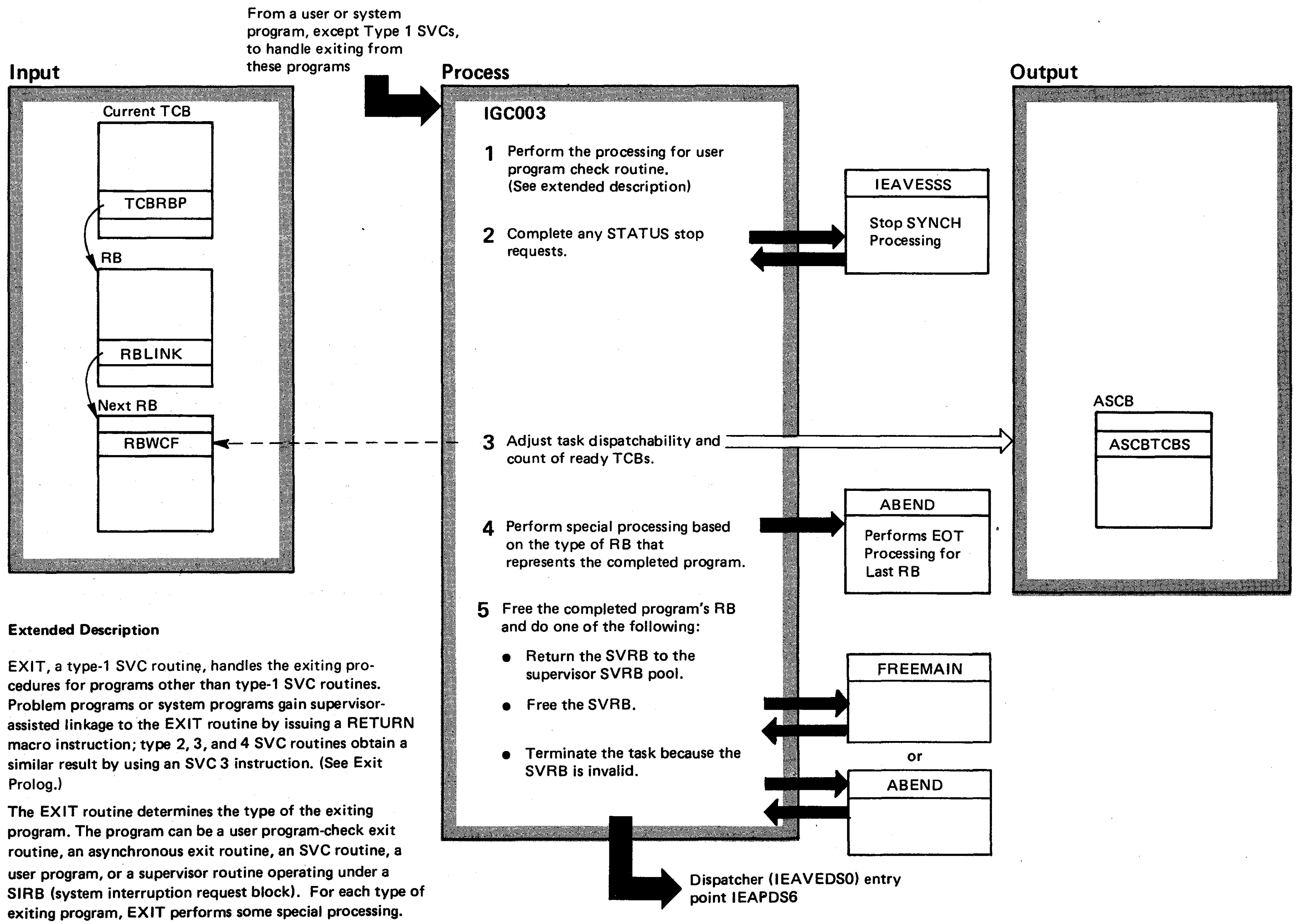


From SVC IH to process an EXTRACT request
Input
Process
Output
Register 1
Address of Parameter List
Register 4
Address of Caller's TCB
Parameter List
Answer Area Address
TCB Address, or 0
Extract Field
0
ASCB
TCB
IGC00040 Entry
IGC00040+8 Entry
1 Checks for valid input.
2 Determine whether the TCB address supplied is for a subtask or for its own TCB.
Invalid TCB
3 Extract required information and place it in the answer area.
ABEND
IGC0001C
Caller via EXIT Prolog (IEAVEEXP)
Register 1
Completion Code
X'128': Invalid answer area
X'228': Invalid parameter list
X'328': Invalid subtask specified
Answer area (supplied by user)
Requested fields

Diagram 20-11. EXTRACT Processing (IEAVTB00) (Part 2 of 2)

Extended Description	Module	Label
EXTRACT processing permits a problem program or system program to request information from its own TCB or the TCB of a subtask. Through the ASCB and TCB, the JSCB (job step control block) and CSCB (command scheduling control block) can be referred to and certain information can be extracted from these control blocks. The information taken from the TCB, ASCB, or subsidiary control block is stored in a caller-specified list in the caller's region.		
Note: On the system generation listing, the entry point name for EXTRACT is IGC0004X, where X means a "12-0" punch.		
1 EXTRACT gives control to ABEND to terminate the caller if any input parameters are not valid. The EXTRACT FRR handles program checks and converts them to appropriate ABEND codes.	IEAVTB00	EXFRR EXABEND
2 EXTRACT considers either the input TCB or the input TCB's subtask valid.		EXLOOP1
3 EXTRACT tests each bit of the extract field in the parameter list. This field represents the FIELDS parameter of the EXTRACT macro instruction. (See *OS/VS System Programming Library: Supervisor* for a list of the TCB fields that can be extracted.) For each bit set, EXTRACT copies appropriate information into the answer area.		EXTCB

Diagram 20-12. EXIT Processing (IEAVEOR) (Part 1 of 2)


From a user or system program, except Type 1 SVCs, to handle exiting from these programs
Input
Current TCB
TCBRBP
RB
RBLINK
Next RB
RBWCF
Process
IGC003
Output
IEAVESSS
Stop SYNCH Processing
ABEND
Performs EOT Processing for Last RB
FREEMAIN
or
ABEND
ASCB
ASCBTCBS
Dispatcher (IEAVEDS0) entry point IEAPDS6

1 Perform the processing for user program check routine. (See extended description)

2 Complete any STATUS stop requests.

3 Adjust task dispatchability and count of ready TCBs.

4 Perform special processing based on the type of RB that represents the completed program.

5 Free the completed program's RB and do one of the following:

- Return the SVRB to the supervisor SVRB pool.
- Free the SVRB.
- Terminate the task because the SVRB is invalid.

Extended Description

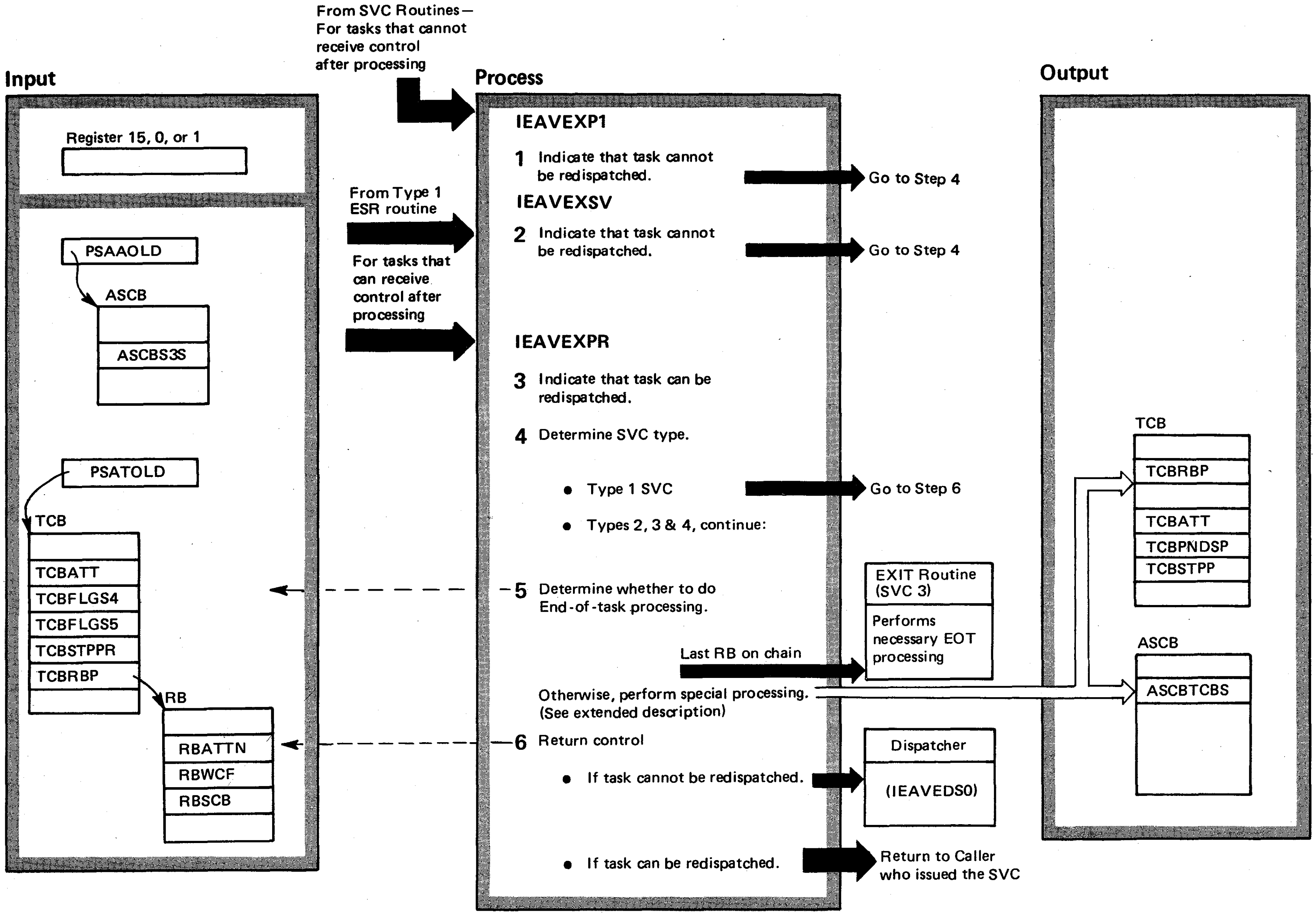
EXIT, a type-1 SVC routine, handles the exiting procedures for programs other than type-1 SVC routines. Problem programs or system programs gain supervisor-assisted linkage to the EXIT routine by issuing a RETURN macro instruction; type 2, 3, and 4 SVC routines obtain a similar result by using an SVC 3 instruction. (See Exit Prolog.)

The EXIT routine determines the type of the exiting program. The program can be a user program-check exit routine, an asynchronous exit routine, an SVC routine, a user program, or a supervisor routine operating under a SIRB (system interruption request block). For each type of exiting program, EXIT performs some special processing.

Diagram 20-12. EXIT Processing (IEAVEOR) (Part 2 of 2)

Extended Description	Module	Label
EXIT considers the first-executed program of a task — the program at the "highest control level" — as an end-of-task condition. Accordingly, EXIT issues SVC 13 routine to perform normal termination of the task.		
The EXIT routine dequeues the RB under which the completed program was operating for all types of completed programs (except user program-check routines, which have no RBs). If the RB had been dynamically acquired, the Exit routine frees the space occupied by the RB.		
The EXIT routine branches to the dispatcher.		
1 User program — check routine (no RB)	IEAVEOR	DOSPIE
• Restore interrupted routine's registers to the TCB general register save area. Registers 14-2 are restored from the PIE to the TCB general register save area.		
• Clear first-time logic switch in the PIE to mark the PIE inactive for the program check IH. An active PIE leads the IH to interpret the program-check interruption as occurring in the program check routine, causing abnormal termination of the current task.		
• Set up the RB old PSW in the interrupted program's RB. The EXIT routine takes the left half from the left half of the SVC old PSW, and the right half from information in the PIE. The PSW information in the PIE is in BC mode.		
The RBOPSW is constructed from two different sources because (1) the user program-check routine has the option of specifying a return point in the interrupted program that is different from the point of interruption, and therefore may store this return address in the right half of the program old PSW in the PIE; and (2) the user program-check routine may have accidentally altered the left half of the program old PSW stored in the PIE.		
2 If no RBs prevent STATUS Stop processing (if the RBATTN field of all the RBs equals 0), reset the TCBATT field of the TCB. Complete STATUS Stop processing, and set the TCBSTPP field. Enter STATUS (IEAVSETS) at IEAVESSS to complete STOP SYNCH processing.		TESTBAR
3 Anything other than a 0 in the RBWCF field of the next RB, or if the STATUS Stop operation finished, indicates that the task going through EXIT has become non-dispatchable. EXIT decreases the ASCBTCBS field in the ASCB to indicate this condition.		CSPROC
4 Exit determines the type of the exiting program by examining the RBSTAB field of its associated request block. This RB is always first on the RB queue when Exit is entered. Depending on the type of RB, the Exit routine performs special processing.		
All RBs		
• Dequeue the RB if it is not the last one on the queue, and mark it inactive.		
• Call the SCBPURGE routine if the RB is the last one on the queue, or if the task has had STAE issued.		TESTRB
SVRBs		
• Move registers 2-14 from the SVRB to the TCB.		DOSVRB
PRBs		
• Call the Program Management subroutine CDEXIT (at entry point IEAPPGMX) to free the programs.	IEAVLK03	DOPRB
IRBs		
• Free RB.		ATTN1
• Move registers 0-15 from the RB to the TCB.		
• Call Attention Exit Epilog for Attention IRBs.		
• For queue element RQEs, return if return has been indicated.		
• Requeue or free the IQEs.		
SIRBs		
• Move registers 0-15 from the RB to the TCB.		DOSIRB
For end-of-task processing, EXIT calls these EOT resource managers; SCBPURGE (IEAVTSBP), Program Management (IEAPPGMX), STATUS (IEAVESSS), Virtual Storage Management (IEAQSPET), and DETACH (IGC062R1).		
5 The EXIT routine returns the SVRB to the supervisor SVRB pool if it was obtained from that pool. If the RB was originally obtained by a GETMAIN, it will be freed by a FREEMAIN. If neither of these conditions can be verified, the task is abended.		TESTREG1 MAINSA2

Diagram 20-13. EXIT Prolog Processing (IEAVEEXP) (Part 1 of 2)



From SVC Routines—For tasks that cannot receive control after processing
Input
Register 15, 0, or 1
PSAAOLD
ASCB
ASCBS3S
PSATOLD
TCB
TCBATT
TCBFLGS4
TCBFLGS5
TCBSTPPR
TCBRBP
RB
RBATTN
RBWCF
RBSCB
From Type 1 ESR routine
For tasks that can receive control after processing
Process
IEAVEXP1
1 Indicate that task cannot be redispatched.
Go to Step 4
IEAVEXSV
2 Indicate that task cannot be redispatched.
Go to Step 4
IEAVEXPR
3 Indicate that task can be redispatched.
4 Determine SVC type.
• Type 1 SVC
Go to Step 6
• Types 2, 3 & 4, continue:
5 Determine whether to do End-of-task processing.
EXIT Routine (SVC 3)
Performs necessary EOT processing
Last RB on chain
Otherwise, perform special processing. (See extended description)
6 Return control
Dispatcher
(IEAVEDS0)
• If task cannot be redispatched.
• If task can be redispatched.
Return to Caller who issued the SVC
Output
TCB
TCBRBP
TCBATT
TCBPNDSP
TCBSTPP
ASCB
ASCBTCBS

Diagram 20-13. EXIT Prolog Processing (IEAVEEXP) (Part 2 of 2)

Extended Description	Module	Label
EXIT Prologue performs the exiting procedure for SVCs. The exiting SVC routine can provide information in registers 0, 1, and 15. Exit Prologue returns these registers to the SVC caller.		
1 EXIT Prolog indicates the caller cannot be redispatched by setting the "Force Dispatch" switch in a register. Some routines cannot be redispatched after EXIT Prolog processing; these routines pass control to the Dispatcher.	IEAVEEXP	IEAVEXP1
2 Some supervisor routines that need entry into the dispatcher use the CALLDISP SVC (Type 1 ESR Code 8), which enters here.		IEAVEXSV
3 EXIT Prolog indicates that the caller can be redispatched after processing.		IEAVEXPR
4 Type 1 SVCs, indicated by the ASCBTYP1 field, complete EXIT Prolog processing by going to Step 6.		GOTYP1
5 The EXIT Prologue routine gives control to EXIT if the last RB on the RB chain represents the caller.		GOTOSVC3

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

Operation	Fields Read	Fields Modified
A) Sets the type 1 switch.		ASCBFLG1 (ASCBTYP1 bit)
B) Complete STATUS Stop processing for the RB unless other RBs indicate that stops cannot be done. Give control to IEAVESSS to perform Stop SYNCH processing.	TCBATT TCBSTPPR	TCBATT TCBSTPPR TCBSTPP TCBPNDSP
C) Decrease the count of ready tasks if the task becomes nondispatchable.	RBLINK RBWCF TCBFLGS4 TCBFLGS5	ASCBTCBS
D) Dequeue the RB and mark it inactive.	RBLINK	TCBRBP
E) Purge any SCBs by giving control to IEAVTSBP.	RBSCB	
F) Move Registers 2-14 into the TCB from the RB.		TCBGRS
G) Return dynamic RBs to the SVRB pool (RBNOCELL=1) or FREEMAIN (RBNOCELL=0).	RBFDYN RBNOCELL	

6 Release all locks and disable; then if the "force dispatcher" switch is set or the task cannot be dispatched (either the RBWCF in the top RB is non-zero or the TCBFLGS4, 5 fields are non-zero, or the Stage 3 Exit Effector Switch (ASCBS3S) is set) the dispatcher is entered, unlocked, and disabled at IEAODS. Otherwise, the current task is redispatched.

Diagram 20-14. STATUS Processing (IEAVSETS) (Part 1 of 6)

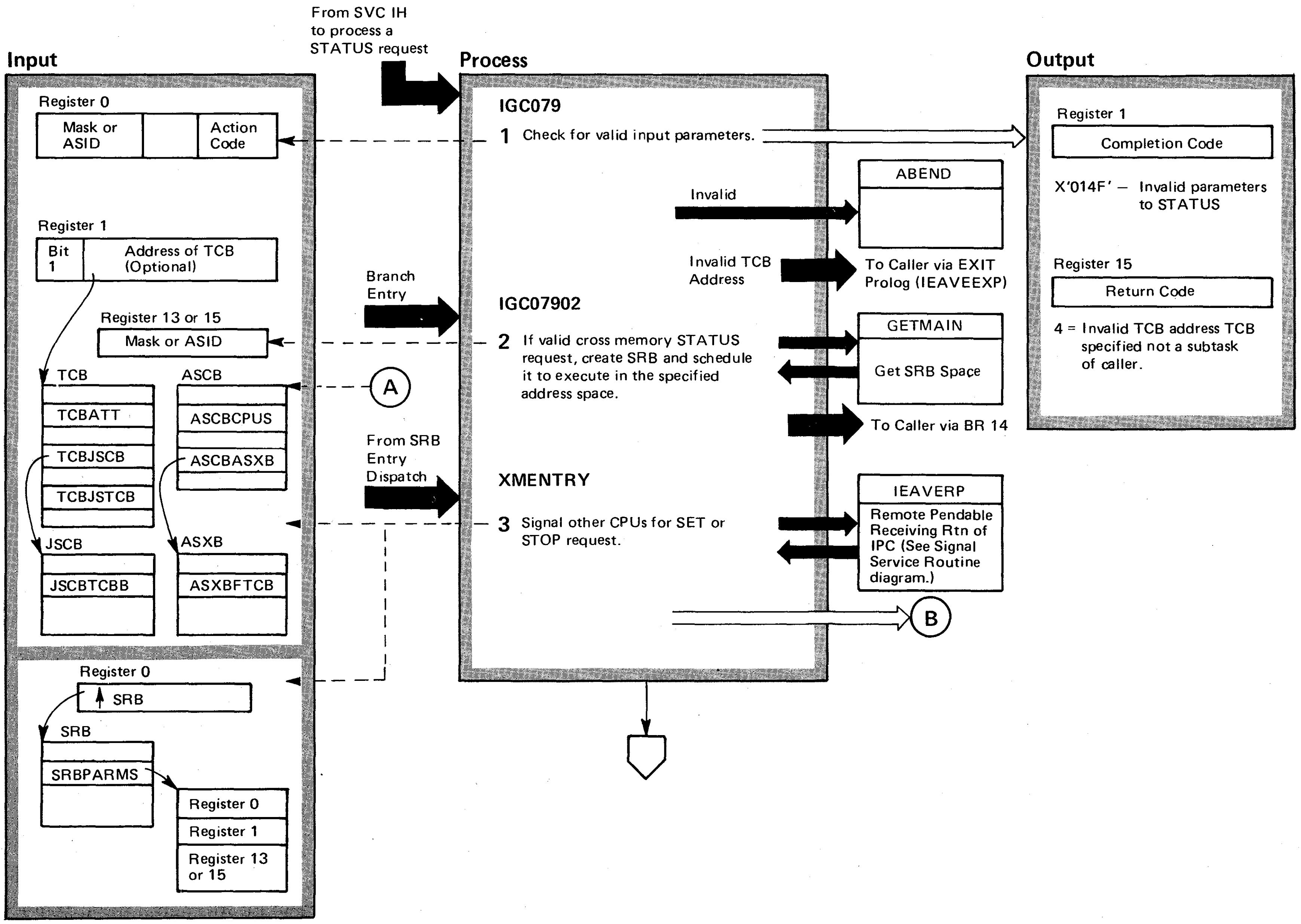
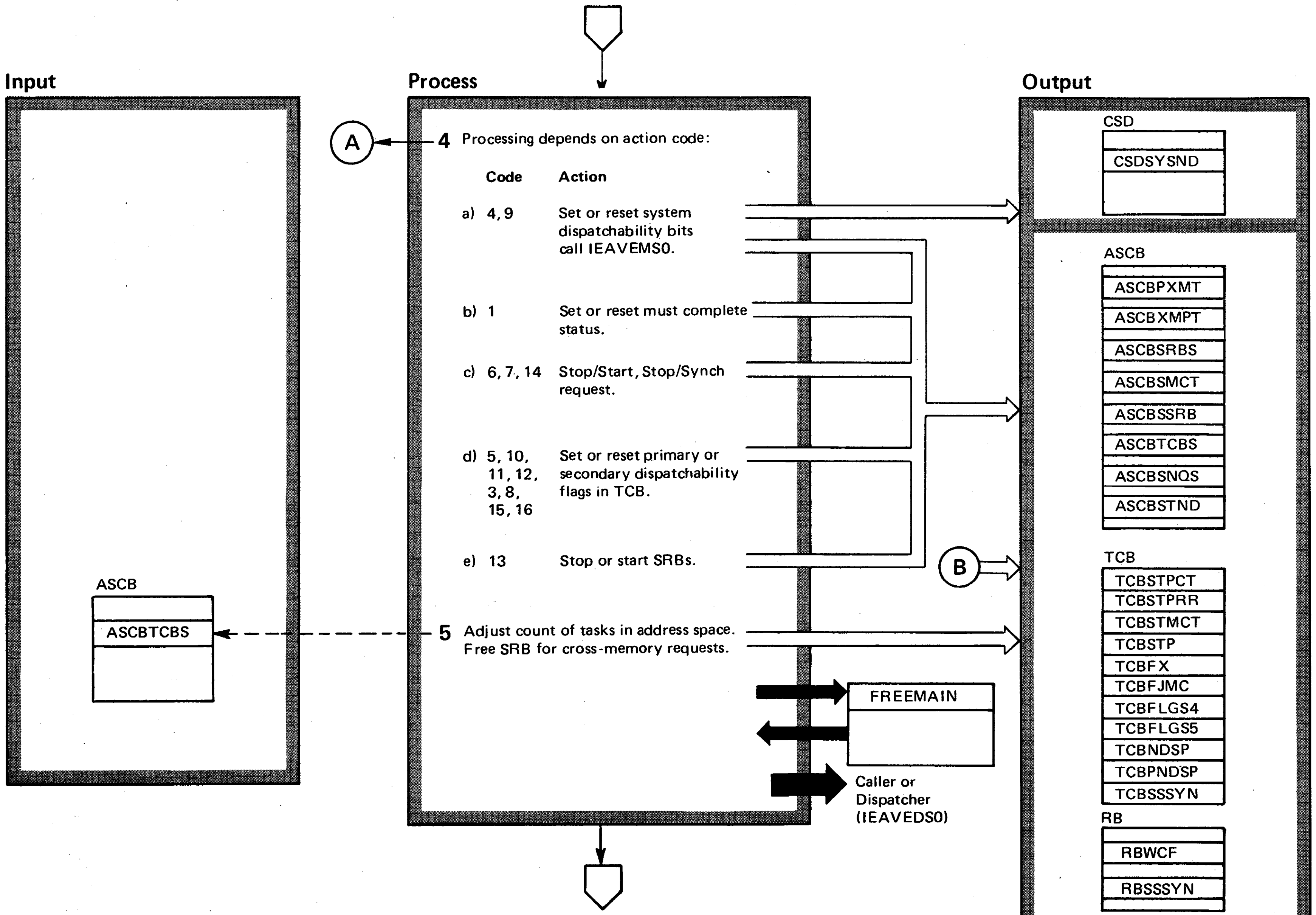

From SVC IH to process a STATUS request
Input
Process
Output
Register 0
Mask or ASID
Action Code
Register 1
Bit 1
Address of TCB (Optional)
Register 13 or 15
Mask or ASID
TCB
TCBATT
TCBJSCB
TCBJSTCB
ASCB
ASCBCPUS
ASCBASXB
JSCB
JSCBTCBB
ASXB
ASXBFTCB
Register 0
SRB
SRB
SRBPARMS
Register 0
Register 1
Register 13 or 15
A
Branch Entry
From SRB Entry Dispatch
IGC079
1 Check for valid input parameters.
Invalid
ABEND
Invalid TCB Address
To Caller via EXIT Prolog (IEAVEEXP)
IGC07902
2 If valid cross memory STATUS request, create SRB and schedule it to execute in the specified address space.
GETMAIN
Get SRB Space
To Caller via BR 14
XMENTRY
3 Signal other CPUs for SET or STOP request.
IEAVERP
Remote Pendable Receiving Rtn of IPC (See Signal Service Routine diagram.)
B
Register 1
Completion Code
X'014F' – Invalid parameters to STATUS
Register 15
Return Code
4 = Invalid TCB address TCB specified not a subtask of caller.

Diagram 20-14. STATUS Processing (IEAVSETS) (Part 2 of 6)

Extended Description	Module	Label
The STATUS routine, used by authorized callers, changes the dispatchability indicators of TCBs, SRBs, ASCBs, a step, or system. This changes the dispatchability of the indicated program. Problem program callers can use STATUS to stop, STOP-SYNCH, or start a particular subtask TCB, or all its subtasks.		
The STATUS routine can perform certain services in an address space other than the one containing the caller. This is called a "cross-memory" function. The requester indicates the cross memory option by including the ASID (address space identifier) parameter in the input parameters. In these cases, STATUS schedules an SRB to the specified address space to complete the service.		
1 STATUS checks for valid input, and passes control to ABEND to terminate callers of invalid parameters. This occurs when a non-supervisor key routine attempts to use a function other than STOP/START TCB or STOP-SYNCH. The ABEND can also occur if an invalid mask is given to STATUS or if the step-must-complete count or stop count is 255 when STATUS is issued.	IEAVSETS	IGC079
2 The STATUS routine gives control to GETMAIN, which gets storage for an SRB. STATUS initializes and schedules the SRB in the address space specified in the ASID, and gives control to the caller. When the SRB gets dispatched, control goes to Step 3.		
3 Since STATUS changes the dispatchability bits for TCBs, SRBs, ASCBs, a step, or system, no other CPU can run in the same address space at that time. Therefore, STATUS issues an RPSGNL macro with the "SWITCH" parameter to ensure no other CPUs are running in the same address space for STOP or SET functions. Before issuing the RPSGNL macro instruction, however, STATUS checks the entire CDAL (common dispatcher active list; that is, the number of currently active dispatchers doing work) to see if any unlocked dispatcher-type functions are active (for example, TCTL). If any dispatcher is active (CDAL entry does not equal zero), STATUS spins on this entry until it becomes zero.		SIGPCPUS

Diagram 20-14. STATUS Processing (IEAVSETS) (Part 3 of 6)

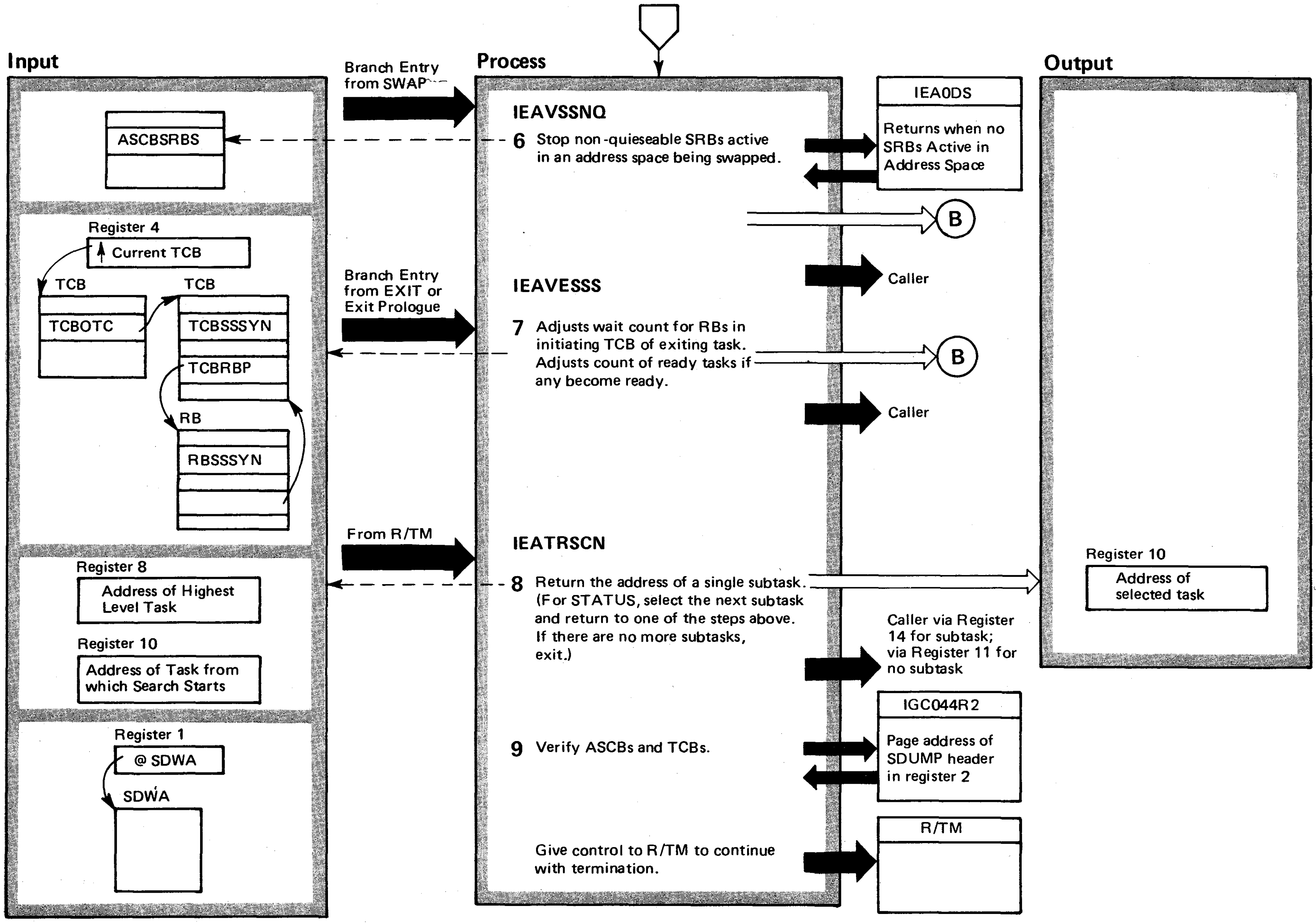


Input
ASCB
ASCBTCBS
Process
A
4 Processing depends on action code:
Code
Action
a) 4, 9
Set or reset system dispatchability bits call IEAVEMSO.
b) 1
Set or reset must complete status.
c) 6, 7, 14
Stop/Start, Stop/Synch request.
d) 5, 10, 11, 12, 3, 8, 15, 16
Set or reset primary or secondary dispatchability flags in TCB.
e) 13
Stop or start SRBs.
5 Adjust count of tasks in address space. Free SRB for cross-memory requests.
FREEMAIN
Caller or Dispatcher (IEAVEDSO)
B
Output
CSD
CSDSYSND
ASCB
ASCBPXMT
ASCBXMPT
ASCBSRBS
ASCBSMCT
ASCBSSRB
ASCBTCBS
ASCBSNQS
ASCBSTND
TCB
TCBSTPCT
TCBSTPRR
TCBSTMCT
TCBSTP
TCBFX
TCBFJMC
TCBFLGS4
TCBFLGS5
TCBNDSP
TCBPNDSP
TCBSSSYN
RB
RBWCF
RBSSSYN

Diagram 20-14. STATUS Processing (IEAVSETS) (Part 4 of 6)

Extended Description	Module	Label
4 STATUS processes 15 different action codes. Figure 2-40 lists the action codes and the fields they change.		
5 When the ASCBTCBS count in the ASCB reaches zero, the Dispatcher will not dispatch any TCBs in that address space. STATUS adjusts the count in the ASCBTCBS field — increases if task becomes dispatchable, or decreases if task is set non-dispatchable.		

Diagram 20-14. STATUS Processing (IEAVSETS) (Part 5 of 6)


Input
Process
Output
Branch Entry from SWAP
Branch Entry from EXIT or Exit Prologue
From R/TM
ASCBSRBS
Register 4
Current TCB
TCB
TCBOTC
TCB
TCBSSSYN
TCBRBP
RB
RBSSSYN
Register 8
Address of Highest Level Task
Register 10
Address of Task from which Search Starts
Register 1
@ SDWA
SDWA
IEAVSSNQ
IEAVESSS
IEATRSCN
IEAODS
Returns when no SRBs Active in Address Space
B
Caller
B
Caller
Caller via Register 14 for subtask; via Register 11 for no subtask
IGC044R2
Page address of SDUMP header in register 2
R/TM
Register 10
Address of selected task

6 Stop non-quieseable SRBs active in an address space being swapped.

7 Adjusts wait count for RBs in initiating TCB of exiting task. Adjusts count of ready tasks if any become ready.

8 Return the address of a single subtask. (For STATUS, select the next subtask and return to one of the steps above. If there are no more subtasks, exit.)

9 Verify ASCBs and TCBs.

Give control to R/TM to continue with termination.

Diagram 20-14. STATUS Processing (IEAVSETS) (Part 6 of 6)

Extended Description	Module	Label
6 The SWAP routine (see the swap-out Processor (IEAVSOUT) diagram in Real Storage Management section) branches to the STATUS routine to stop non-quiescable SRBs. STATUS sets the ASCBSNQS field in the ASCB. STATUS next checks for SRBs running in the address space ready to be swapped. STATUS resets the ASCBSNQS and sets ASCBSTND fields if there are SRBs running; it gives control to the caller if there are no SRBs running. Control goes to the Dispatcher if there are SRBs running. The Dispatcher decreases the ASCBSRBS count when the running SRB exits, and gives control to STATUS when the count goes to 0. This loop continues until there are no more SRBs running in the address space.		
7 Exit checks for a STOP SYNCH request by looking at the TCB stop pending flag. If a STOP SYNCH request exists, EXIT enters the STATUS routine. STATUS decreases the RBWCF field of the requester's RB (requester of STOP SYNCH) by 1. When the RBWCF field reaches 0, STATUS resets the RBSSSYN and TCBSSSYN fields, and increases the count of ready tasks in the ASCBTCBS field of the ASCB.		

Extended Description	Module	Label
8 When entered via the macro instruction STATUS SET, MC, STEP, the STATUS routine sets the caller's task in "step" must-complete status. (If the request specifies the RESET operand, STATUS clears the must-complete status set previously.) The routine sets the must-complete flag in the current TCB, the prohibit-asynchronous-exits flag in the current TCB, and the step "must-complete" dispatchability flag in other TCBs of the job step.		STEPMC
If the request indicates STEP, then all tasks in the job step and the initiator are affected.		
For STEP, the caller's task is always exempt from being set nondispatchable.		
9 The STATUS Recovery routine uses the CHAP recovery routine (IGC044R2) to recover the TCB queues and to verify the current ASCB.		
STATUS passes IGC044R2 the address of the dump header 'IEAVSETSIGC079ƀƀIGC079ƀƀERRORƀINƀSTATUS' to be used for SVC Dump of SQA, LSQA, and the Trace Table.		
STATUS sets recording parameters (SDWARECP) to module name, IEAVSETS, CSECT name, IGC079, and FRR name IGC079.		

Code	Label	Locks Other than Local	Fields Referenced	Fields Set
1	STEPMC		TCBJSTCB JSCBTCBP TCBJSCB	TCBFJMC TCBFX TCBSTMCT ASCBSMC TCBSTP
3, 8	NDSTEP SDSTEP		ASXBFTCB PSATOLD TCBJSTCB	TCBFLGS 4, 5 (P) TCBNDSP (S)
4, 9	NDSYSTEM SDSYSTEM	CMS DISP	ASCBPXMT CVTCSD	ASCBXMPT CSDSYSND
5, 10	NDTCB SDTCB			SAME AS 3, 8
11, 12	EXPLICIT TCB			SAME AS 3, 8
6, 7	STOP/START		TCBATT	TCBSTPCT TCBSTPP TCBPNDSP TCBSTPPR
14	STOP, SYNCH		6 & 7 + TCBRBP TCBFC	6 & 7 + RBSSSYN TCBSSSYN RBWCF
15, 16	CALLER ND, SD			SAME AS 3, 8
13	SRBS	DISP SALLOC		ASXBFTCB ASCBSRBS ASVT TCBSRBND TCBPNDSP ASCBSSRB ASCBSTND PSAANEW

Figure 2-40. STATUS Action Codes and Fields They Change

Diagram 20-15. MODESET Processing (IEAVMODE) (Part 1 of 2)

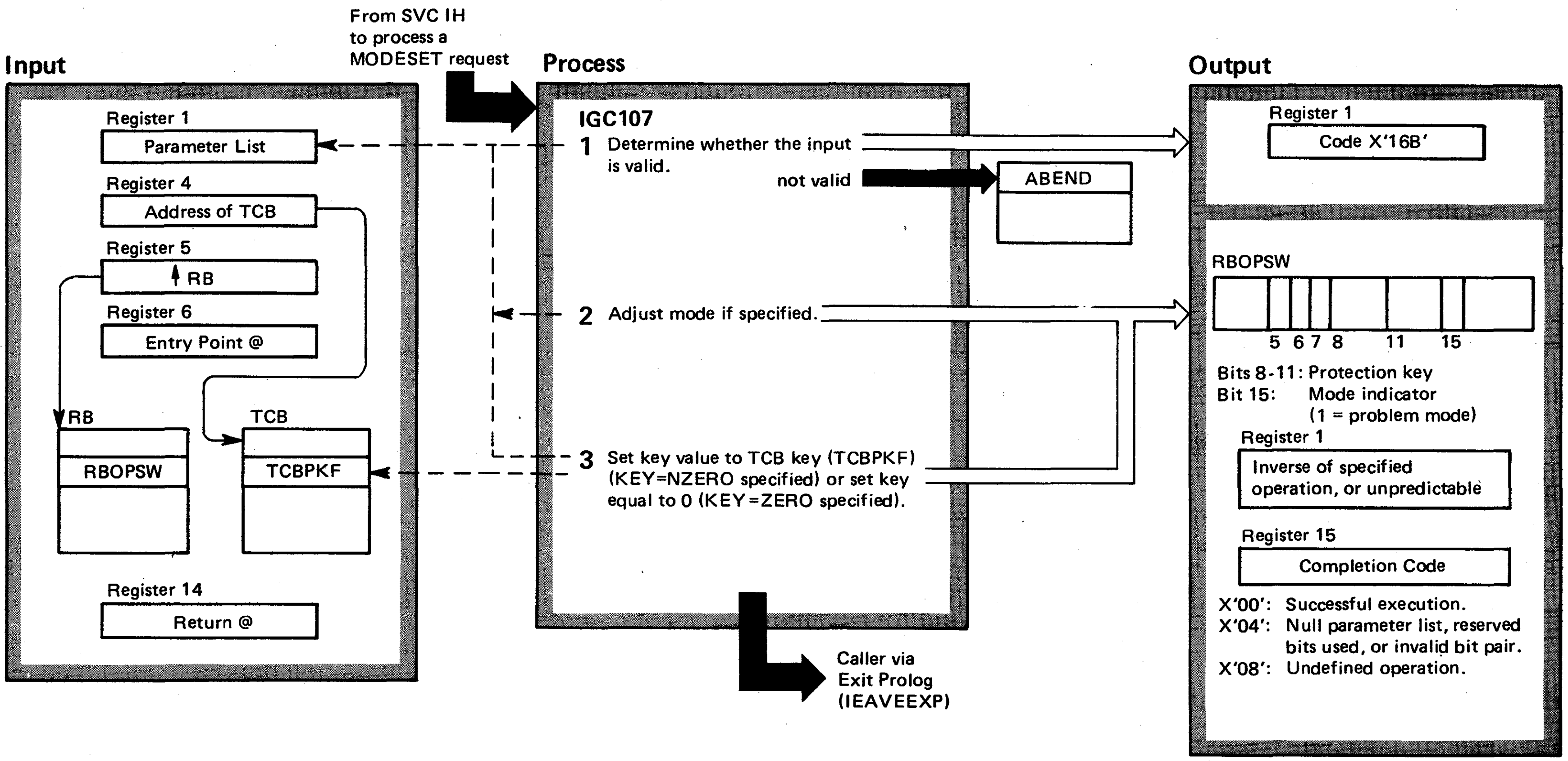

From SVC IH to process a MODESET request
Input
Register 1
Parameter List
Register 4
Address of TCB
Register 5
↑ RB
Register 6
Entry Point @
RB
RBOPSW
TCB
TCBPKF
Register 14
Return @
Process
IGC107
1 Determine whether the input is valid.
not valid
ABEND
2 Adjust mode if specified.
3 Set key value to TCB key (TCBPKF) (KEY=NZERO specified) or set key equal to 0 (KEY=ZERO specified).
Caller via Exit Prolog (IEAVEEXP)
Output
Register 1
Code X'16B'
RBOPSW
5 6 7 8 11 15
Bits 8-11: Protection key
Bit 15: Mode indicator (1 = problem mode)
Register 1
Inverse of specified operation, or unpredictable
Register 15
Completion Code
X'00': Successful execution.
X'04': Null parameter list, reserved bits used, or invalid bit pair.
X'08': Undefined operation.

Diagram 20-15. MODESET Processing (IEAVMODE) (Part 2 of 2)

Extended Description	Module	Label
By entering the MODESET routine through a macro call, an authorized problem program or system program can change its mode and change its protection key. In this case, MODESET alters the RBOPSW, which controls the calling task.		
1 MODESET determines whether the input is valid, and abnormally terminates callers that provide invalid input, with a code of X'16B'.	IEAVMODE	
2 MODESET changes the mode, as indicated by the requester.		
3 MODESET sets a nonzero key (value obtained from TCBPKF field).		

Diagram 20-16. TESTAUTH Processing (IEAVTEST) (Part 1 of 2)

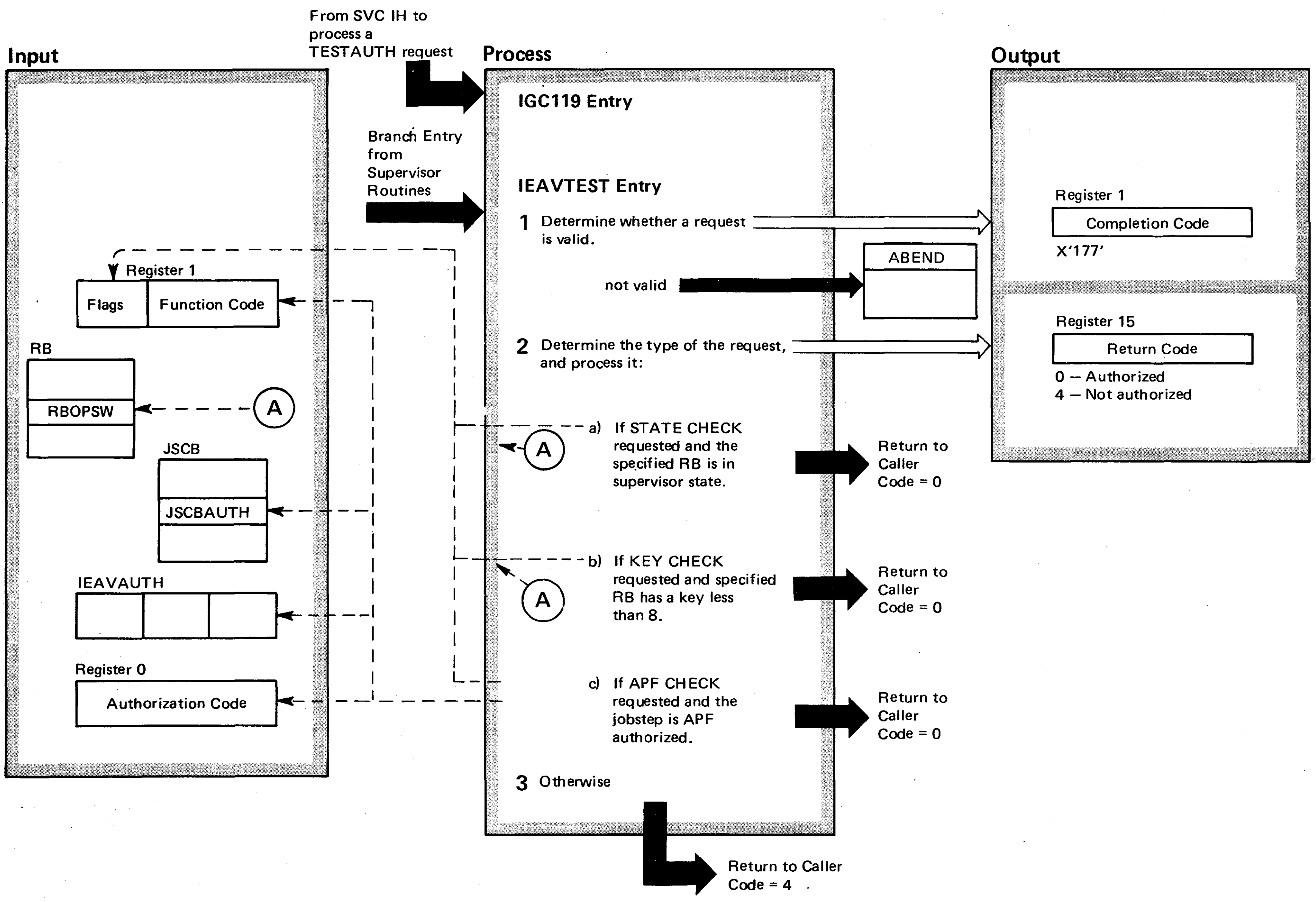

From SVC IH to process a TESTAUTH request
Input
Process
Output
IGC119 Entry
Branch Entry from Supervisor Routines
IEAVTEST Entry
1 Determine whether a request is valid.
not valid
ABEND
Register 1
Completion Code
X'177'
Register 1
Flags
Function Code
RB
RBOPSW
A
2 Determine the type of the request, and process it:
Register 15
Return Code
0 — Authorized
4 — Not authorized
a) If STATE CHECK requested and the specified RB is in supervisor state.
Return to Caller Code = 0
JSCB
JSCBAUTH
b) If KEY CHECK requested and specified RB has a key less than 8.
Return to Caller Code = 0
IEAVAUTH
Register 0
Authorization Code
c) If APF CHECK requested and the jobstep is APF authorized.
Return to Caller Code = 0
3 Otherwise
Return to Caller Code = 4

Diagram 20-16. TESTAUTH Processing (IEAVTEST) (Part 2 of 2)

Extended Description	Module	Label
TESTAUTH processing is called by SVC routines or the SVC IH to test whether a task has the authorization to request a specific function. As input parameters, TESTAUTH accepts flags indicating the request (or requests) desired. If the caller requests APF, TESTAUTH accepts a function code and, optionally, an authorization code. If no authorization code is specified, TESTAUTH uses the job-step authorization, found in the JSCB (job-step control block). The input parameters are indexes to a matrix called IEAVAUTH, which is built in the nucleus during system generation.		
1 TESTAUTH determines whether the requester passes valid input. Control goes to ABEND to terminate the requester if the input is invalid.	IEAVTEST	IEAVTEST
2 TESTAUTH compares the authorization code against the first byte of IEAVAUTH, and compares the function code against the second byte. If either authorization code or function code is greater than X'02', it is invalid. The only valid codes for either parameter are 0, meaning nonrestricted, and 1, indicating restricted. For example, a supervisor routine with an authorization code of 1 can perform both restricted (code 1) and non-restricted (code 0) operations. The authorization and function codes are the indexes to the matrix in the third byte of IEAVAUTH. Using the authorization code as the row identifier, and the function code as the column identifier, TESTAUTH finds the matrix element. Only if the authorization code is 0 and the function code is 1 is the user unauthorized.		RETRY
3 Control returns to the caller with a return code of 4, indicating that the caller does not have authorization.		

Program management services divide into three categories: searching for and scheduling requested modules; synchronizing exit routines to execute during supervisor programs; and fetching modules into storage.

Searching for and scheduling modules consists of:

- Linking to a module. The requester issues a LINK macro instruction to perform this service.
- Loading a module. The requester issues a LOAD macro instruction to perform this service.
- Transfering control to a module. The requester issues an XCTL macro instruction to perform this service.
- Deleting a module. The requester issues a DELETE macro instruction to perform this service.
- Identifying alias names with modules. The requester issues an IDENTIFY macro instruction to perform this service.

The requester issues a SYNCH macro to synchronize exit routines.

Program Fetch brings modules into storage. The requester indirectly calls Program Fetch when he requests a module not in virtual or auxiliary storage. Program management services invoke Program Fetch to bring the requested module into storage.

Searching for and Scheduling Modules

Program management services find a module by scanning control blocks from different queues. These control blocks — the CDE (contents directory element) or LLE (load list element) — form different queues and directories; each queue or directory describes a different part of storage. Then, program management services schedule the requested modules to be executed.

The queues and directories searched by program management are:

- The JPA (job pack area) storage areas.
- The LPA (link pack area) storage areas.
- The auxiliary storage libraries.

JPA Storage Areas

The JPA (job pack area) in virtual storage contains modules needed for the execution of jobs. The JPA resides in subpools 251 and 252 of a region. Problem programs, including TSO tasks, execute in the JPA. Modules in the JPA may be executed only by the user in whose region they are stored.

These are three JPA storage areas:

- The JPA.
- The job pack area queue.
- The load list.

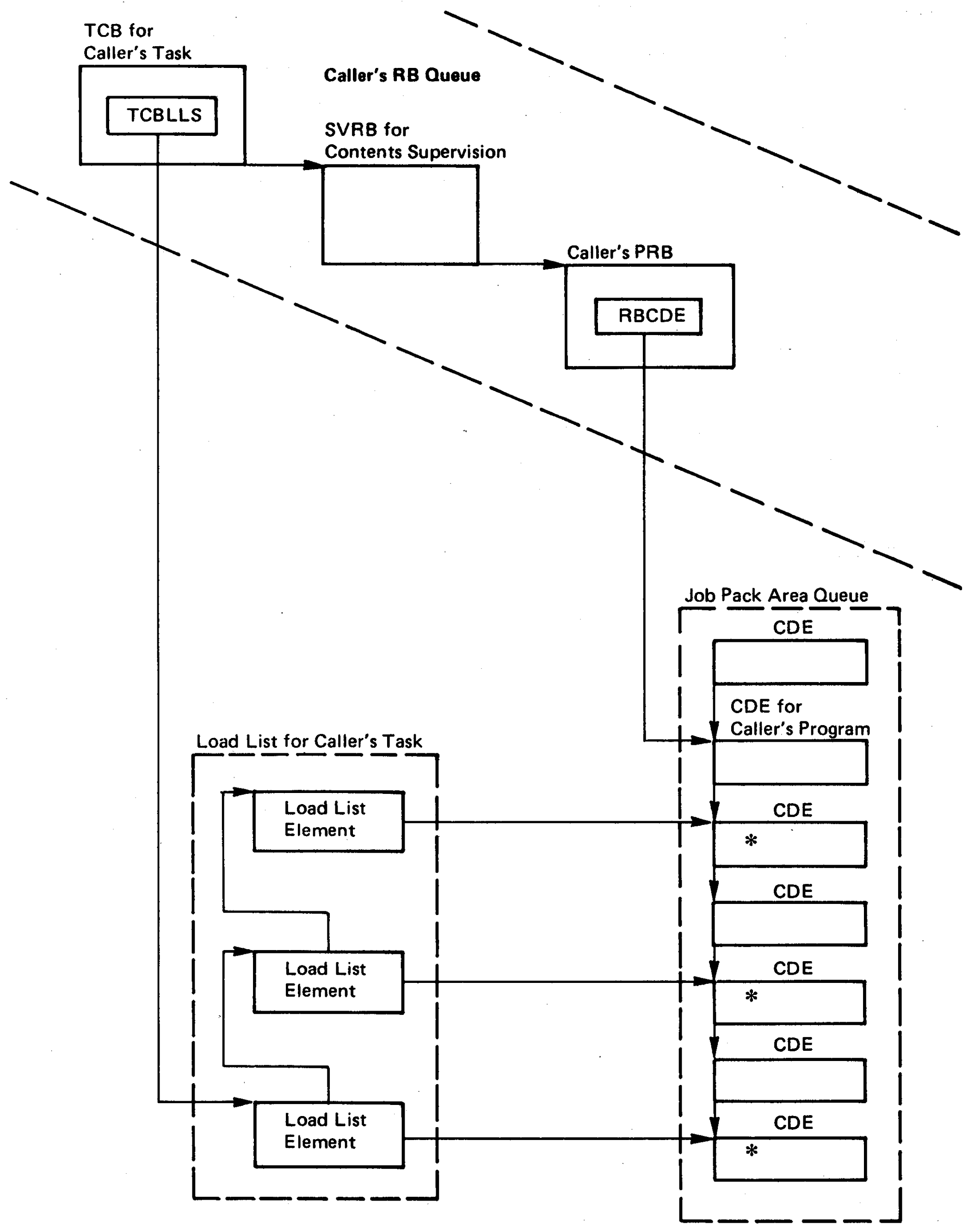
The JPA: CDEs represent modules in the JPA. Each CDE contains:

- The name of the module it represents.
- A pointer to the module's entry point.
- A use count that represents the total number of successful requests for a module by ATTACH, LINK, LOAD, and XCTL macro instructions. (The maximum use count is 32,757.)

If a caller has specified an alias entry point within a called module, there are two CDEs for the module. The major CDE contains the entry-point name; a minor CDE contains the alias entry-point name.

The Job Pack Area Queue: The CDEs representing a user's modules in the JPA are chained together and are called the JPAQ (job pack area queue). The JPAQ is in the LSQA assigned to a region. Each job step in the system has its own JPAQ. The beginning of the JPAQ is pointed to by the TCBJPQ field in the job-step TCB.

The Load List: Each time the LOAD service allocates a module to a requester, the use count in the CDE is increased. Also, an LLE (load list element) is created if one does not exist, and its responsibility count (LLECOUNT) is increased. The LLEs for each task in the job step are chained together to form the load list, which is the first queue the LOAD routine searches. Figure 2-41 shows the control blocks for modules in the JPA, including LLEs.



Legend:

— — — — Delineates queue

———► Pointer

* CDE for module loaded for caller's task

Figure 2-41. Control Blocks For Modules in the JPA

The need for a responsibility count in the LLE separate from the use count in the CDE is not readily apparent. Each time the LOAD service successfully allocates a module, the requesting routine may issue a DELETE macro when it no longer needs the module. The DELETE routine decreases the use and responsibility counts, and frees the module and its storage ares if they are both 0, meaning that there are no more outstanding requests.

LPA Storage Areas

The LPA is an area in virtual storage containing selected reenterable and serially reusable routines that are loaded at IPL time and can be used concurrently by all tasks in the system. Five LPA storage areas are defined:

- Pageable LPA
- LPA Directory
- Modified LPA
- LPA Queue
- Fixed LPA

Pageable LPA: An area residing in virtual storage below the SQA (system queue area) and above the CSA (common service area). The PLPA contains:

- Type 3 and 4 SVCs
- Access methods and other read-only system programs
- Any reenterable read-only user programs (selected by the installation) that can be shared by system users

LPA Directory: The LPA directory is a record of every program in the PLPA. The directory is created during nucleus initialization and consists of LPA directory entries (LPDEs) for each entry point in the PLPA modules. LPDEs for major entry points contain a CDE and a compressed extent list; LPDEs for alias entry points contain the name of a related major entry instead of a compressed extent list.

Modified LPA: The modified LPA is optionally specified via the "MLPA=" parameter and contains modules (from SYS1.SVCLIB, SYS1.LPALIB, and/or SYS1.LINKLIB) that are to be temporarily included in the PLPA as additions to or replacements for existing modules. The modified LPA must be specified at each IPL if it is to be used.

LPA Queue: The LPA queue is a record of all fixed, MLPA, and currently active PLPA modules. Entries in the LPA queue are chained contents directory entries (CDEs), one per entry point. When an LPA module is no longer needed (use count in CDE = 0), the control blocks that represent it in the LPA are removed. Currently active PLPA modules are still represented by LPDEs on the LPA directory.

Fixed LPA: The fixed LPA is an optional extension of the link pack area and can be defined to enhance system performance or to satisfy time dependencies of modules. If a fixed LPA is present, it is searched before the pageable LPA. Fixed LPA modules are represented by CDEs on the LPA queue and are used in preference to identical paged copies of modules in the PLPA. The fixed LPA is set up during nucleus initialization and resides in nondynamic, nonpageable low storage where the fixed control program is mapped 1:1 with virtual storage.

Auxiliary Storage Libraries

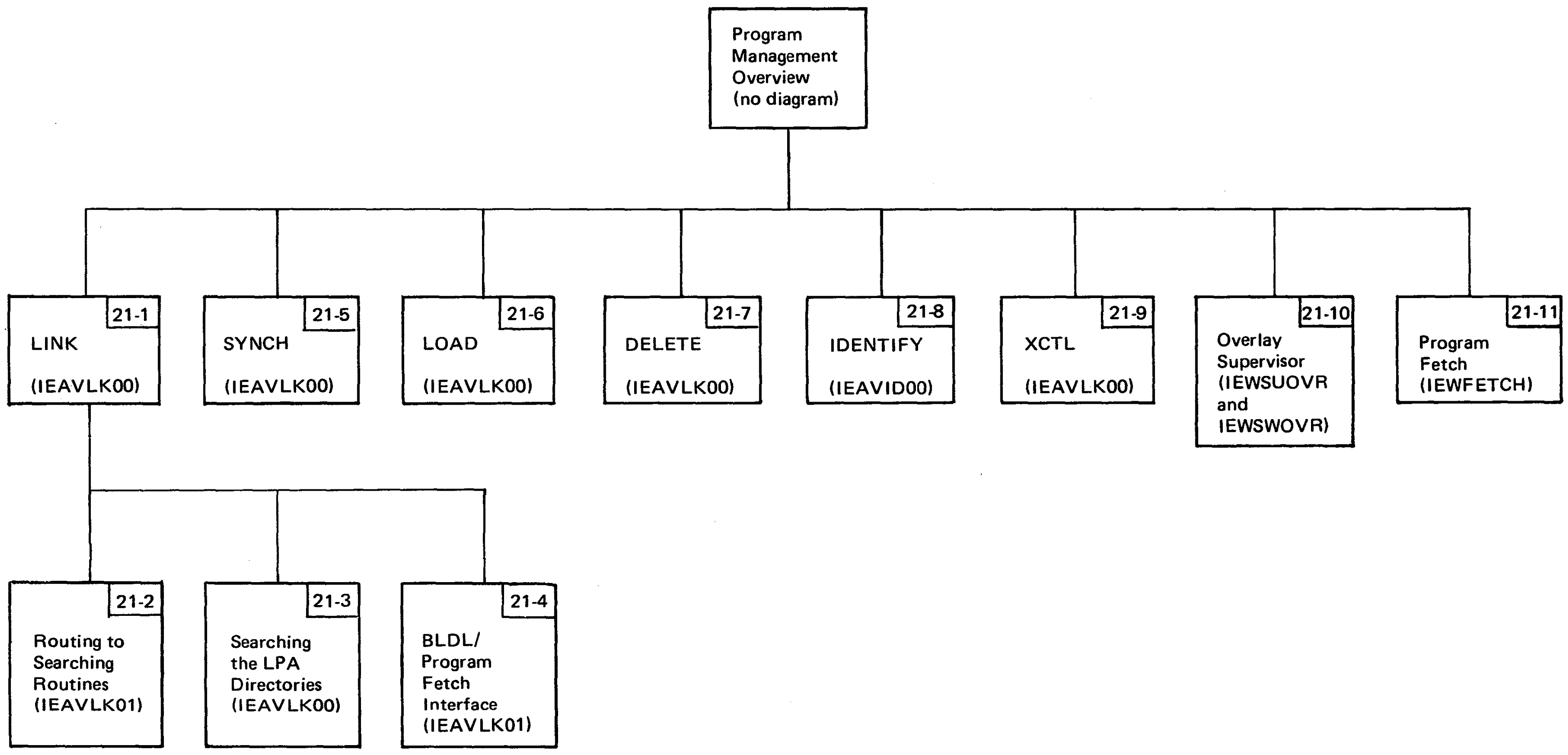
When program management services cannot find a requested module in virtual storage, BLDL searches the libraries on auxiliary storage. PDS DEs (partitioned data set directory elements) represent those modules on auxiliary storage.

Synchronizing Exit Routines

The SYNCH routine, after receiving control from the SVC IH (interruption handler), creates, initializes, and schedules for execution a PRB (program request block). This allows a supervisory program to take a synchronous exit to a problem program.

Fetching Modules into Storage

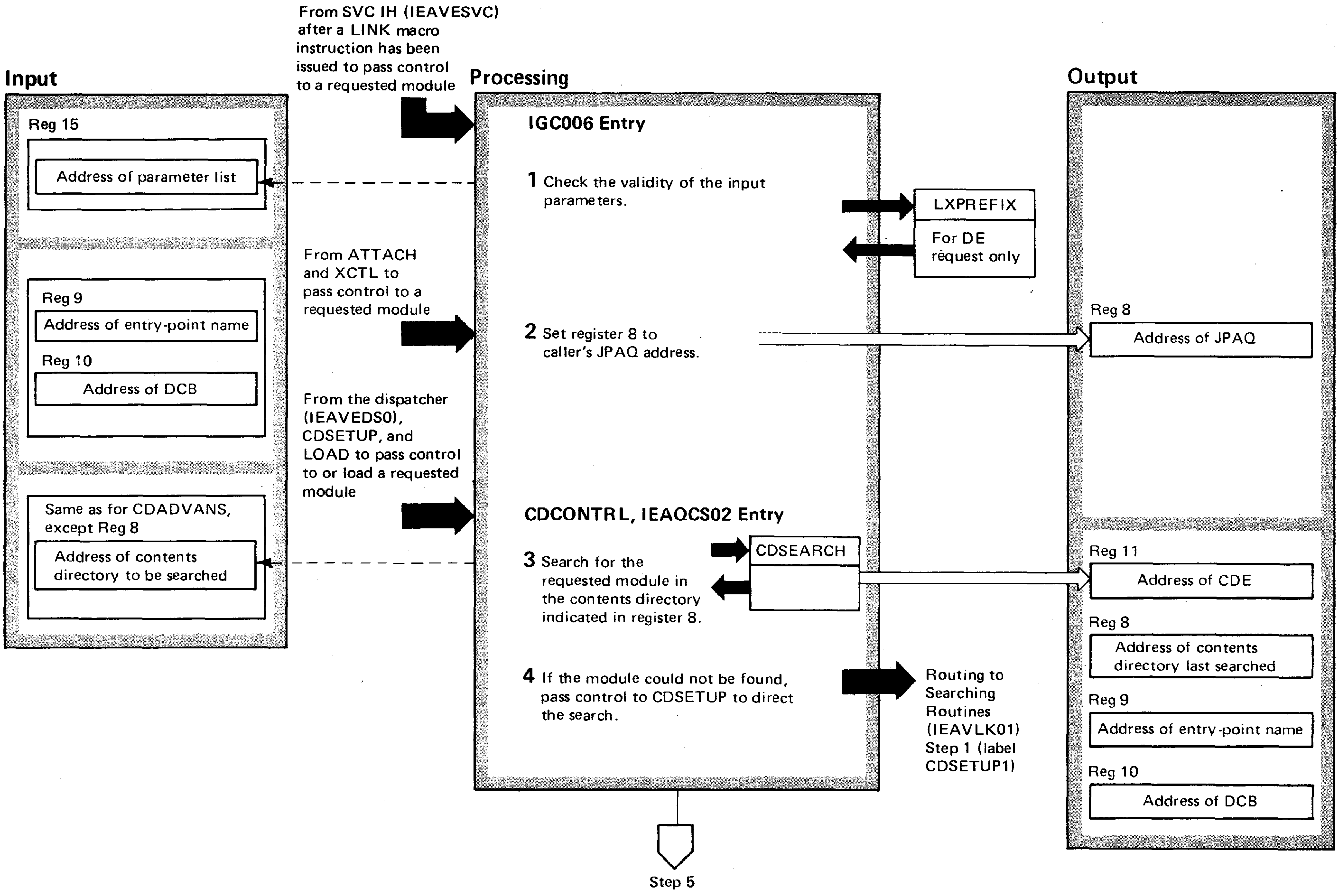
Program management services use Program Fetch to load requested modules into storage. If LINK, LOAD, or XCTL services do not locate the requested modules in virtual storage, these services will give control to Program Fetch to bring the module into virtual storage from auxiliary storage.



Program Management Overview (no diagram)
21-1
LINK (IEAVLK00)
21-5
SYNCH (IEAVLK00)
21-6
LOAD (IEAVLK00)
21-7
DELETE (IEAVLK00)
21-8
IDENTIFY (IEAVID00)
21-9
XCTL (IEAVLK00)
21-10
Overlay Supervisor (IEWSUOVR and IEWSWOVR)
21-11
Program Fetch (IEWFETCH)
21-2
Routing to Searching Routines (IEAVLK01)
21-3
Searching the LPA Directories (IEAVLK00)
21-4
BLDL/ Program Fetch Interface (IEAVLK01)

Figure 2-42. Program Management Visual Contents

Diagram 21-1. LINK Routine (IEAVLK00) (Part 1 of 6)


Input
Reg 15
Address of parameter list
Reg 9
Address of entry-point name
Reg 10
Address of DCB
Same as for CDADVANS, except Reg 8
Address of contents directory to be searched
Processing
IGC006 Entry
LXPREFIX
For DE request only
CDCONTRL, IEAQCS02 Entry
CDSEARCH
Step 5
Output
Reg 8
Address of JPAQ
Reg 11
Address of CDE
Reg 8
Address of contents directory last searched
Reg 9
Address of entry-point name
Reg 10
Address of DCB

From SVC IH (IEAVESVC) after a LINK macro instruction has been issued to pass control to a requested module

From ATTACH and XCTL to pass control to a requested module

From the dispatcher (IEAVEDSO), CDSETUP, and LOAD to pass control to or load a requested module

1 Check the validity of the input parameters.

2 Set register 8 to caller's JPAQ address.

3 Search for the requested module in the contents directory indicated in register 8.

4 If the module could not be found, pass control to CDSETUP to direct the search.

Routing to Searching Routines (IEAVLK01) Step 1 (label CDSETUP1)

Diagram 21-1. LINK Routine (IEAVLK00) (Part 2 of 6)

Extended Description	Module	Label
LINK creates the linkage to a specified load module for a user. LINK uses Program Fetch to bring into virtual storage those specified modules not already in virtual storage.		
1 LINK checks the input parameters for all users.	IEAVLK00	LXPREFIX
2 LINK places the address of the requester's JPAQ (job pack area queue) in register 8 to indicate to CDSEARCH which queue to search.		
3 The CDSEARCH subroutine searches for the requested module in the contents directory indicated in register 8.		IEAQCS02
4 If the module could not be found, CDCONTRL passes control to CDSETUP1 to direct the search.		CDSETUP1

Diagram 21-1. LINK Routine (IEAVLK00) (Part 3 of 6)

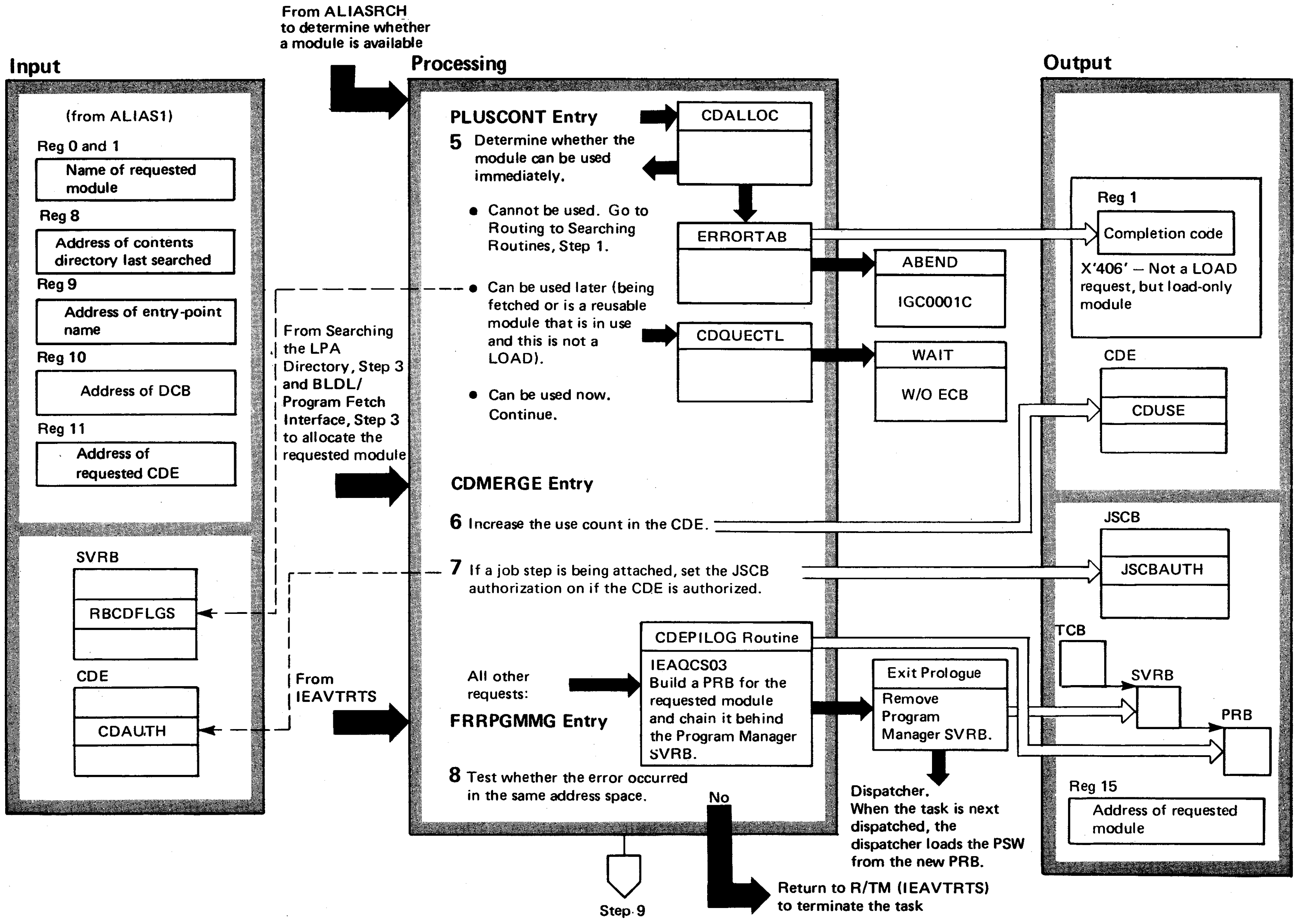

From ALIASRCH to determine whether a module is available
Input
Processing
Output
(from ALIAS1)
Reg 0 and 1
Name of requested module
Reg 8
Address of contents directory last searched
Reg 9
Address of entry-point name
Reg 10
Address of DCB
Reg 11
Address of requested CDE
SVRB
RBCDFLGS
CDE
CDAUTH
From Searching the LPA Directory, Step 3 and BLDL/ Program Fetch Interface, Step 3 to allocate the requested module
From IEAVTRTS
PLUSCONT Entry
5 Determine whether the module can be used immediately.
• Cannot be used. Go to Routing to Searching Routines, Step 1.
• Can be used later (being fetched or is a reusable module that is in use and this is not a LOAD).
• Can be used now. Continue.
CDALLOC
ERRORTAB
CDQUECTL
ABEND
IGC0001C
WAIT
W/O ECB
CDMERGE Entry
6 Increase the use count in the CDE.
7 If a job step is being attached, set the JSCB authorization on if the CDE is authorized.
All other requests:
FRRPGMMG Entry
CDEPILOG Routine
IEAQCS03
Build a PRB for the requested module and chain it behind the Program Manager SVRB.
Exit Prologue
Remove Program Manager SVRB.
8 Test whether the error occurred in the same address space.
No
Step 9
Dispatcher.
When the task is next dispatched, the dispatcher loads the PSW from the new PRB.
Return to R/TM (IEAVTRTS) to terminate the task
Reg 1
Completion code
X'406' – Not a LOAD request, but load-only module
CDE
CDUSE
JSCB
JSCBAUTH
TCB
SVRB
PRB
Reg 15
Address of requested module

Diagram 21-1. LINK Routine (IEAVLK00) (Part 4 of 6)

Extended Description	Module	Label
5 The CDALLOC subroutine of LINK considers three conditions to determine if a module can be used immediately:		CDALLOC
• Cannot be used.		
• Can be used later.		
• Can be used immediately.		
When modules cannot be used, control goes to "Routing to Search Routines (IEAVLK01)" to begin searching for the requested module.		
When the module can be used later, CDQUECTL queues the requests to be processed later, and passes control to the dispatcher (IEAVEDS0).		CDQUECTL
Processing continues when the module can be used immediately.		
6 LINK increases the use count in the CDE (contents directory element) to reflect that the requested module can be processed.		CDEMERGE
7 LINK sets the JSCBAUTH field of the JSCB to indicate authorization if the CDE is authorized.		CDEMERGE
Error Processing		
Error processing is the same for LINK, LOAD, ATTACH, and XCTL.		
8 FRRPGMMG determines whether the error occurred in the same address space as that of the routine currently executing.	IEAVLK03	FRRPGMMG
If not, R/TM (Recovery/Termination Management) will continue with termination.		

Diagram 21-1. LINK Routine (IEAVLK00) (Part 5 of 6)

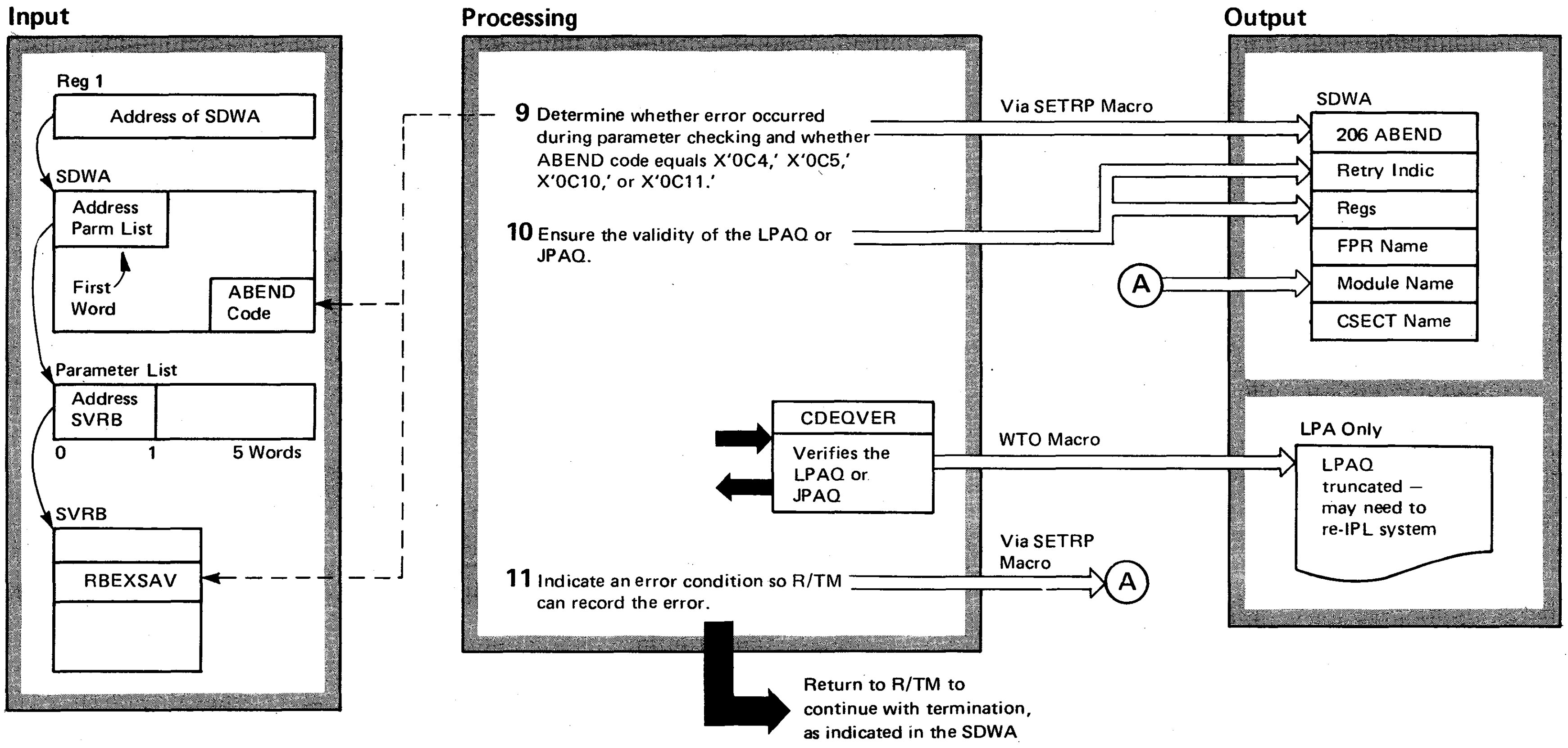
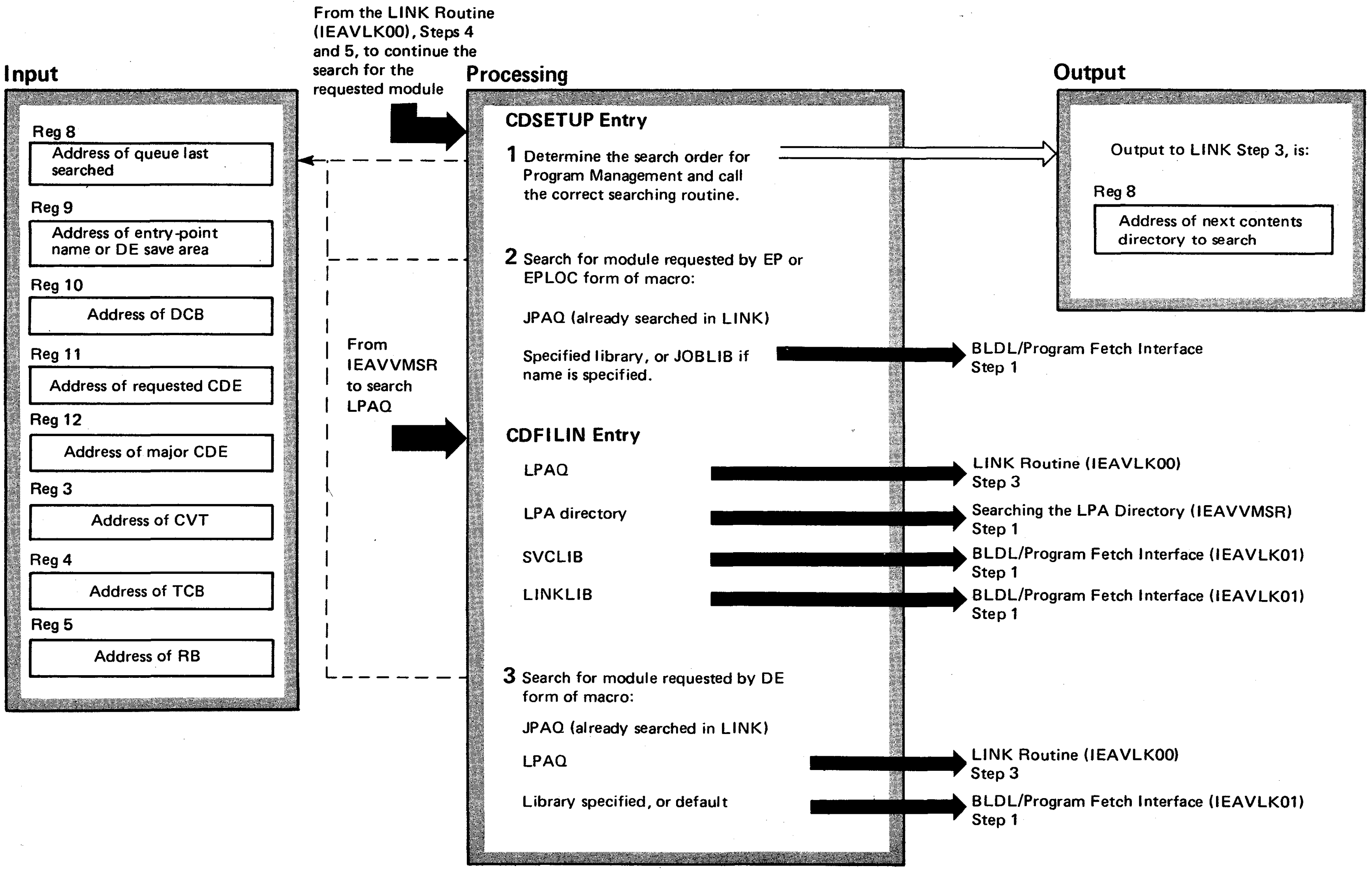

Input
Reg 1
Address of SDWA
SDWA
Address Parm List
First Word
ABEND Code
Parameter List
Address SVRB
0
1
5 Words
SVRB
RBEXSAV
Processing
9 Determine whether error occurred during parameter checking and whether ABEND code equals X'0C4,' X'0C5,' X'0C10,' or X'0C11.'
10 Ensure the validity of the LPAQ or JPAQ.
CDEQVER
Verifies the LPAQ or JPAQ
11 Indicate an error condition so R/TM can record the error.
Return to R/TM to continue with termination, as indicated in the SDWA
Via SETRP Macro
WTO Macro
Via SETRP Macro
A
Output
SDWA
206 ABEND
Retry Indic
Regs
FPR Name
Module Name
CSECT Name
LPA Only
LPAQ truncated — may need to re-IPL system

Diagram 21-1. LINK Routine (IEAVLK00) (Part 6 of 6)

Extended Description	Module	Label
9 Invalid input data should have a 206 ABEND code. FRRPGMMG checks the ABEND code in the SDWA (system diagnostic work area), and changes X'0C4,' X'0C5,' X'0C10,' or X'0C11.'	IEAVLK03	
10 The CDEQVER subroutine ensures the validity of the LPAQ or JPAQ. CDEQVER issues an error message to the operator if necessary.	IEAVLK03	NXTTST
11 FRRPGMMG indicates an error condition so R/TM can record the error. Control returns to R/TM to continue with the termination.	IEAVLK03	PERC

Diagram 21-2. Routing to Searching Routines (IEAVLK01) (Part 1 of 2)



From the LINK Routine (IEAVLK00), Steps 4 and 5, to continue the search for the requested module

Input

Reg 8
Address of queue last searched

Reg 9
Address of entry-point name or DE save area

Reg 10
Address of DCB

Reg 11
Address of requested CDE

Reg 12
Address of major CDE

Reg 3
Address of CVT

Reg 4
Address of TCB

Reg 5
Address of RB

Processing

CDSETUP Entry

1 Determine the search order for Program Management and call the correct searching routine.

2 Search for module requested by EP or EPLOC form of macro:

JPAQ (already searched in LINK)

Specified library, or JOBLIB if name is specified. → BLDL/Program Fetch Interface Step 1

From IEAVVMSR to search LPAQ

CDFILIN Entry

LPAQ → LINK Routine (IEAVLK00) Step 3

LPA directory → Searching the LPA Directory (IEAVVMSR) Step 1

SVCLIB → BLDL/Program Fetch Interface (IEAVLK01) Step 1

LINKLIB → BLDL/Program Fetch Interface (IEAVLK01) Step 1

3 Search for module requested by DE form of macro:

JPAQ (already searched in LINK)

LPAQ → LINK Routine (IEAVLK00) Step 3

Library specified, or default → BLDL/Program Fetch Interface (IEAVLK01) Step 1

Output

Output to LINK Step 3, is:

Reg 8
Address of next contents directory to search

Diagram 21-2. Routing to Searching Routines (IEAVLK01) (Part 2 of 2)

Extended Description	Module	Label
The CDSETUP subroutine determines the search order and routes control to the appropriate subroutines, based on the input parameters written in the macro request, for a requested module. CDSETUP follows the search order described in steps 2 and 3 in the order indicated.		
1 CDSETUP determines the search order for Program Management and calls the correct search routine, either CDSEARCH (for queue search) or IEAVVMSR (for directory search).	IEAVLK01	CDSETUP
2 The CDSETUP subroutine searches for the module requested by the EP or EPLOC form of the macro in the following manner:		
a. The CDSEARCH subroutine searches the contents directory entries in the JPA (job pack area) for the requested module.	IEAVLK00	
b. If the requester issued the DCB operand with the macro request, CDSETUP searches the specified library. If the requester did not issue the DCB operand, the CDSETUP routine searches the jobstep TCB's job library.	IEAVLK01	
c. CDFILIN searches the contents directory entries for load modules contained in the active link pack area to find an entry containing the specified entry point name.	IEAVLK01 IEAVLK00	
d. CHKLPDES searches the pageable link pack area.		
e. CHKLPDES gives control to the BLDL/Program Fetch interface to search SVCLIB (if DCB specified for SVCLIB).		
f. CHKLPDES searches the link library.		

Extended Description	Module	Label
3 The CDSEARCH subroutine searches for the module requested by the DE form of the macro in the following manner:		
a. It searches the contents directory entries for load modules contained in the job pack area.	IEAVLK00	
b. CDSEARCH searches the contents directory entries if the specified directory entry is for a load module contained in the link library.		
c. If the requester issued the DCB operand with the request, PGMFETCH fetches the specified load module. If the register did not issue the DCB operand, BUILDEL searches either the job library, link library, or task library, according a byte (the 'z-byte') in the PDS directory entry.	IEAVLK01	

Diagram 21-3. Searching the LPA Directory (IEAVLK00) (Part 1 of 2)

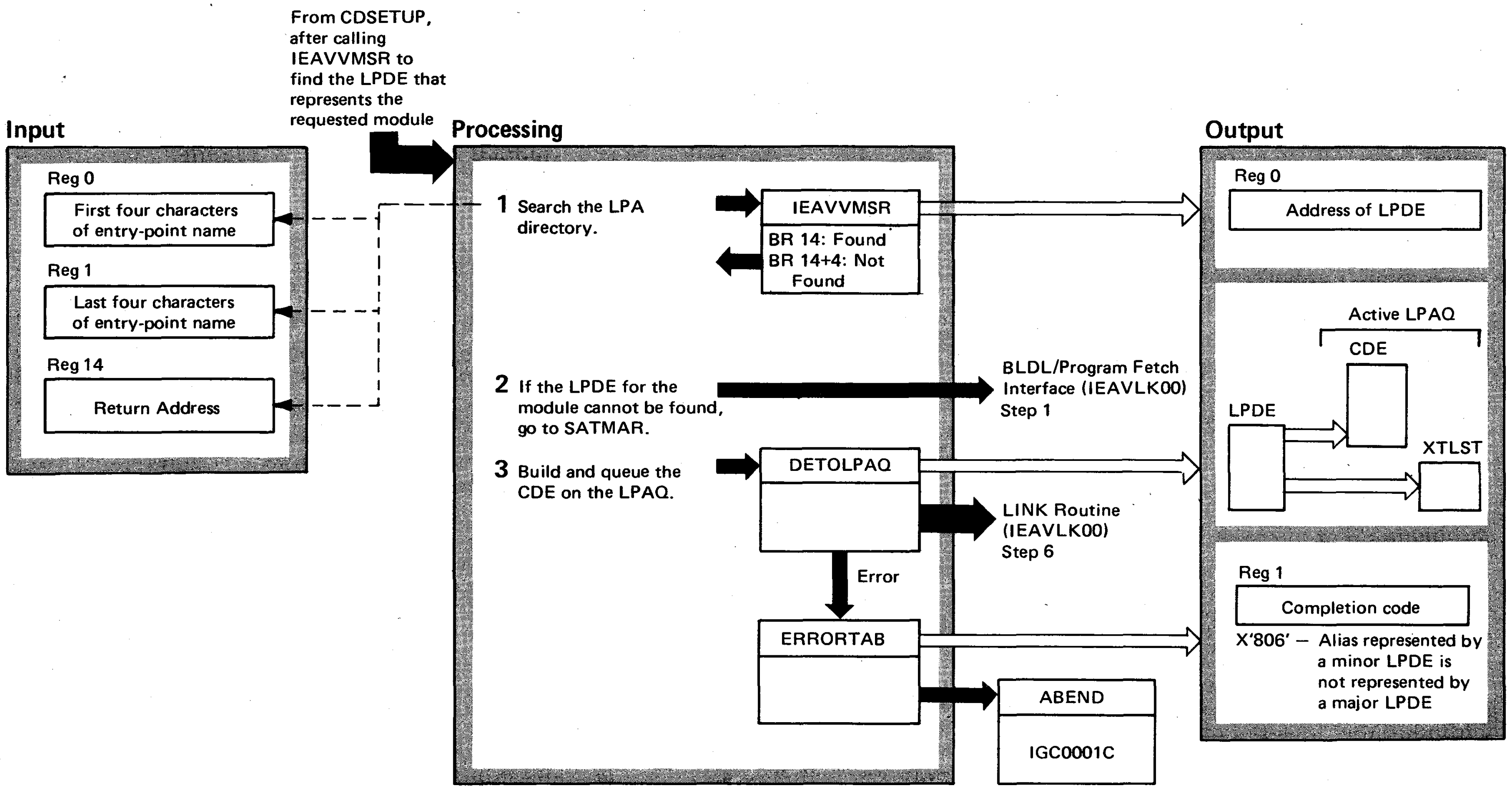
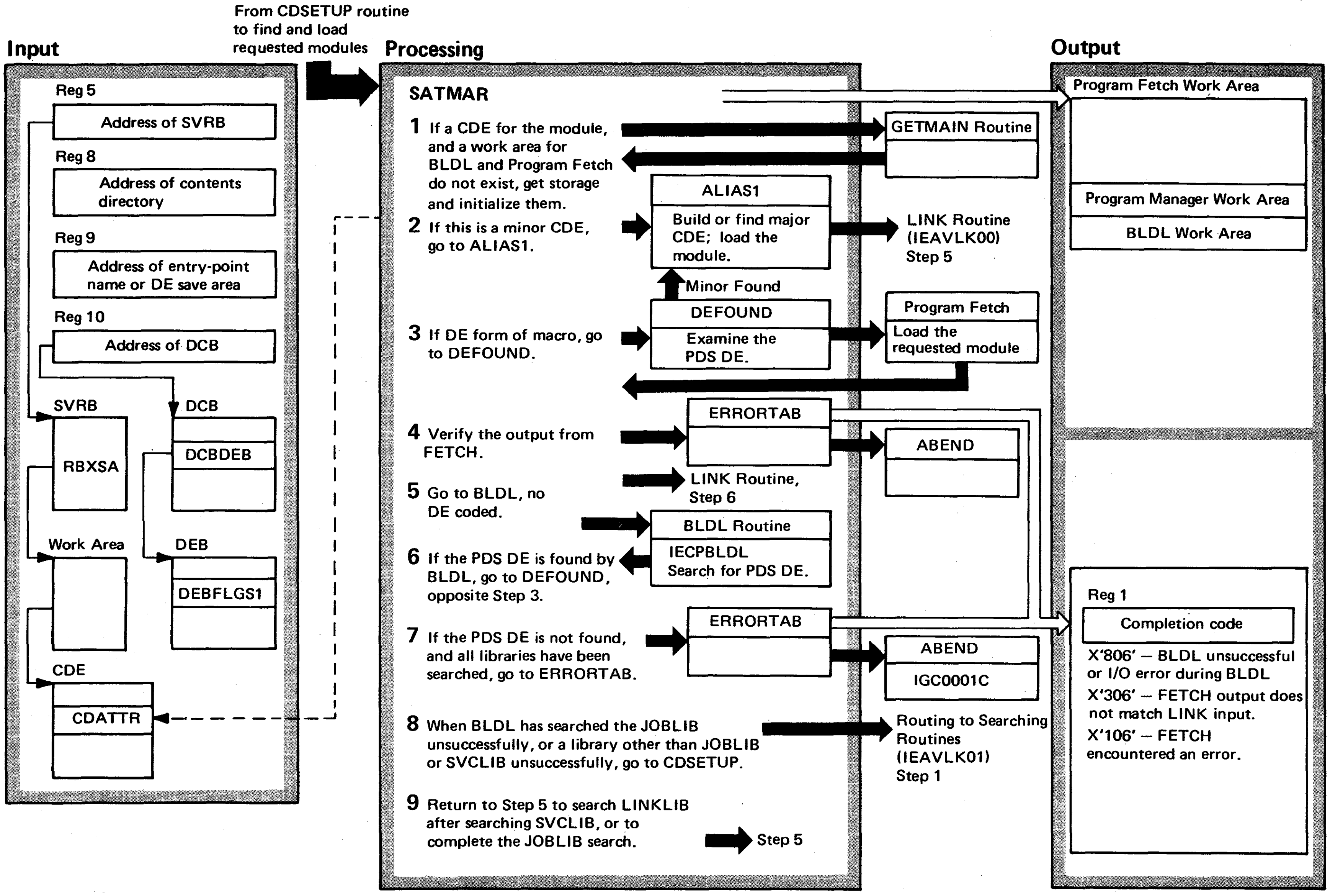

From CDSETUP, after calling IEAVVMSR to find the LPDE that represents the requested module
Input
Reg 0
First four characters of entry-point name
Reg 1
Last four characters of entry-point name
Reg 14
Return Address
Processing
1 Search the LPA directory.
IEAVVMSR
BR 14: Found
BR 14+4: Not Found
2 If the LPDE for the module cannot be found, go to SATMAR.
BLDL/Program Fetch Interface (IEAVLK00) Step 1
3 Build and queue the CDE on the LPAQ.
DETOLPAQ
LINK Routine (IEAVLK00) Step 6
Error
ERRORTAB
ABEND
IGC0001C
Output
Reg 0
Address of LPDE
Active LPAQ
CDE
LPDE
XTLST
Reg 1
Completion code
X'806' – Alias represented by a minor LPDE is not represented by a major LPDE

Diagram 21-3. Searching the LPA Directory (IEAVLK00) (Part 2 of 2)

Extended Description	Module	Label
The IEAVVMSR subroutine searches the LPA directory to attempt to locate the specified module.		
1 IEAVVMSR computes an index factor to search the LPA directory. After the computation, IEAVVMSR has the address of an LPDE (link pack directory entry) in the LPA directory, and determines whether the name in the LPDE, or the name in another LPDE in the chain matches the requested name.	IEAVLK00	IEAVVMSR
2 Control next passes to the SATMAR subroutine if CDSETUP cannot find the module in the LPD.	IEAVLK01	CDSETUP
3 CDSETUP passes control to DETOLPAQ to build and initialize a CDE (contents directory element).		CDSETUP

Diagram 21-4. BLDL/Program Fetch Interface (IEAVLK01) (Part 1 of 2)



From CDSETUP routine to find and load requested modules
Input
Reg 5
Address of SVRB
Reg 8
Address of contents directory
Reg 9
Address of entry-point name or DE save area
Reg 10
Address of DCB
SVRB
RBXSA
DCB
DCBDEB
Work Area
DEB
DEBFLGS1
CDE
CDATTR
Processing
SATMAR
GETMAIN Routine
ALIAS1
Build or find major CDE; load the module.
LINK Routine (IEAVLK00) Step 5
Minor Found
DEFOUND
Examine the PDS DE.
Program Fetch
Load the requested module
ERRORTAB
ABEND
LINK Routine, Step 6
BLDL Routine
IECPBLDL Search for PDS DE.
ERRORTAB
ABEND
IGC0001C
Routing to Searching Routines (IEAVLK01) Step 1
Step 5
Output
Program Fetch Work Area
Program Manager Work Area
BLDL Work Area
Reg 1
Completion code
X'806' – BLDL unsuccessful or I/O error during BLDL
X'306' – FETCH output does not match LINK input.
X'106' – FETCH encountered an error.

1 If a CDE for the module, and a work area for BLDL and Program Fetch do not exist, get storage and initialize them.

2 If this is a minor CDE, go to ALIAS1.

3 If DE form of macro, go to DEFOUND.

4 Verify the output from FETCH.

5 Go to BLDL, no DE coded.

6 If the PDS DE is found by BLDL, go to DEFOUND, opposite Step 3.

7 If the PDS DE is not found, and all libraries have been searched, go to ERRORTAB.

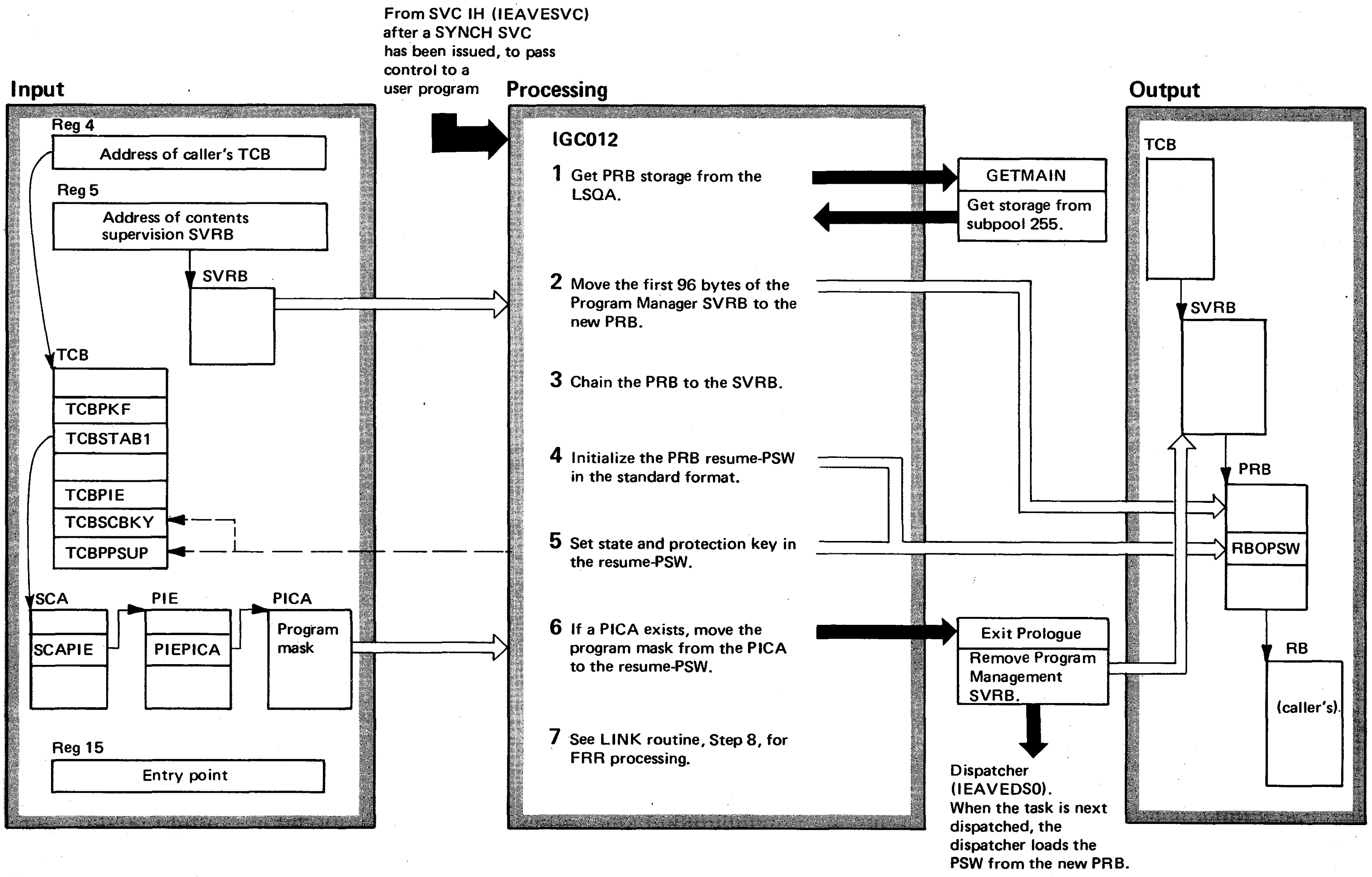
8 When BLDL has searched the JOBLIB unsuccessfully, or a library other than JOBLIB or SVCLIB unsuccessfully, go to CDSETUP.

9 Return to Step 5 to search LINKLIB after searching SVCLIB, or to complete the JOBLIB search.

Diagram 21-4. BLDL/Program Fetch Interface (IEAVLK01) (Part 2 of 2)

Extended Description	Module	Label
The BLDL/Program Fetch Interface constructs any necessary control blocks needed by Program Fetch to perform the fetch operation.		
1 The SATMAR subroutine creates a CDE (contents directory element) and queues it to the job-step (TCBJPQ) prior to BLDL and Program Fetch processing. This ensures that subsequent requests for the same module will be deferred during BLDL or Program Fetch processing.	IEAVLK01	SATMAR
2 SATMAR passes control to ALIAS1 to build or find the major CDE if the work area points to a minor CDE.		SATMAR
3 Control goes to DEFOUND to examine the PDS DE (partitioned data set directory element) when the caller codes the DE form of the macro, or on return from a successful BLDL.		SATMAR
4 The output from Fetch must match the input to Fetch.		PGMFETCH
5 BUILDEL, a Program Management routine, calls the BLDL routine to find the PDS DE for the requested module.		BUILDEL
6 BUILDEL passes control to DEFOUND if BLDL finds the PDS DE.		BUILDEL
7 ERRORTAB indicates the error condition if no PDS DE can be found on any library.		ERRORTAB
8 SATMAR gives control to CDSETUP to search for the requested module.		SATMAR
9 Control goes to Step 5 to search the LINKLIB after searching the SVCLIB (if necessary), or to complete the JOBLIB search.		

Diagram 21-5. SYNCH Routine (IEAVLK00) (Part 1 of 2)

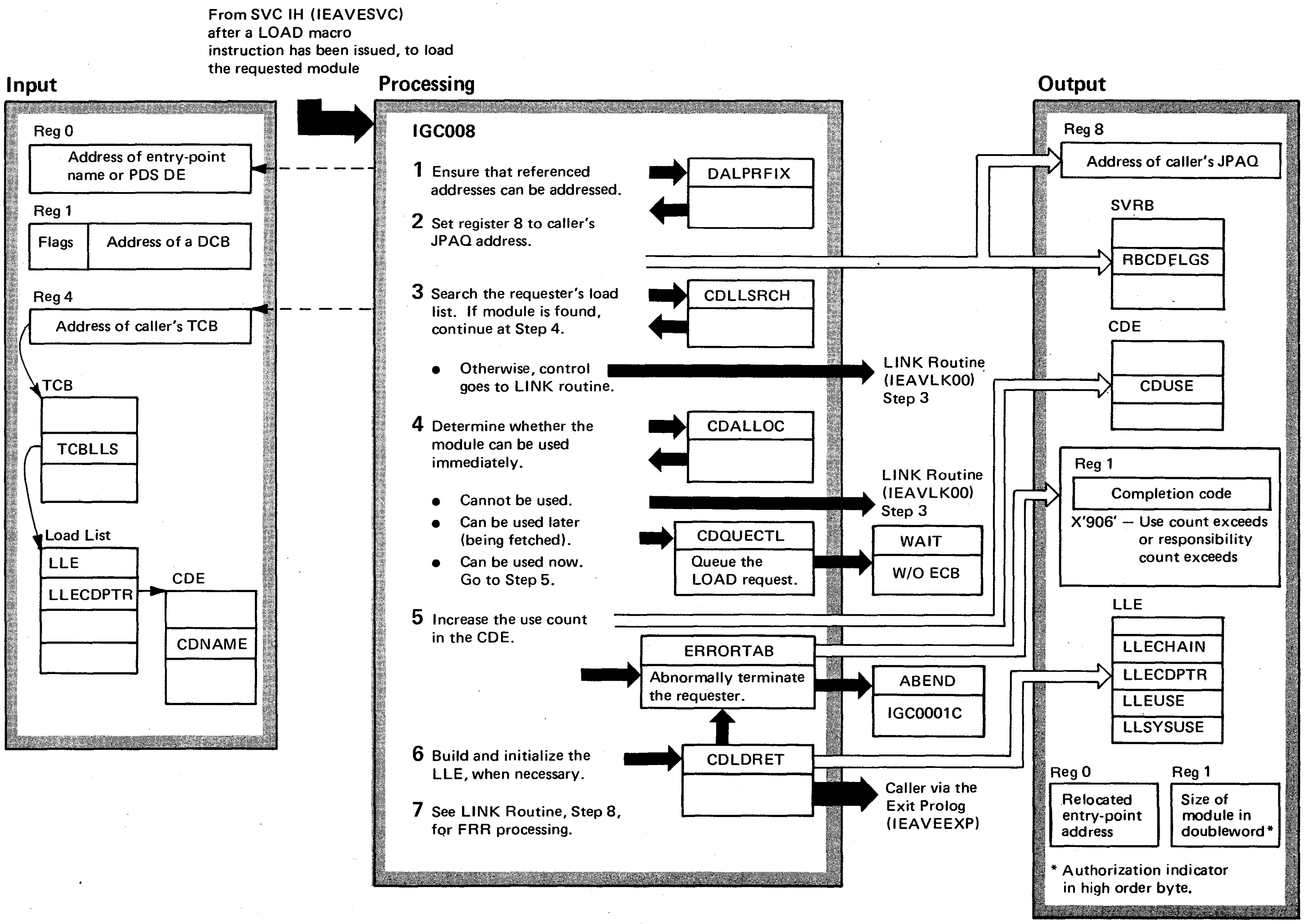


From SVC IH (IEAVESVC) after a SYNCH SVC has been issued, to pass control to a user program
Input
Reg 4
Address of caller's TCB
Reg 5
Address of contents supervision SVRB
SVRB
TCB
TCBPKF
TCBSTAB1
TCBPIE
TCBSCBKY
TCBPPSUP
SCA
SCAPIE
PIE
PIEPICA
PICA
Program mask
Reg 15
Entry point
Processing
IGC012
1 Get PRB storage from the LSQA.
2 Move the first 96 bytes of the Program Manager SVRB to the new PRB.
3 Chain the PRB to the SVRB.
4 Initialize the PRB resume-PSW in the standard format.
5 Set state and protection key in the resume-PSW.
6 If a PICA exists, move the program mask from the PICA to the resume-PSW.
7 See LINK routine, Step 8, for FRR processing.
GETMAIN
Get storage from subpool 255.
Exit Prologue
Remove Program Management SVRB.
Dispatcher (IEAVEDSO). When the task is next dispatched, the dispatcher loads the PSW from the new PRB.
Output
TCB
SVRB
PRB
RBOPSW
RB
(caller's)

Diagram 21-5. SYNCH Routine (IEAVLK00) (Part 2 of 2)

Extended Description	Module	Label
The SYNCH routine allows a supervisor routine to take a synchronous exit to a user program. SYNCH creates, initializes, and schedules for execution a PRB (program request block) that represents the synchronous exit request. Control returns from the user program to the supervisor routine that issued the SYNCH request.		
1 SYNCH issues GETMAIN for the storage for the PRB.	IEAVLK00	IGC012
2 SYNCH moves the first 96 bytes of the Program Management SVRB (supervisor request block) into the newly created PRB.		THRUX
3 SYNCH chains the PRB behind the SVRB.		THRUX
4 The standard format for the PSW (program status word), X'000D000000', will be modified by SYNCH to set the first byte to the proper program mask. The resume PSW in this step refers to the RBOPSW field in the PRB.		THRUX
5 SYNCH checks the TCBSYNCH field of the TCB for a 0 to determine whether the SYNCH request will enter a STAE exit routine. If TCBSYNCH contains a 0, the SYNCH request will not enter a STAE exit. Additionally, the requested program will execute with the protection key indicated in the caller's TCBPKF field, and in problem state. If TCBSYNCH equals 1, the SYNCH request will enter a STAE exit. SYNCH sets the RBOPSW to indicate problem state when the value in the TCBPPSUP field of the TCB equals 1, or to supervisor state if the value equals 0. SYNCH then sets the RBOPSW protection key to equal the value in the TCBSCBKY field of the TCB.		SYNCTEST
6 SYNCH moves the program mask from any existing PICA to the resume PSW (in RBOPSW).		PICAMASK

Diagram 21-6. LOAD Routine (IEAVLK00) (Part 1 of 2)


From SVC IH (IEAVESVC) after a LOAD macro instruction has been issued, to load the requested module
Input
Reg 0
Address of entry-point name or PDS DE
Reg 1
Flags
Address of a DCB
Reg 4
Address of caller's TCB
TCB
TCBLLS
Load List
LLE
LLECDPTR
CDE
CDNAME
Processing
IGC008
DALPRFIX
CDLLSRCH
LINK Routine (IEAVLK00) Step 3
CDALLOC
LINK Routine (IEAVLK00) Step 3
CDQUECTL
Queue the LOAD request.
WAIT
W/O ECB
ERRORTAB
Abnormally terminate the requester.
ABEND
IGC0001C
CDLDRET
Caller via the Exit Prolog (IEAVEEXP)
Output
Reg 8
Address of caller's JPAQ
SVRB
RBCDFLGS
CDE
CDUSE
Reg 1
Completion code
X'906' – Use count exceeds or responsibility count exceeds
LLE
LLECHAIN
LLECDPTR
LLEUSE
LLSYSUSE
Reg 0
Relocated entry-point address
Reg 1
Size of module in doubleword*
* Authorization indicator in high order byte.

1 Ensure that referenced addresses can be addressed.

2 Set register 8 to caller's JPAQ address.

3 Search the requester's load list. If module is found, continue at Step 4.

- Otherwise, control goes to LINK routine.

4 Determine whether the module can be used immediately.

- Cannot be used.
- Can be used later (being fetched).
- Can be used now. Go to Step 5.

5 Increase the use count in the CDE.

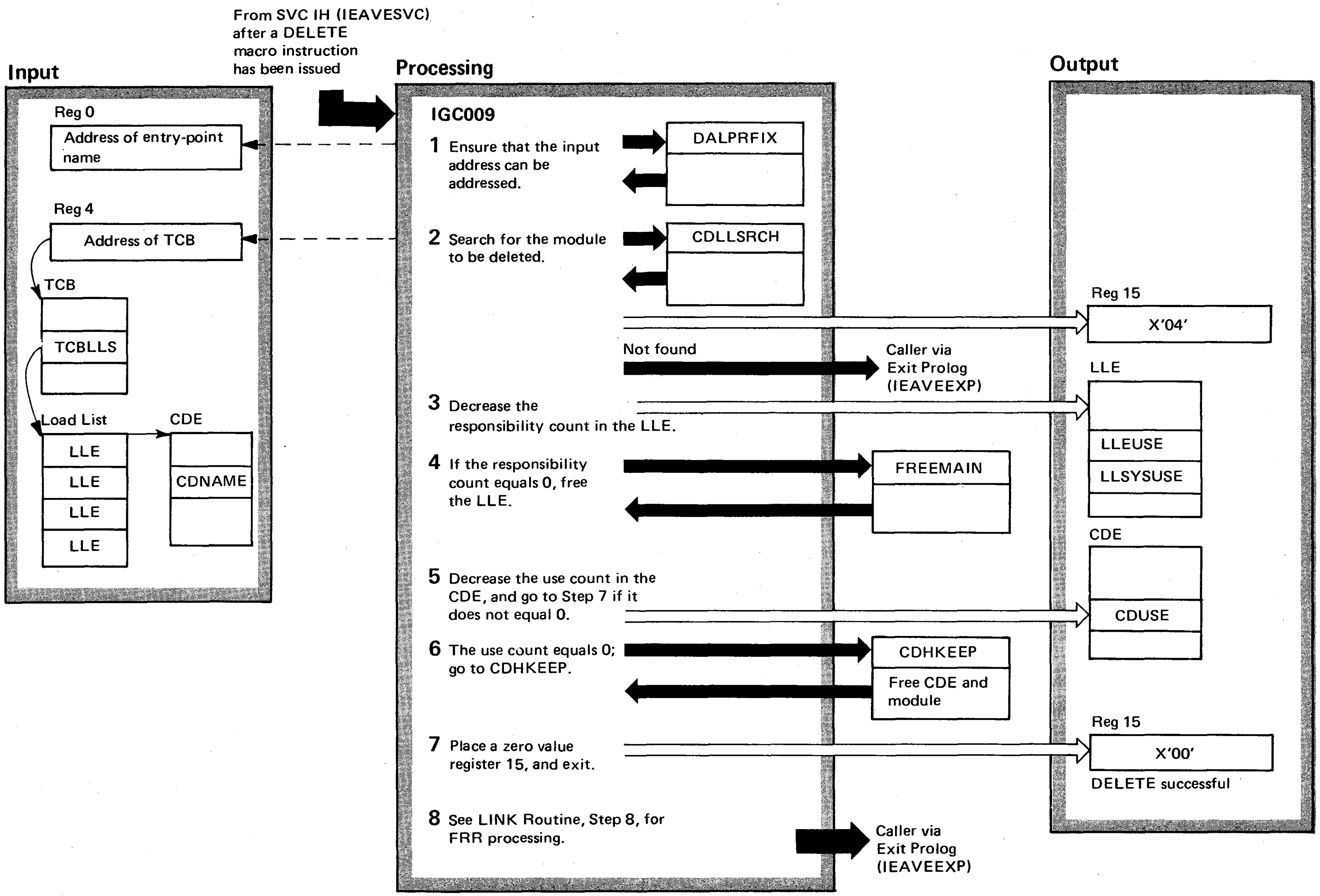
6 Build and initialize the LLE, when necessary.

7 See LINK Routine, Step 8, for FRR processing.

Diagram 21-6. LOAD Routine (IEAVLK00) (Part 2 of 2)

Extended Description	Module	Label
The LOAD Routine brings a module containing a specified entry point into virtual storage if no useable copies exist in storage.		
1 LOAD calls DALPRFIX to ensure that the input parameters are valid.	IEAVLK00	IGC008
2 LOAD places the address of the requesting JPAQ (job pack area queue) in register 8 in case LINK must continue to search for the module. Then, LOAD sets the lower order bit in RBCDFLGS equal to 1 to indicate a load request.		IGC008
3 LOAD gives control to CDLLSRCH to search for the requester's load list. If CDLLSRCH cannot find the load list, LOAD passes control to the LINK routine.		CDLLSRCH
4 The LOAD routine considers three conditions to determine module useability:		CDALLOC
• Cannot be used.		
• Can be used later.		
• Can be used immediately.		
When a module cannot be used, control goes to the LINK routine, which begins searching for the requested module.		
When the module can be used later, CDQUECTL queues the requests to be processed later, and issues a WAIT macro instruction.		CDQUECTL
Processing continues when the module can be used immediately.		
5 LOAD increases the use count in the CDUSE field of the CDE.		CDMOPUP
LOAD passes control to the ABEND routine to terminate the requester if the use count exceeds 32,767.		
6 CDLDRET gets storage for an LLE (load list element), if none already exists, and chains it to the caller's load list.		CDLDRET
CDLDRET increases the responsibility count in the LLE, and if the count exceeds 32,767, gives control to the ABEND routine to terminate the requester.		
CDLDRET also increases the system responsibility count in field LLSYSUSE for system requests.		

Diagram 21-7. DELETE Routine (IEAVLK00) (Part 1 of 2)



IGC009

1 Ensure that the input address can be addressed.

2 Search for the module to be deleted.

3 Decrease the responsibility count in the LLE.

4 If the responsibility count equals 0, free the LLE.

5 Decrease the use count in the CDE, and go to Step 7 if it does not equal 0.

6 The use count equals 0; go to CDHKEEP.

7 Place a zero value register 15, and exit.

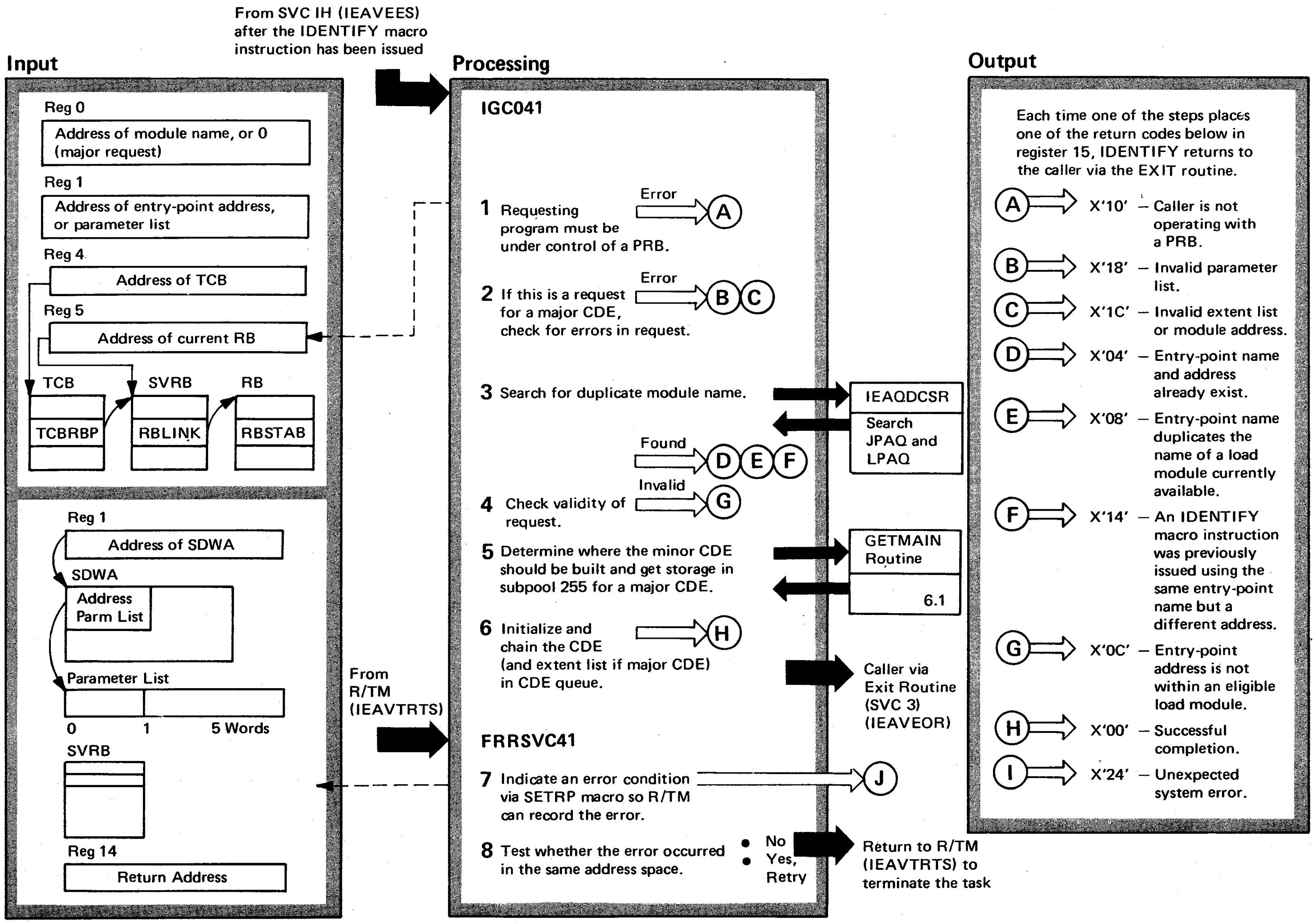
8 See LINK Routine, Step 8, for FRR processing.

Diagram 21-7. DELETE Routine (IEAVLK00) (Part 2 of 2)

Extended Description	Module	Label
The DELETE routine enables the requester who issued a LOAD request to remove those modules he brought into virtual storage. DELETE decreases the use count of the CDE (contents directory element) and the responsibility count of the LLE (load list element). DELETE then frees the LLE when the responsibility count reaches 0.		
1 DELETE calls DALPRFIX to ensure that the requested module's entry-point name can be addressed.	IEAVLK00	IGC009
2 DELETE passes control to CDLLSRCH to search for the requested module.		CDLLSRCH
3 The LLEUSE count increases by one for every LOAD request. DELETE decreases the responsibility count of the LLE by 1. DELETE also decreases the system responsibility count (LLSYSUSE) for requests from system routines.		
Note: The LLE responsibility count indicates the number of oustanding LOAD requests for the module.		

Extended Description	Module	Label
4 DELETE gives cohtrol to the FREEMAIN routine to free the storage occupied by the LLE if the LLEUSE count equals 0.		DELNORM
5 The CDE use count represents the total number of requests made by either ATTACH, LINK, XCTL, or LOAD macro instructions. The count increases each time one of these macros is successfully issued, and decreases each time a DELETE is successfully issued or the routine goes through exit.		MAJOR
		MAJOR
6 The CDUSE field contains the use count. The CDHKEEP routine frees the virtual storage occupied by the program, its extent list, and its major and minor CDEs when the use count reaches 0 (for JPQ modules).	IEAVLK02	CDHKEEP
7 DELETE passes the caller a return code of 0 to indicate completion of DELETE.		DELETXIT

Diagram 21-8. IDENTIFY Routine (IEAVID00) (Part 1 of 4)



IGC041

1 Requesting program must be under control of a PRB. — Error → A

2 If this is a request for a major CDE, check for errors in request. — Error → B, C

3 Search for duplicate module name. — Found → D, E, F

4 Check validity of request. — Invalid → G

5 Determine where the minor CDE should be built and get storage in subpool 255 for a major CDE.

6 Initialize and chain the CDE (and extent list if major CDE) in CDE queue. → H

FRRSVC41

7 Indicate an error condition via SETRP macro so R/TM can record the error. → J

8 Test whether the error occurred in the same address space.
- No
- Yes, Retry

Each time one of the steps places one of the return codes below in register 15, IDENTIFY returns to the caller via the EXIT routine.

A	X'10'	– Caller is not operating with a PRB.
B	X'18'	– Invalid parameter list.
C	X'1C'	– Invalid extent list or module address.
D	X'04'	– Entry-point name and address already exist.
E	X'08'	– Entry-point name duplicates the name of a load module currently available.
F	X'14'	– An IDENTIFY macro instruction was previously issued using the same entry-point name but a different address.
G	X'0C'	– Entry-point address is not within an eligible load module.
H	X'00'	– Successful completion.
I	X'24'	– Unexpected system error.

Diagram 21-8. IDENTIFY Routine (IEAVID00) (Part 2 of 4)

Extended Description	Module	Label
The IDENTIFY routine searches for and identifies a module's embedded entry-point name (a name not established by the linkage editor). IDENTIFY creates a CDE (contents directory entry to represent the embedded entry-point name.		
1 IDENTIFY passes an error code in register 15 if the caller is not operating with a PRB.	IEAVID00	YESPRB
2 A subroutine of IDENTIFY, MAJORCDE, builds a major CDE. MAJORCDE performs the same operations as IDENTIFY, which builds minor CDEs. Steps 4-7 show the operations for both IDENTIFY and MAJORCDE.		MAJORCDE
3 IDENTIFY (or MAJORCDE) passes control to the IEAQCDSR subroutine to search for a duplicate module name.		NOMIN
4 IDENTIFY passes an error code in register 15 if the caller issues an invalid request.		XLINST
5 IDENTIFY builds the major/minor CDEs in the JPAQ (job pack area queue) or LPAQ (link pack area queue), depending on the location of the major CDE, and the authorization of the caller.		GETCDE
MAJORCDE builds major CDEs in the LSQA.		NAMETEST
6 IDENTIFY chains the CDE in the CDE queue.		CDESETUP
MAJORCDE chains the CDE in the CDE queue.		NAMETEST
7 FRRSVC41 indicates an error condition so R/TM can record the error.		FRRSVC41
8 FRRSVC41 determines whether the error occurred in the same address space with the routine currently executing. FRRSVC41 will retry the routine if the error occurred in the same address space.		FRRSVC41
If the error occurred in a different address space, R/TM will continue with termination.		SVC41PRC

Diagram 21-8. IDENTIFY Routine (IEAVID00) (Part 3 of 4)

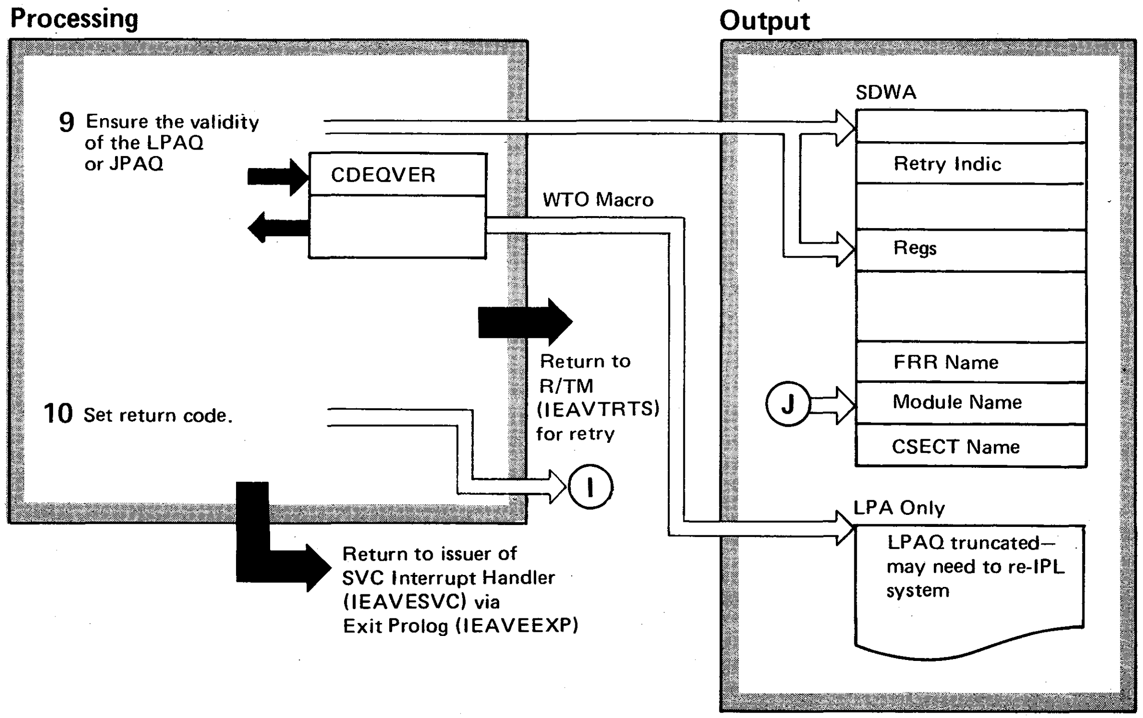

Processing
Output
9 Ensure the validity of the LPAQ or JPAQ
CDEQVER
WTO Macro
Return to R/TM (IEAVTRTS) for retry
10 Set return code.
I
Return to issuer of SVC Interrupt Handler (IEAVESVC) via Exit Prolog (IEAVEEXP)
SDWA
Retry Indic
Regs
FRR Name
J
Module Name
CSECT Name
LPA Only
LPAQ truncated— may need to re-IPL system

Diagram 21-8. IDENTIFY Routine (IEAVID00) (Part 4 of 4)

Extended Description	Module	Label
9 The CDEQVER subroutine ensures the validity of the LPAQ or JPAQ. CDEQVER issues an error message to the operator, if necessary.		FRRSVC41
FRRSVC41 saves registers 6 and 13 in the SDWA.		SVC41PRC
10 FRRSVC41 sets a return code of X'24' and returns to the caller.		SVC41RTY

Diagram 21-9. XCTL Routine (IEAVLK00) (Part 1 of 6)

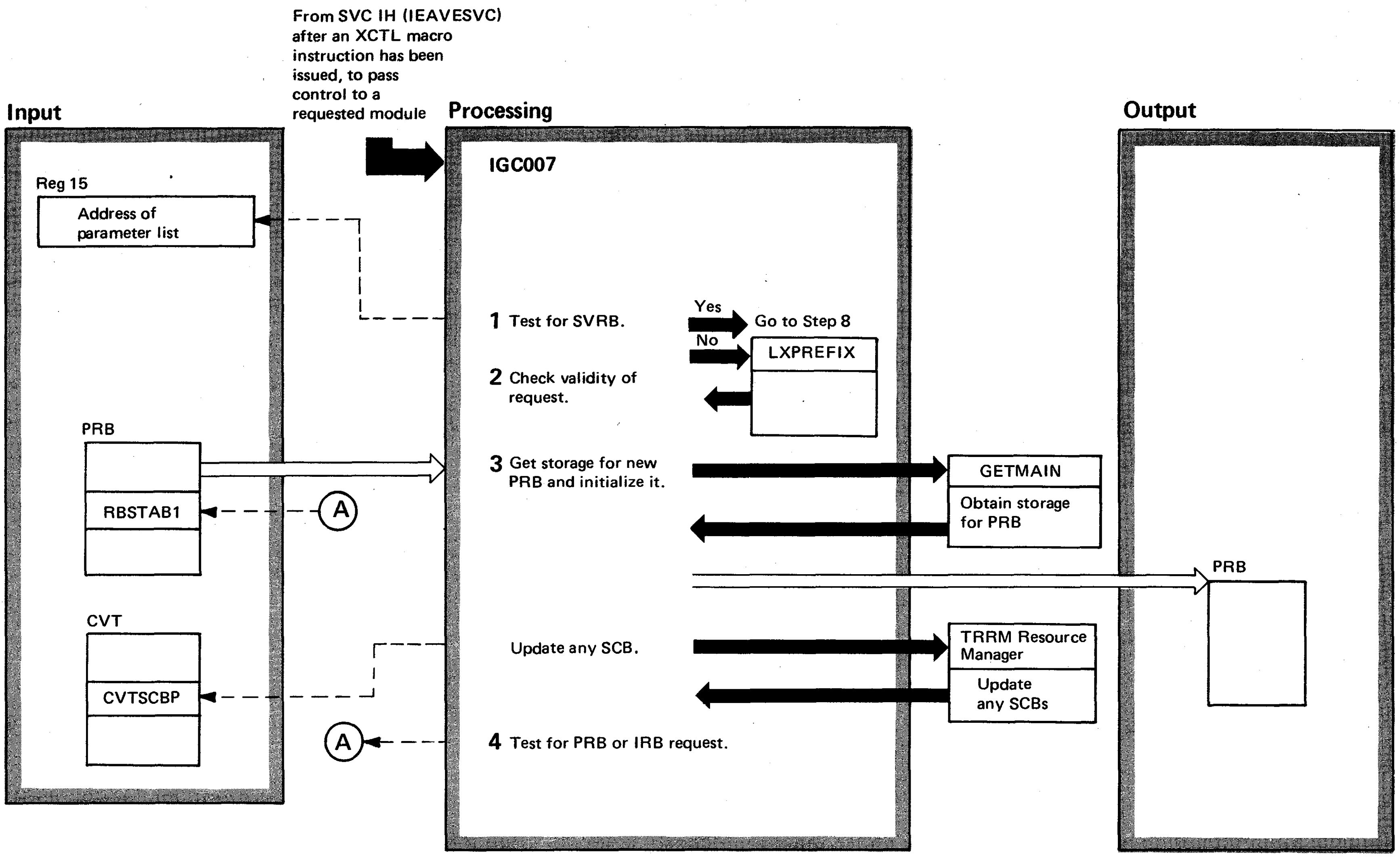

From SVC IH (IEAVESVC) after an XCTL macro instruction has been issued, to pass control to a requested module
Input
Reg 15
Address of parameter list
PRB
RBSTAB1
CVT
CVTSCBP
A
A
Processing
IGC007
1 Test for SVRB.
Yes
Go to Step 8
No
LXPREFIX
2 Check validity of request.
3 Get storage for new PRB and initialize it.
GETMAIN
Obtain storage for PRB
Update any SCB.
TRRM Resource Manager
Update any SCBs
4 Test for PRB or IRB request.
Output
PRB

Diagram 21-9. XCTL Routine (IEAVLK00) (Part 2 of 6)

Extended Description	Module	Label
The XCTL routine creates the linkage to a specified load module and ensures that the requester does not regain control after the specified load module has been executed. The specified load module executes with the same protection key and in the same state as the requester.		
The XCTL routine only performs the XCTL service for requesters represented by an SVRB (supervisor request block); it calls LINK to honor requests made by requesters operating with a PRB (program request block) or IRB (interruption request block).		
1 Control goes to step 8 to process SVRBs.	IEAVLK00	
2 The LXPREFIX subroutine checks the validity of the request.		LXPREFIX
3 XCTL passes control to the GETMAIN routine to obtain storage for a new PRB (program request block). XCTL initializes the new PRB with the information in the old PRB.		NOTSVRB
The TRRM (task recovery resource manager) updates any SCB (STAE control block) associated with the requester's PRB.		
4 XCTL checks the RBSTAB1 field of the PRB to determine the type of request.		

Diagram 21-9. XCTL Routine (IEAVLK00) (Part 3 of 6)

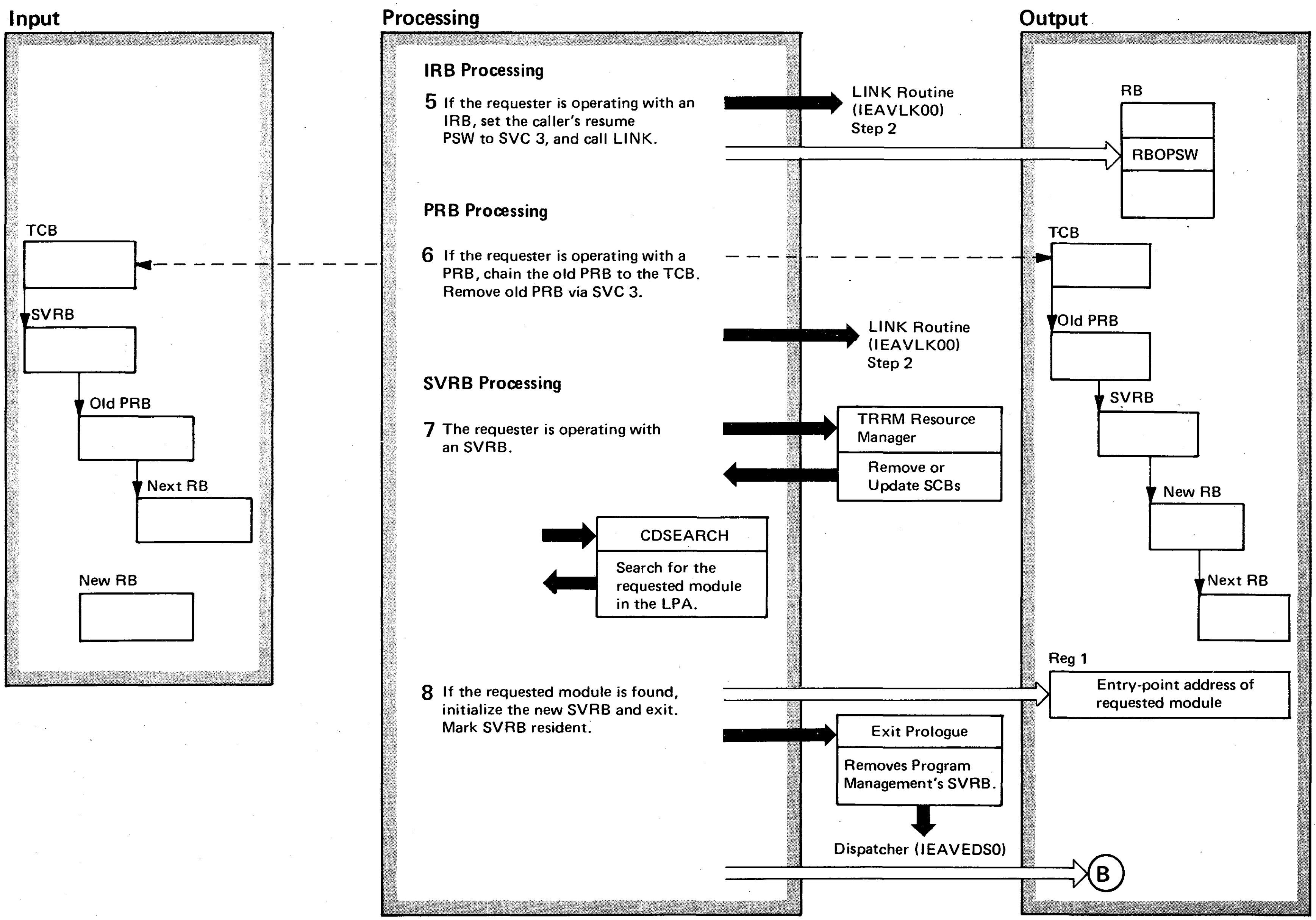

Input
TCB
SVRB
Old PRB
Next RB
New RB
Processing
IRB Processing
5 If the requester is operating with an IRB, set the caller's resume PSW to SVC 3, and call LINK.
PRB Processing
6 If the requester is operating with a PRB, chain the old PRB to the TCB. Remove old PRB via SVC 3.
SVRB Processing
7 The requester is operating with an SVRB.
CDSEARCH
Search for the requested module in the LPA.
8 If the requested module is found, initialize the new SVRB and exit. Mark SVRB resident.
LINK Routine (IEAVLK00) Step 2
LINK Routine (IEAVLK00) Step 2
TRRM Resource Manager
Remove or Update SCBs
Exit Prologue
Removes Program Management's SVRB.
Dispatcher (IEAVEDS0)
Output
RB
RBOPSW
TCB
Old PRB
SVRB
New RB
Next RB
Reg 1
Entry-point address of requested module
B

Diagram 21-9. XCTL Routine (IEAVLK00) (Part 4 of 6)

Extended Description	Module	Label
5 For IRB requests, XCTL sets the resume PSW (RBOPSW field) to the address of an SVC 3 instruction to cause the requester to exit. Control passes to LINK at entry point CDADVANS.		IRBPROC
6 XCTL chains the old PRB to the TCB. The old PRB now points to the SVRB. XCTL removes the old PRB by using the SVC 3 instruction.		
7 For SVRB requests, XCTL passes control to CDSEARCH to search for the requested module in the LPA (Link Pack Area) after regaining control from TRRM.		
8 If found in LPA, XCTL sets the value in the resume PSW (RBOPSW) to the entry-point address of the requested load module, and marks the SVRB as resident in the RBSTAB field, then exits. (Resident means that the SVRB resides in the CDE queue.)		

Diagram 21-9. XCTL Routine (IEAVLK00) (Part 5 of 6)

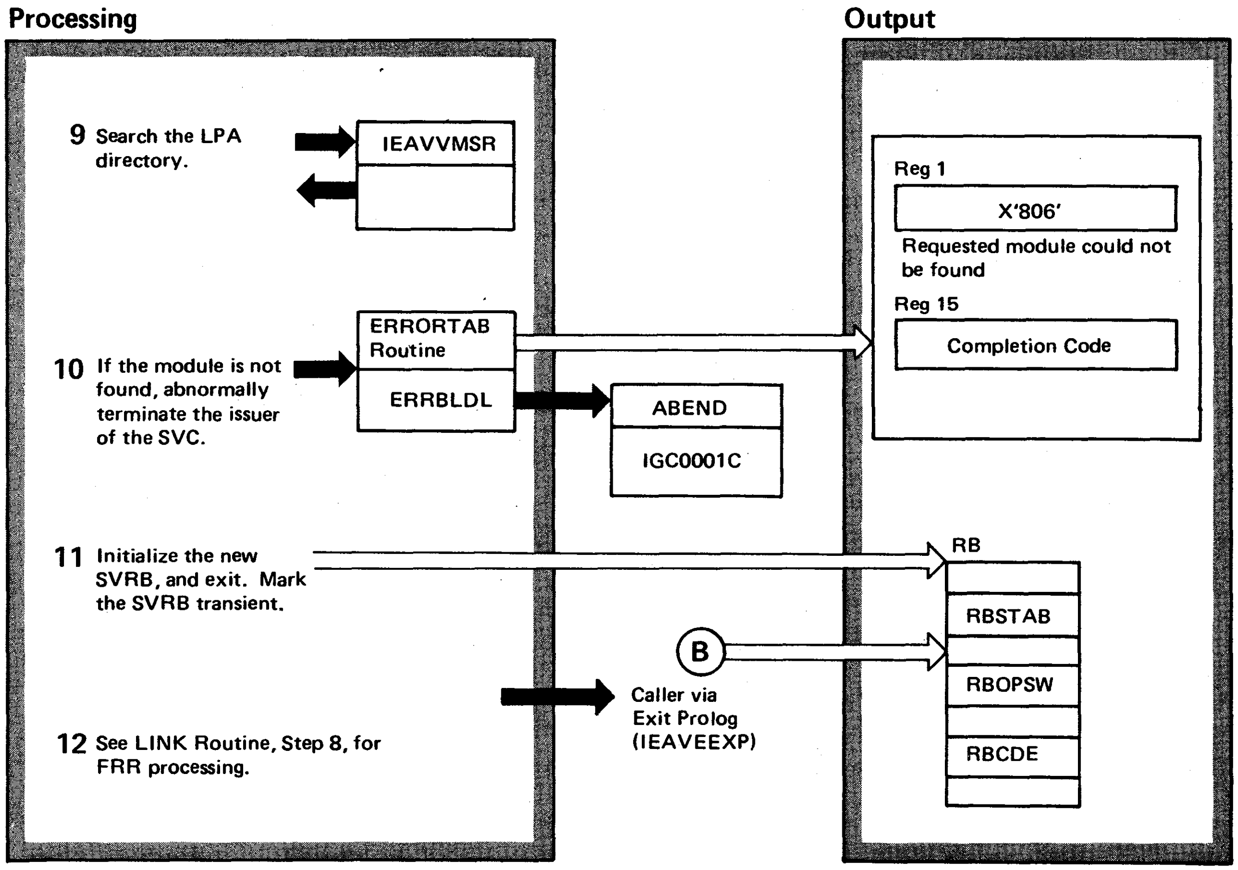
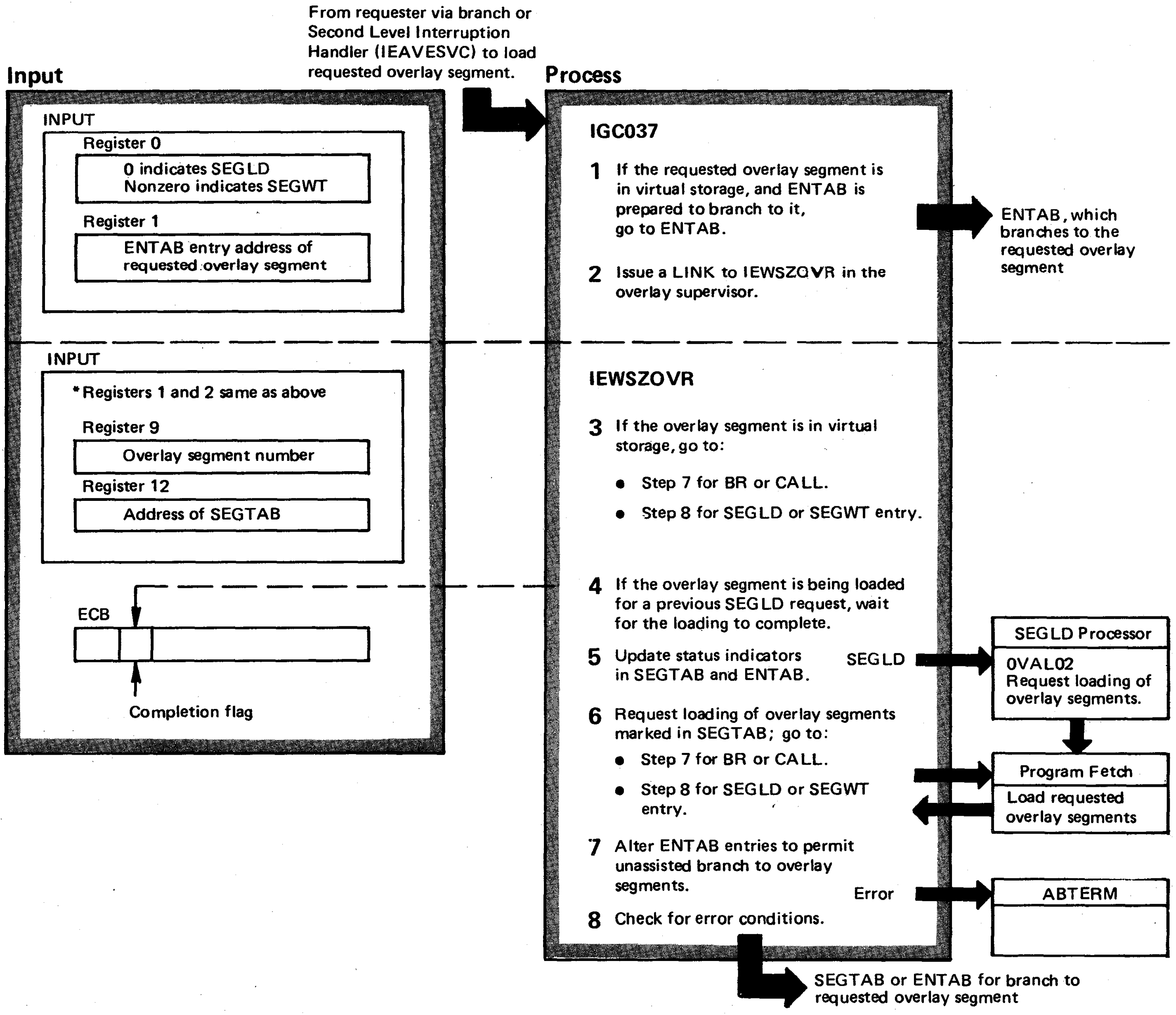

Processing
Output
9 Search the LPA directory.
IEAVVMSR
10 If the module is not found, abnormally terminate the issuer of the SVC.
ERRORTAB Routine
ERRBLDL
ABEND
IGC0001C
Reg 1
X'806'
Requested module could not be found
Reg 15
Completion Code
11 Initialize the new SVRB, and exit. Mark the SVRB transient.
RB
RBSTAB
RBOPSW
RBCDE
B
Caller via Exit Prolog (IEAVEEXP)
12 See LINK Routine, Step 8, for FRR processing.

Diagram 21-9. XCTL Routine (IEAVLK00) (Part 6 of 6)

Extended Description	Module	Label
9 If not found in LPA, XCTL passes control to IEAVVMSR to search the LPA directory.		PLPASRCH
10 If not found on LPDE, XCTL gives control to the ERRORTAB subroutine to create the X'806' error code prior to finally giving ABEND control to abnormally terminate the requester.		
11 If found on LPDE, XCTL sets the value in the resume PSW (RBOPSW) to the entry-point address of the requested load module, and marks the SVRB as transient in the RBSTAB field, then exits. (Transient means that the SVRB resides in the pageable LPA.)		FOUNDEM

Diagram 21-10. Overlay Supervisor (IEWSUOVR and IEWSWOVR) (Part 1 of 2)



From requester via branch or Second Level Interruption Handler (IEAVESVC) to load requested overlay segment.

Input

INPUT

Register 0

0 indicates SEGLD
Nonzero indicates SEGWT

Register 1

ENTAB entry address of requested overlay segment

INPUT

*Registers 1 and 2 same as above

Register 9

Overlay segment number

Register 12

Address of SEGTAB

ECB

Completion flag

Process

IGC037

1 If the requested overlay segment is in virtual storage, and ENTAB is prepared to branch to it, go to ENTAB.

2 Issue a LINK to IEWSZOVR in the overlay supervisor.

ENTAB, which branches to the requested overlay segment

IEWSZOVR

3 If the overlay segment is in virtual storage, go to:

- Step 7 for BR or CALL.
- Step 8 for SEGLD or SEGWT entry.

4 If the overlay segment is being loaded for a previous SEGLD request, wait for the loading to complete.

5 Update status indicators in SEGTAB and ENTAB.

6 Request loading of overlay segments marked in SEGTAB; go to:

- Step 7 for BR or CALL.
- Step 8 for SEGLD or SEGWT entry.

7 Alter ENTAB entries to permit unassisted branch to overlay segments.

8 Check for error conditions.

SEGLD

SEGLD Processor

0VAL02
Request loading of overlay segments.

Program Fetch

Load requested overlay segments

Error

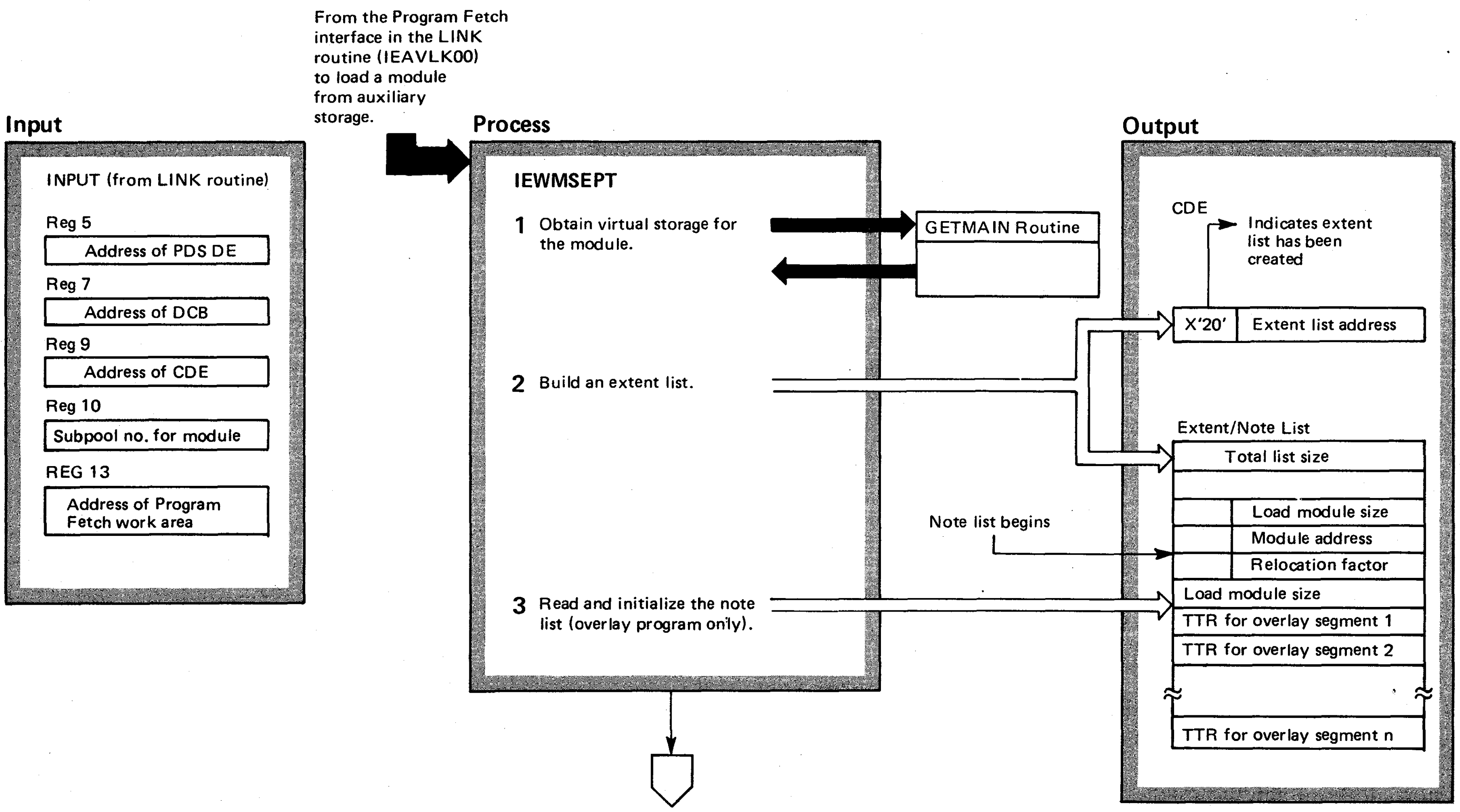
ABTERM

SEGTAB or ENTAB for branch to requested overlay segment

Diagram 21-10. Overlay Supervisor (IEWSUOVR and IEWSWOVR) (Part 2 of 2)

Extended Description	Module	Label	
Overlay is a programming technique that minimizes the virtual storage requirements of a program. When the overlay technique is used, a program is divided into overlay segments, each of which can contain up to 524,288 bytes of text. The overlay supervisor directs the loading of these overlay segments as they are requested. When an overlay program is link-edited, the linkage editor builds an SEGTAB (overlay segment table), and one or more ENTABs (entry tables). It makes these tables part of the overlay module. There is only one SEGTAB in an overlay program. The SEGTAB describes (1) the relationships of overlay segments in the program, and (2) which overlay segments are in virtual storage or being loaded. The SEGTAB is the first portion in the root overlay segment, which contains control information for the overlay program and remains in virtual storage while the overlay program is being executed. There can be an ENTAB in each overlay segment of the program. The overlay supervisor uses the ENTAB to determine which overlay segment must be loaded when a branch instruction or macro instruction refers to an overlay segment not in virtual storage. The overlay supervisor gains control when an overlay segment issues a SEGLD or SEGWT macro request (SVC 37) for another overlay segment, or when an overlay segment issues a CALL macro (SVC 45) or branch instruction to an address in another overlay segment not in virtual storage. The caller enters the resident overlay module, IEWSUOVR.	IEWSUOVR IEWSWOVR	IGC037	This module checks the validity of the input parameters and then issues a LINK to module IEWSWOVR using its alias name, IEWZOVR. If a usable copy of IEWSWOVR is found, it is executed; otherwise, a copy is fetched into virtual storage. IEWSWOVR marks the overlay segments to be overlaid, determines which new overlay segments should be loaded, and branches to Program Fetch to read the overlay segments into virtual storage. A separate branch to Program Fetch is made to read each overlay segment. In both cases, the overlay supervisor examines the SEGTAB to determine whether the requested overlay segment is already in virtual storage, and whether all overlay segments between the requested overlay segment and the root overlay segment are in virtual storage. All must be in virtual storage, and if they are not, the overlay supervisor calls Program Fetch to load them. After the required overlay segments are in virtual storage, if the caller has issued a CALL or branch instruction, the overlay supervisor alters the ENTABs of the loaded overlay segments. The modified ENTABs permit future branches to loaded overlay segments without help from the overlay supervisor. Finally, depending on how it was called, the overlay supervisor passes control to the: • Caller before loading is complete (SEGLD) • Caller after loading is complete (SEGWT) • Branch address in the requested overlay segment after it is loaded (CALL or branch instruction).

Diagram 21-11. Program Fetch (IEWFETCH) (Part 1 of 10)


From the Program Fetch interface in the LINK routine (IEAVLK00) to load a module from auxiliary storage.
Input
INPUT (from LINK routine)
Reg 5
Address of PDS DE
Reg 7
Address of DCB
Reg 9
Address of CDE
Reg 10
Subpool no. for module
REG 13
Address of Program Fetch work area
Process
IEWMSEPT
GETMAIN Routine
Output
CDE
Indicates extent list has been created
X'20'
Extent list address
Extent/Note List
Total list size
Load module size
Module address
Relocation factor
Note list begins
Load module size
TTR for overlay segment 1
TTR for overlay segment 2
TTR for overlay segment n

1 Obtain virtual storage for the module.

2 Build an extent list.

3 Read and initialize the note list (overlay program only).

Diagram 21-11. Program Fetch (IEWFETCH) (Part 2 of 10)

Extended Description	Module	Label
The Program Fetch routine, which is a single module in the nucleus, loads modules for supervisor routines. It transfers modules into virtual storage from libraries (organized as partitioned data sets) on direct access storage devices. Program Fetch reads a module into a continuous block of virtual storage, and relocates address constants in the module. It can process several load requests concurrently.	IEWMSEPT	

The subroutines of program management that search for requested modules and the overlay supervisor use Program Fetch to load modules.

The searching subroutines of program management enter Program Fetch after a LINK, LOAD, XCTL, or ATTACH macro instruction has been issued, and a usable copy of the requested module is not available in virtual storage. For this type of entry, Program Fetch transfers the entire module from auxiliary storage to virtual storage.

The overlay supervisor enters Program Fetch after a SEGWT, SEGID, or CALL macro instruction, or after a branch instruction has been issued for an overlay segment that is not in virtual storage. For this type of entry, Program Fetch loads only the requested overlay segment.

In loading a nonresident module or an overlay segment, the major phases of Program Fetch processing are:

- **Initialization.** Program Fetch initializes a fetch work area, builds an extent list, and (if the module is in an overlay structure) fetches the module's note list. Program Fetch gets virtual storage for the load module.
- **Loading.** Program Fetch calls channel programs that transfer text records, RLD records, and control records into virtual storage.
- **Relocation.** Using the RLD records, Program Fetch changes the values of the address constants in the loaded program from relative load module addresses to absolute virtual storage addresses.
- **Termination.** Program Fetch checks the completion of I/O operations, calculates the relocated module entry-point address in virtual storage, initializes the overlay segment table (if the module is in overlay structure), sets up a return code, and returns control to the caller.

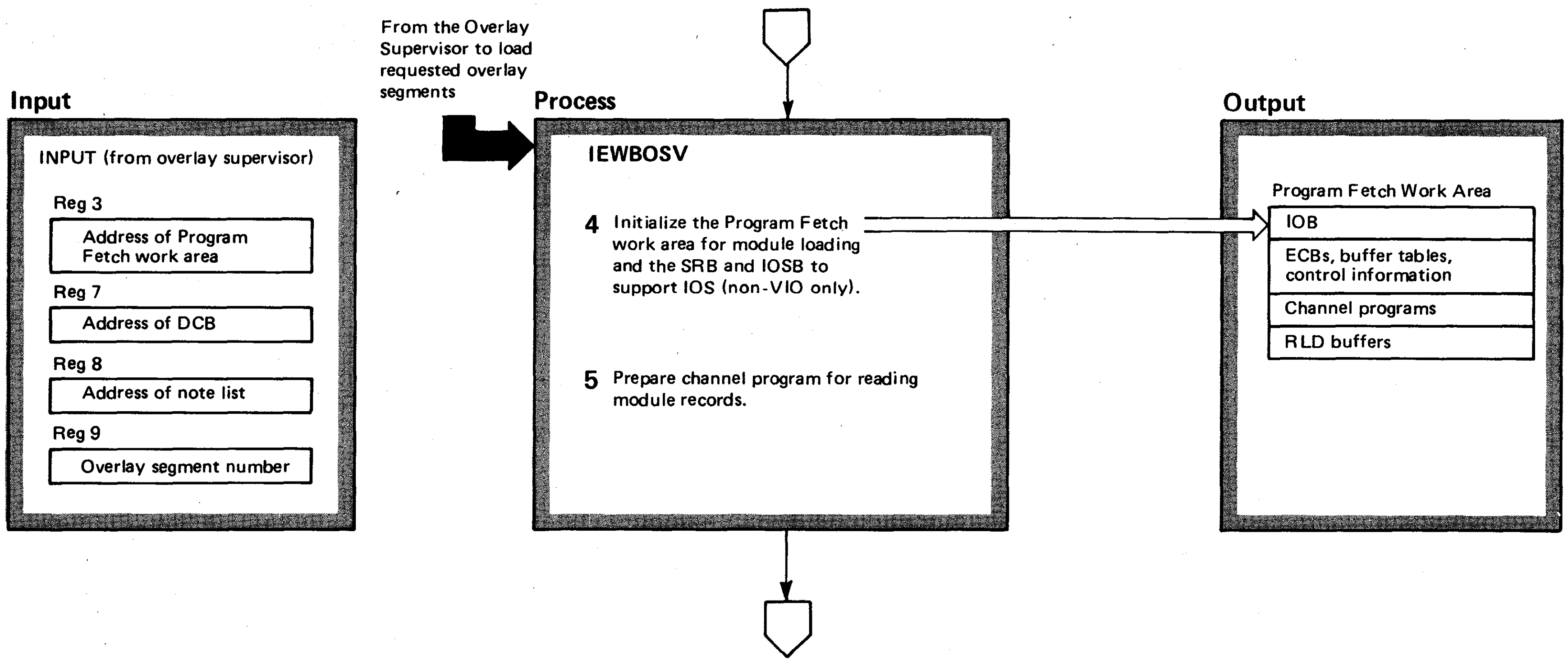
Extended Description	Module	Label

1 Steps 1-5 are the initialization process performed by Program Fetch. During initialization, Program Fetch calls GETMAIN to get the virtual storage it needs for module loading.

2 The extent list contains the virtual storage address of and the length of each section of a module eligible for loading. Program Fetch issues a GETMAIN macro instruction to obtain storage for an extent list (and a note list if the module is in overlay). GETMAIN returns the extent list address and Program Fetch places it in the CDE.

3 If the module being loaded is in overlay, Program Fetch initiates channel programs that read the note list into storage (storage obtained during extent list processing). The linkage editor placed the note list in the overlay module. The note list contains the relative disk address (TTR) for reading each overlay segment of the module. The TTR of the note list is obtained from the PDS DE, converted to an absolute disk address, and used in the channel program request to read the note list into virtual storage. Before the note list is read, Program Fetch builds a note list prefix that it uses when called to load an overlay segment.

Diagram 21-11. Program Fetch (IEWFETCH) (Part 3 of 10)



From the Overlay Supervisor to load requested overlay segments

Input

INPUT (from overlay supervisor)

Reg 3
Address of Program Fetch work area

Reg 7
Address of DCB

Reg 8
Address of note list

Reg 9
Overlay segment number

Process

IEWBOSV

4 Initialize the Program Fetch work area for module loading and the SRB and IOSB to support IOS (non-VIO only).

5 Prepare channel program for reading module records.

Output

Program Fetch Work Area

- IOB
- ECBs, buffer tables, control information
- Channel programs
- RLD buffers

Diagram 21-11. Program Fetch (IEWFETCH) (Part 4 of 10)

Extended Description	Module	Label
4 Program Fetch initializes a work area whose address is furnished by the caller. It places in the work area information that it will use to load the requested module. This information consists of:		

- **An input/output block (IOB).** The IOB provides information that the EXCP Processor needs for its interface with the VIO processor when the program module is being loaded from a VIO data set.
- **An input/output supervisor block (IOSB) and a service request block (SRB).** The IOSB provides information the I/O Supervisor needs when the program module is being loaded from a standard (nonVIO) data set. The SRB provides the structure under which the I/O requests issued by Program Fetch are scheduled by the I/O Supervisor.
- **Two event control blocks (ECBs).** One ECB is posted by the SRB termination routine when the I/O request is complete. The other is posted by the system pagefix routine when requests issued by Program Fetch to fix real storage are complete.
- **Three channel programs.** The channel programs are similar. They are used to overlap the reading of one or more module records with the relocation of address constants pointed to by a previously loaded RLD record.
- **Three RLD buffers.** Each buffer is 260 bytes long and is capable of holding an RLD record, a control record, or a composite control and RLD record.
- **A buffer table.** This table contains a 12-byte entry for each RLD buffer. Each entry contains:
  - A pointer to the next entry.
  - The address of an RLD buffer.
  - The address of a channel program.
- **A text table.** This table is used in CCW translation, and contains:
  - The address of the text CCW currently active in the channel program.
  - The virtual location at which the above CCW is reading text data.

Extended Description	Module	Label
In addition, Program Fetch requests storage for another work area if the DCB (data control block) does not refer to SYS1.LINKLIB, SYS1.SVCLIB, or JOBLIB or if the DCB is not associated with a system request. Program Fetch also sets a switch in the Program Management work area to indicate whether the program module is being loaded from a library authorized by the Authorized Program Facility (APF).		
Program Fetch builds a DCB in the work area; the only valid field in this DCB is a pointer to the DEB. Before copying the DEB into the work area, Program Fetch calls the DEBCHK routine to check the validity of the DEB. The DCB and DEB are used for all I/O requests.	Program Fetch	

5 Preparing for Execution of a Channel Program: Program Fetch passes to the I/O supervisor an absolute disk address at which the first I/O operation is to begin. It does this by:

- Obtaining the relative track and record address (TTR) of the first text record from the data set directory entry, or obtaining the TTR of the needed segment from the note list.
- Converting the relative address to an absolute address, via a branch to a "convert" routine that is resident in the nucleus.
- Placing the absolute disk-seek address in the Program Fetch input/output block (IOB) or IOSB, for later use by the I/O supervisor.

The absolute disk-seek address used for subsequent I/O requests is obtained from count data which is read while loading the text records.

The extent of the module's virtual storage area (text buffer) to be fixed is calculated for each I/O request. This provides real storage for the text CCWs that are introduced in the channel program switching process. The buffer begins at the point when Program Fetch is currently loading text records, and continues for a length of 18K bytes, unless the end of the module is encountered first.

Diagram 21-11. Program Fetch (IEWFETCH) (Part 5 of 10)

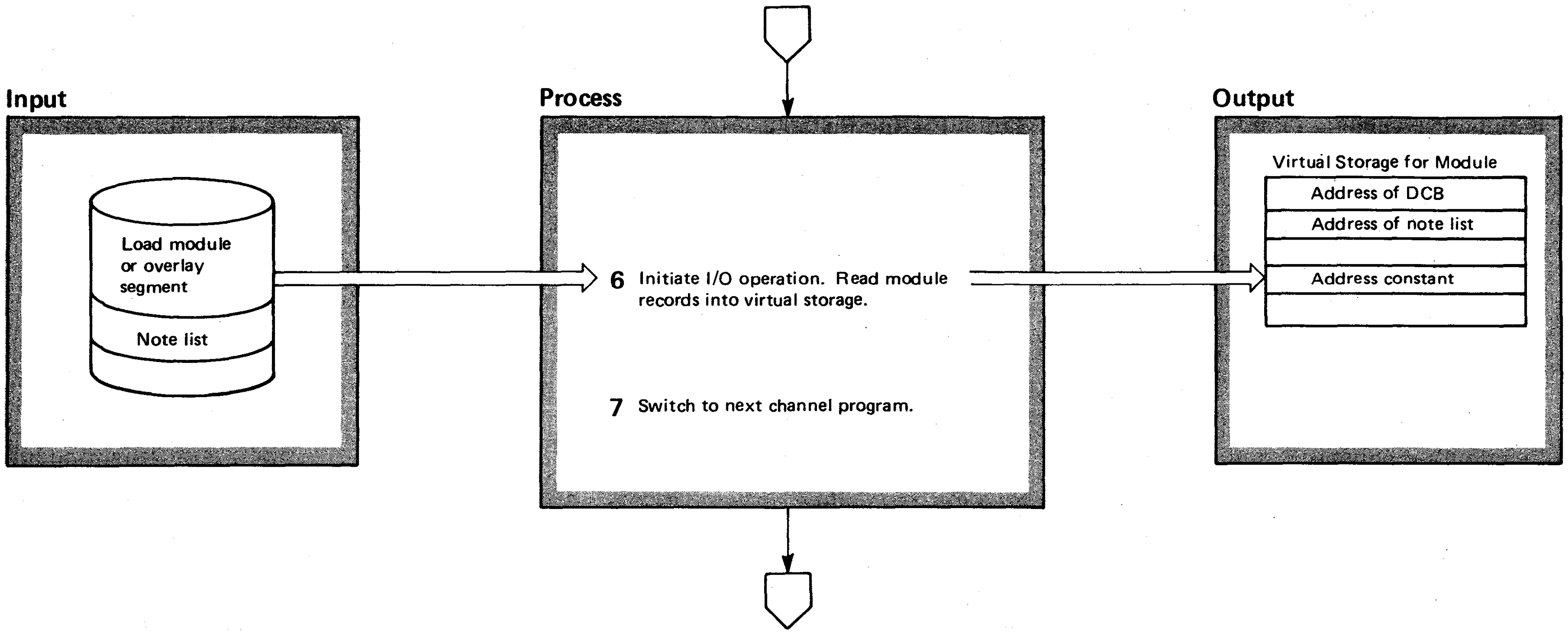

Input
Load module or overlay segment
Note list
Process
6 Initiate I/O operation. Read module records into virtual storage.
7 Switch to next channel program.
Output
Virtual Storage for Module
Address of DCB
Address of note list
Address constant

Diagram 21-11. Program Fetch (IEWFETCH) (Part 6 of 10)

Extended Description	Module	Label
6 Program Fetch starts a channel program by issuing a STARTIO macro instruction to obtain branch linkage to the I/O supervisor. The SRB address is provided as an operand of the macro instruction.		

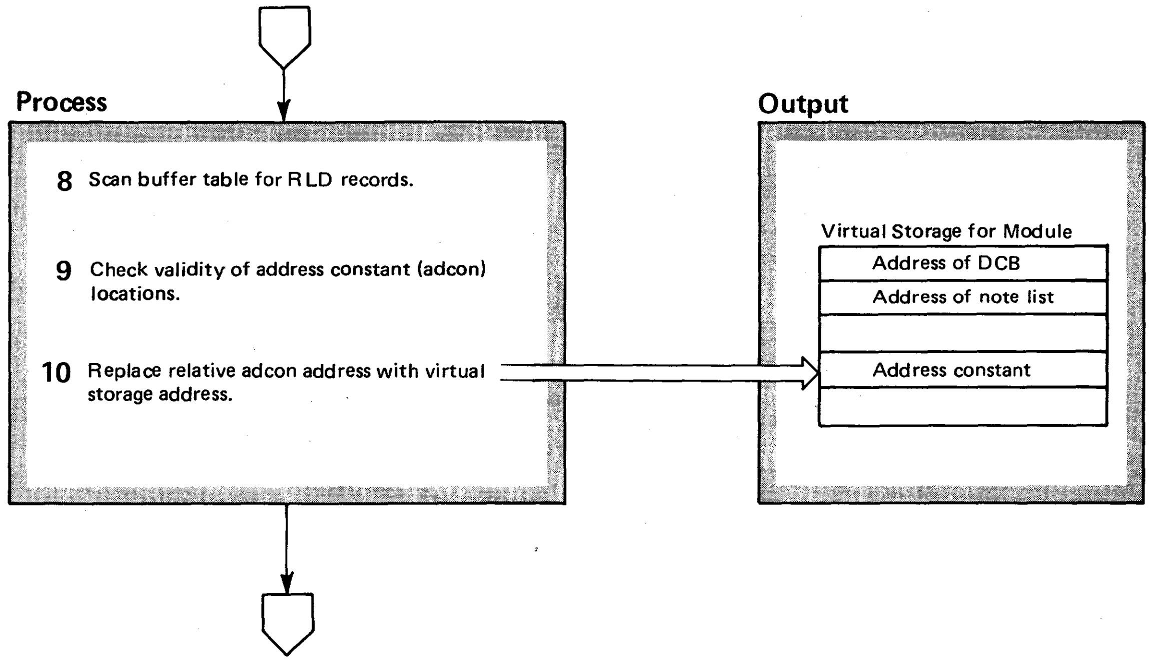
Prior to issuing STARTIO, Program Fetch uses the PGFIX macro instruction to fix its work and the text buffer in real storage. In this manner, page faults are avoided when the I/O supervisor or appendages address the fixed storage.

Other areas referenced during the I/O request are in the fetch work area (fixed for the duration of the loading operation) or are resident in the system nucleus. After these areas are fixed, all Fetch CCWs are translated and an IDAL is built for the text CCW if necessary. The local lock is held while this is done to prevent an address space swap from occurring. An address space swap would cause the real storage addresses referred to by Program Fetch to change.

The text CCWs are retranslated each time a new block of text is to be read. They are translated from information in the text table. For text CCWs that cause page boundaries to be crossed, an IDAL is created. All real addresses are obtained using the LRA instruction.

The I/O supervisor issues a Start I/O instruction, followed by a Stand-Alone Seek command. The Stand-Alone Seek command moves the access arm of the direct access device to the seek address contained in the IOSB. The I/O supervisor, via a Transfer in Channel command, then passes control to a fetch channel program, whose address the Program Fetch routine placed in its IOSB. The fetch channel program causes the first text record to be read into virtual storage. The I/O supervisor returns control to Program Fetch to wait for posting of an event control block by the SRB termination routine. Such posting indicates that the I/O is complete either because the module or segment has been completely read or because a permanent error has occurred.

Diagram 21-11. Program Fetch (IEWFETCH) (Part 7 of 10)



8 Scan buffer table for RLD records.

9 Check validity of address constant (adcon) locations.

10 Replace relative adcon address with virtual storage address.

Diagram 21-11. Program Fetch (IEWFETCH) (Part 8 of 10)

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

8 Switching of Channel Programs: Each channel program reads a text record followed by an RLD or control record, or it reads only the RLD or control record. When a text record is not followed by a control record, the next channel program switches to single-record mode. The single-record mode continues until a control record is encountered causing a switch to two-record mode.

A CCW in each channel program causes a program-controlled interruption (PCI). The PCI causes the I/O supervisor to pass control to the Disabled Interrupt Exit (DIE) routine. The appendage examines the current RLD buffer to determine the channel program switching required, and operates as follows:

- If the current RLD buffer contains an RLD record, the NOP CCW in the current channel program is altered to TIC the CCW, which reads a control record or RLD record into the Program Fetch work area. The TIC address is translated using the LRA instruction.
- If the current RLD buffer contains control information, the text CCW in the next channel program is initialized. Before chaining is attempted, however, the extent of the read is examined to determine whether it exceeds the text buffer fixed for the current I/O request. If the fixed limits are exceeded, the current channel program is not altered and a "buffer full" condition is set. If the text buffer is not exceeded, the current channel program NOP is altered to TIC to the next channel program to read a text record, and a control or RLD record after the text CCW and TIC address have been translated.
- If the current RLD buffer contains an RLD record with the end-of-module indicator, the "end" flag is set. If the buffer contains a control record with the end-of-module indicator, the next channel program is prepared to read a text record only and the "end" flag is set.

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

In all the above cases, the buffer table is examined to determine whether an RLD record was read by the previous channel program, and, if so, the RLD record is passed to the relocate subroutine. Control is then returned to the I/O supervisor.

The Post Status routine (for normal exits) is entered by the I/O supervisor when the channel program has terminated. The appendage returns control to the I/O supervisor to schedule the SRB termination routine when channel end is due to the fact that:

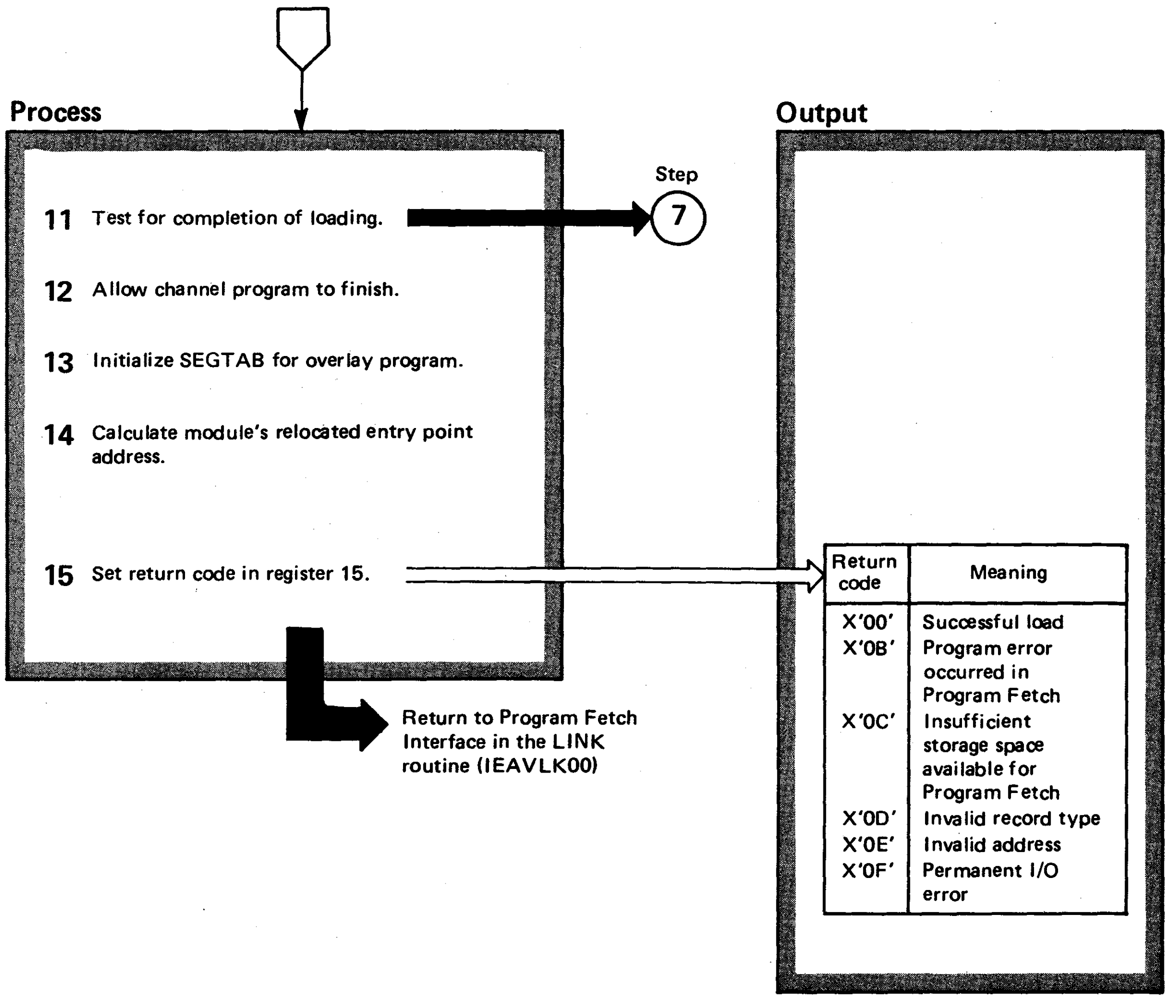
- The entire module or segment has been loaded.
- An invalid record type or an invalid address has been found.
- A permanent I/O error has occurred.

When channel end occurs because the note list has been read, the Post Status routine (for normal exits) resets the channel program to begin reading the program module text and returns control to the I/O supervisor to restart the channel program.

When channel end occurs because the next block of text to be read will lie partially or entirely outside the limits of the currently fixed real-storage buffer area, the Post Status routine (for normal exits) frees the currently fixed area and fixes the new area beginning at the location where the next block of text is to be read. The exit routine then completes translation of the text CCW and returns control to the I/O supervisor to restart the channel program.

When none of the above conditions is present, channel end occurred because the TIC instruction was stored by the DIE routine after the channel had fetched the NOP CCW. In this case, the Post Status routine (for normal exit) returns control to the I/O supervisor to restart the channel program.

Diagram 21-11. Program Fetch (IEWFETCH) (Part 9 of 10)


Process
Output
Step
7
Return to Program Fetch Interface in the LINK routine (IEAVLK00)

11 Test for completion of loading.

12 Allow channel program to finish.

13 Initialize SEGTAB for overlay program.

14 Calculate module's relocated entry point address.

15 Set return code in register 15.

Return code	Meaning
X'00'	Successful load
X'0B'	Program error occurred in Program Fetch
X'0C'	Insufficient storage space available for Program Fetch
X'0D'	Invalid record type
X'0E'	Invalid address
X'0F'	Permanent I/O error

Diagram 21-11. Program Fetch (IEWFETCH) (Part 10 of 10)

Extended Description	Module	Label
11 Program Fetch is restarted after the SRB termination routine has posted an ECB. If the I/O was terminated because of an error, control is passed to the Program Fetch termination routine for cleanup operations; otherwise, the relocation subroutine of Program Fetch then examines the buffer table to determine whether an RLD record (containing relocatable address constants) is in an RLD buffer. If an RLD record was read by the last channel program executed, the relocation subroutine relocates each address constant specified in the record.		
The relocation subroutine adjusts the value of an address constant by combining (adding or subtracting) a relocation factor with the value of the constant. Each RLD record contains the linkage-editor-assigned address of the constant and a flag that indicates addition or subtraction of the relocation factor.		
If the linkage-editor-assigned address of the constant yields a location outside the storage area assigned to the load module, no storing takes place. Control is then passed to the Termination routine.		
13 If the control record before the next text record contains an "end" indicator, the DIE routine sets an "end" flag to inform the termination subroutine. After relocation has been performed, a test of the "end" flag causes the subroutine to be entered.		
The Termination routine performs cleanup operations and places a completion code in the return register.		
The relocated entry-point address is calculated and placed in a register for use by the caller. If the module loaded was the root segment of an overlay program, the address of the DCB and the note list are placed in the segment table for the overlay supervisor.		

Recovery termination management (R/TM) cleans up system resources when a task or address space terminates. Specifically, R/TM performs normal and abnormal task termination, normal and abnormal address space termination, writes dumps, records errors, provides for recovery of supervisory routines via routing control to functional recovery routines, and recovers the system when a CPU in a tightly coupled multiprocessing environment fails. R/TM provides these functions for both system and problem program routines.

Logically, R/TM consists of four interrelated groups of functions that perform R/TM services:

- RTM1: Attempts recovery after a request for an R/TM service from supervisory routines. The CALLRTM macro instruction gives control to RTM1. RTM1 resides in the nucleus.
- RTM2: Performs normal and abnormal task termination for both system and problem program routines. The ABEND macro instruction (SVC 13) requests these RTM2 services. RTM2 resides in the link pack area (LPA).
- Address space termination: Provides normal and abnormal address space termination for supervisory routines. The CALLRTM macro instruction is used to request this service. Address space termination resides in the LPA.
- R/TM support functions: Provide error recording, formatting of dumps, creating recovery control blocks for STAE, ESTAE, STAI, ESTAI, and ESTAR, and recovering from the failure of a CPU in a tightly coupled multiprocessing system.

RTM1 Functions

RTM1 attempts recovery from hardware and software errors for routines protected by FRRs (functional recovery routines, defined by the routine that requests the recovery protection). RTM1 schedules RTM2 processing to terminate those tasks or address spaces, via SVC 13, that cannot recover. To achieve recovery, RTM1 routes control to the FRRs when program checks, machine checks, paging errors, invalid SVCs, or restarts occur.

RTM1 functions are divided into three logical categories:

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

SLIH Mode Processing

RTM1, when in SLIH mode, schedules recovery for errors in system-mode functions, and initiates recovery for errors in task-mode processing. (See the "RTM2 Services" section for a description of recovering from errors in task-mode processing.) System mode recovery involves routing control to functional recovery routines (FRRs) and requesting error recording.

To implement recovery for system-mode functions, RTM1 routes control to the FRRs defined on FRR stacks for specific paths through the supervisor. (The MO Diagram "Routing to FRRs" fully defines the FRR stacks and the paths through the supervisor that they protect.) The system-mode functions use the SETFRR macro instruction (an inline-expanding macro instruction that places the address of the FRR on the stack) to make the FRR known to the system; supervisor control FRRs are placed in the system at initialization time. When an error occurs, RTM1 routes control to the FRRs, thus allowing a recovery path through system-mode functions.

Service Mode Processing

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

For task termination, requested by a CALLRTM TYPE=ABTERM macro instruction, RTM1 establishes an interface to RTM2. This interface differs for tasks in the current, or executing, address space, or for tasks in another address space. For ABTERM of a task in the current address space, RTM1 sets the RB (request block) resume PSW to point to the address of an SVC 13 instruction which will be executed first when it is redispatched. For ABTERM of a task in another address space, RTM1 must first reschedule itself as an SRB (service request block) in the address space executing the task to be terminated. Thus it appears that the CALLRTM TYPE=ABTERM request was issued by a task in the same address space. RTM1 uses this interface to give control to RTM2 as an RB issuing an SVC 13 instruction. RTM2 performs the actual recovery/termination processing.

The PGIOERR (page I/O error) service request differs for non-locked tasks or for locked tasks and SRBs. For non-locked tasks, RTM2 sets an RB to point to an SVC 13 instruction, thereby giving control to RTM2 to execute a task termination. For locked tasks or SRBs, RTM1 establishes an interface to allow FRRs to gain control. RTM1 does this by causing the task or SRB to invalidly issue an SVC. This effects an re-entry into RTM1 in SLIH mode; RTM1 can then route control to FRRs defined for the path that failed. Figure 2-46 illustrates PGIOERR processing, and refers to MO diagrams in the Method of Operation section that describes the processing.

Hardware Error Mode

RTM1, when operating in hardware error mode, logically operates as a subroutine of the machine check handler (MCH). (See the publication *OS/VS2 Recovery Management Support Logic,* SY27-7250, for a complete description of the MCH.) RTM1 performs software repair, gathers data about the error, and records the error. When MCH cannot recover from the error, RTM1 sets up an MCH re-entry to attempt software repair. Figure 2-47 illustrates how RTM1 handles a hardware error.

RTM2 Functions

RTM2 terminates tasks and controls the clean up of their associated resources and control blocks. RTM2 handles normal tasks termination tasks that cannot complete their processing due to an error. Resource managers, routines called by RTM2, clean up the resources and control blocks associated with a task or address space to complete termination. No longer does R/TM, when performing termination, clean these resources; the component owning the resource performs the clean up.

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

Normal Termination

When the last program to be executed for a task ends, it returns control to the EXIT routine. EXIT gives control to RTM2 to perform normal end-of-task processing. Figure 2-48 depicts the steps that occur for normal task termination. (The Task Management section describes EXIT and exit prolog processing in detail.)

Abnormal Termination

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

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

In task termination, only the resources of the current (failing) task and its subtasks are released. The current task (the task being terminated) is treated as the top terminating task (the highest-level task in the chain of terminating tasks); the current task and all its subtasks are abnormally terminated.

In step termination, all tasks in the job step are terminated. The job-step task is treated as the top terminating task; the chain of terminating tasks originates with step task, the highest-level task in the job step, produces the same result as a step termination.

For abnormal termination, RTM2 provides the following services:

- Retry of a terminating task, if possible.

- Allowing tasks that cannot retry to process special exits.
- Display a snapshot of storage.
- Wait for subtask termination to complete.
- Purge subtask resources.
- Convert ABEND requests to the jobstep level.

Figure 2-49 shows how RTM2 handles an abnormal termination, and points to MO diagrams in the Method of Operation section that describes the processing.

Retry Terminating Tasks

RTM2 permits task scheduled for termination to bypass termination and resume processing if they have created exits for this function. These exits are:

- STAE (specify task asynchronous exit).
- ESTAE (extended STAE).
- STAI (specify task asynchronous interruption).
- ESTAI (extended STAI).
- ESTAR (extended specify task asynchronous retry).

These exits receive control from RTM2 prior to termination completing. (This facility complements the FRR facility in RTM1.) The exits may attempt to recover the task being terminated; if successful, RTM2 does not terminate the task. If the exit does not recover the task, task termination continues. Figure 2-50 shows retry.

Term Exits

Whereas RTM2 allows retry during most task terminations, certain conditions, for example CANCEL requests, ancestor task abnormally terminating, timer expiration, cannot be retried. However, a special feature of ESTAE/ESTAI exits, called the "Term" option, can be used to enable an ESTAE or ESTAI exit to gain control during these situations. (The user indicates this by specifying TERM=YES when the ESTAE or ESTAI is issued.) During "normal" error recovery processing for a task, these exits function in exactly the same way as exits created without the Term option. But for a situation that cannot be retried, these specially marked exits are given control so that a user may clean-up resources, write records, print messages, or perform any other important function before RTM2 completes the termination. Retry, even though requested, is not permitted by RTM2. Figure 2-51 shows how RTM2 processes a CANCEL request and routes control to term exits.

Display Storage

RTM2 will display storage, via SNAP, for all tasks in the failing task tree, when requested by the DUMP option.

Wait for Subtask Termination

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

Purge Subtasks

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

Convert to Step

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

Address Space Termination

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

Address space termination begins after RTM1 invokes the address space termination controller, by scheduling the address space termination SRB to post it. The address space termination controller determines the address space being terminated, and dequeues the ASCB. The address space termination controller then attaches the address space termination task to complete the termination. The termination will be complete after all the resources associated with the address space have been purged by the address space termination controller and RTM2. The figure 2-52 shows the control flow of an address space termination.

Recovery Termination Management Support Functions

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

R/TM services consist of the following:

- STA (specify task asynchronous conditions) and ESTA (extended STA) services. STA and ESTA services create SCBs (STA control blocks) to represent user-written abnormal condition exits. R/TM will give control to these exits during termination processing.
- ACR (alternate CPU recovery). ACR provides a method for the system to continue functioning after one CPU in a tightly coupled multiprocessing system fails.
- SETFRR. This is an inline-expanding macro instruction that places an FRR (functional recovery routine) on the correct FRR stack. R/TM routines route control to FRRs after an error occurs.
- Initializing FRR stacks. This creates FRR stacks during system initialization, and changes FRR stacks in response to VARY CPU commands.
- SVC 51. SVC 51 provides formatted or unformatted displays. SVC 51 include SNAP dump, SVC DUMP, and schedule dump.
- CHNGDUMP (change dump). The CHNGDUMP operator command overrides the dump options in the system for SDUMP and ABEND dumps.
- Recording. R/TM uses recording to record errors and records created during recovery or termination processing.

STA Services

The STA services create SCB (STA control blocks) that represent caller-requested asynchronous exits. STA services, requested via an SVC 60 instruction, create five types of SCBs:

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

Alternate CPU Recovery (ACR)

ACR provides a multiprocessing system the ability to recover system operation, executing on the operational CPU, after one CPU fails. ACR saves as much work from the failing CPU as possible, and terminates work it cannot save. ACR performs this by treating the work in progress as an abnormal termination condition. This allows ACR to attempt software recovery through the use of recovery and retry routines defined in the system at the time of the malfunction in the failing CPU. ACR will also remove I/O devices, channel paths, or other CPU dependencies affiliated with the failing CPU by placing them offline.

SETFRR

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

Initializing FRR Stacks

During initialization, this function initializes the FRR stacks used by the system, and places pointers to these stacks in the RSVT (recovery stack vector table) of the PSA. The VARY CPU command can use this function. The FRR stacks initialized by this function are:

- SVC-I/O-dispatcher stack, used by these supervisor control routines.
- Machine check stack, used by the machine check handler after a machine check occurs.
- Program check stack, used by the program check handler after a program check occurs.
- The three external interrupt handler stacks, used by the external interrupt handler to process three levels of recursion. (See the Supervisor Control section for a description of the external interrupt handler and its use of the FRR stacks.)
- Restart interrupt handler stack, used by the restart interrupt handler.
- Normal stack, used by supervisor control routines processing on behalf of problem programs which utilize supervisor services.

Dumping

SVC 51 produces two types of dumps — formatted and unformatted. The following text explains formatted and unformatted dumps.

Formatted Dump — SNAP Dump

The SNAP routine is invoked by a SNAP macro instruction. The SNAP macro instruction, whose expansion contains an SVC 51, causes the SVC (SVC SLIH) to call the SVC DUMP routine. The SVC DUMP routine checks the SNAP parameter list to determine whether a SNAP macro instruction has been issued. If so, the SVC DUMP routine passes control to the SNAP routine.

The SNAP macro instruction can be issued by ABEND Dump during abnormal termination, or by a user program at any time. Thus, SNAP processing can provide either a formatted abnormal dump or a formatted dynamic dump. The ABEND Dump routine can specify either a SYSABEND or a SYSUDUMP dump.

The default dump options for a SYSABEND dump consist of the major control blocks belonging to the task, enqueue control blocks, LSQA (local system queue area), programs and dynamically acquired storage, and the GTF or trace table entries. The default dump options for a SYSUDUMP dump differ only in the omission of LSQA.

These default options reside in the SYS1.PARMLIB members IEAABD00 for SYSABEND, and in IEADMP00 for SYSUDUMP. SNAP dump processing merges these options with those specified on the request. Figure 2-53 shows how SNAP determines the type of dump and dump options requested.

If a dynamic dump is requested (the SNAP macro is issued by a user program), the storage areas to be dumped are specified by the operands of the SNAP macro. (See *OS/VS2 Supervisor Services and Macro Instructions* for information on how to obtain a dump.)

The use of the SNAP routine is restricted to tasks that do not have job-step tasks within their subtask structure at entry to SNAP processing. If a task has a subtask that is a job-step task, control is returned immediately to the caller.

Unformatted Dump — SVC DUMP

The SVC DUMP service provides a quick, unformatted dump of virtual storage directly to a data set. To use SVC DUMP, callers must have APF (authorized program facility) authorization, or be in control program key. The SDUMP macro instruction calls SVC DUMP processing either by SVC 51 or branch entry.

The SVC DUMP service consists of three routines;

- NIP initialization, which sets up the SYS1.DUMP data sets as specified by the operator with the DUMP option in the 'SPECIFY SYSTEM PARAMETERS' command.
- SVC 51, which performs the dump of virtual storage.
- Dump task, a permanent task in each address space, which dumps the contents of each address space.
- Nucleus routine, which handles branch entries and schedules the dump task in the specified address space.

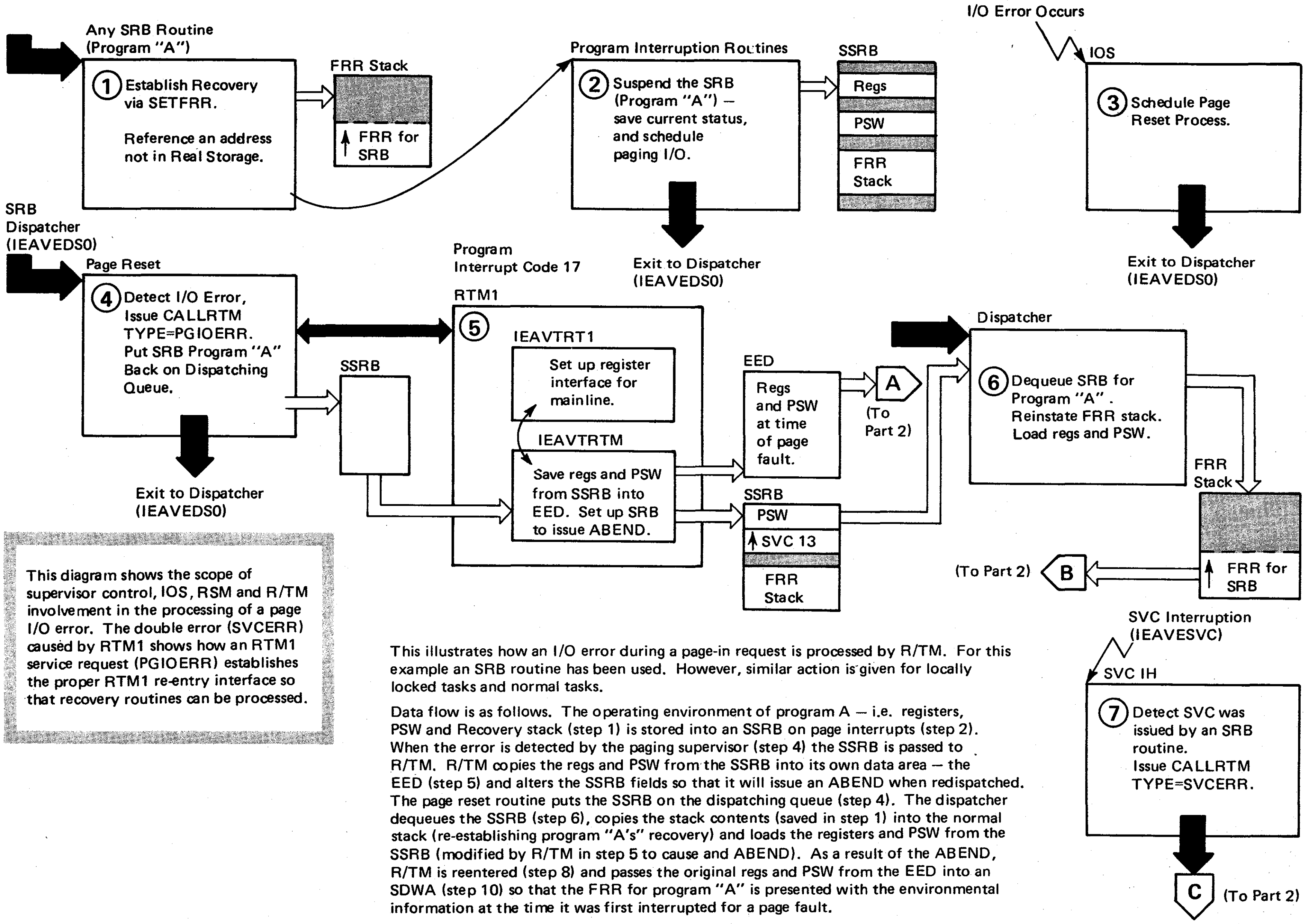
The SVC 51 routine dumps the contents of virtual storage from the address space in which the request occurred - operating under the caller's task - or it initiates the dump of another address space by posting the permanent dump task in the destination address space being dumped by scheduling an SRB and operating under the dump task. Figure 2-54 illustrates the dump function.

CHNGDUMP Operator Command

The CHNGDUMP operator command overrides any dump options that already exist in the system, and allows the operator to create new options that differ from the existing options. (See the publication *OS/VS2 System Programming Library: Supervisor*, GC28-0628, for a complete description of the CHNGDUMP operator command and its uses.)

Recording Services

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


Any SRB Routine
(Program "A")
1 Establish Recovery via SETFRR.
Reference an address not in Real Storage.
FRR Stack
FRR for SRB
Program Interruption Routines
2 Suspend the SRB (Program "A") – save current status, and schedule paging I/O.
SSRB
Regs
PSW
FRR Stack
Exit to Dispatcher (IEAVEDSO)
I/O Error Occurs
IOS
3 Schedule Page Reset Process.
Exit to Dispatcher (IEAVEDSO)
SRB Dispatcher (IEAVEDSO)
Page Reset
4 Detect I/O Error, Issue CALLRTM TYPE=PGIOERR. Put SRB Program "A" Back on Dispatching Queue.
Exit to Dispatcher (IEAVEDSO)
SSRB
Program Interrupt Code 17
RTM1
5
IEAVTRT1
Set up register interface for mainline.
IEAVTRTM
Save regs and PSW from SSRB into EED. Set up SRB to issue ABEND.
EED
Regs and PSW at time of page fault.
A
(To Part 2)
SSRB
PSW
SVC 13
FRR Stack
Dispatcher
6 Dequeue SRB for Program "A". Reinstate FRR stack. Load regs and PSW.
FRR Stack
FRR for SRB
(To Part 2)
B
SVC Interruption (IEAVESVC)
SVC IH
7 Detect SVC was issued by an SRB routine. Issue CALLRTM TYPE=SVCERR.
C
(To Part 2)

This diagram shows the scope of supervisor control, IOS, RSM and R/TM involvement in the processing of a page I/O error. The double error (SVCERR) caused by RTM1 shows how an RTM1 service request (PGIOERR) establishes the proper RTM1 re-entry interface so that recovery routines can be processed.

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

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

Figure 2-46. Page I/O Error Processing (Part 1 of 2)

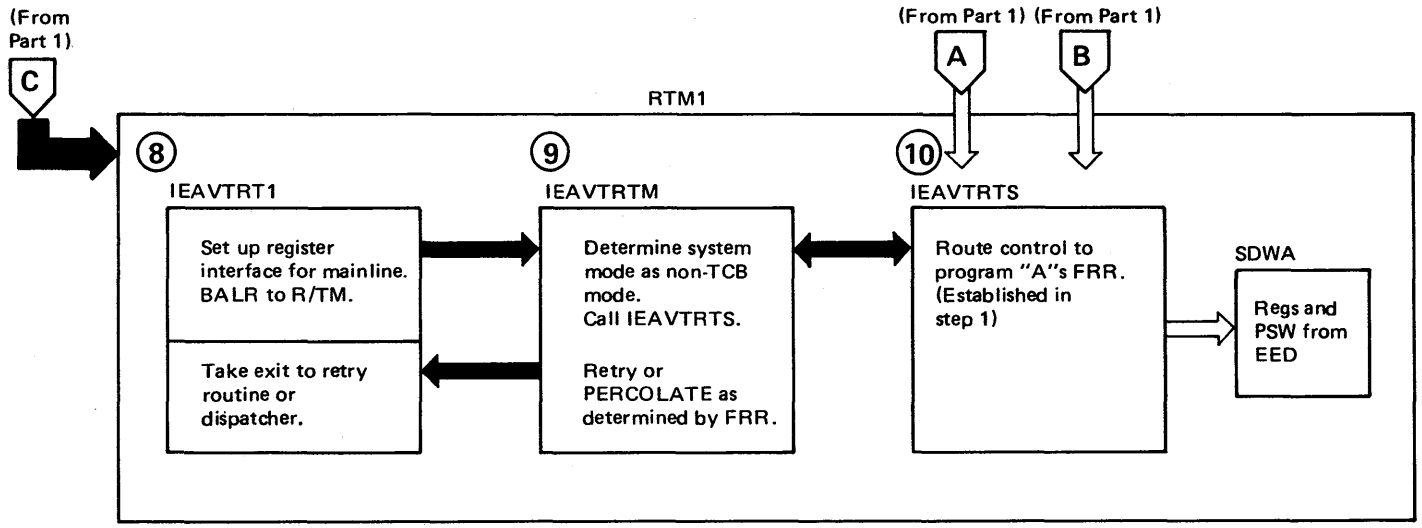
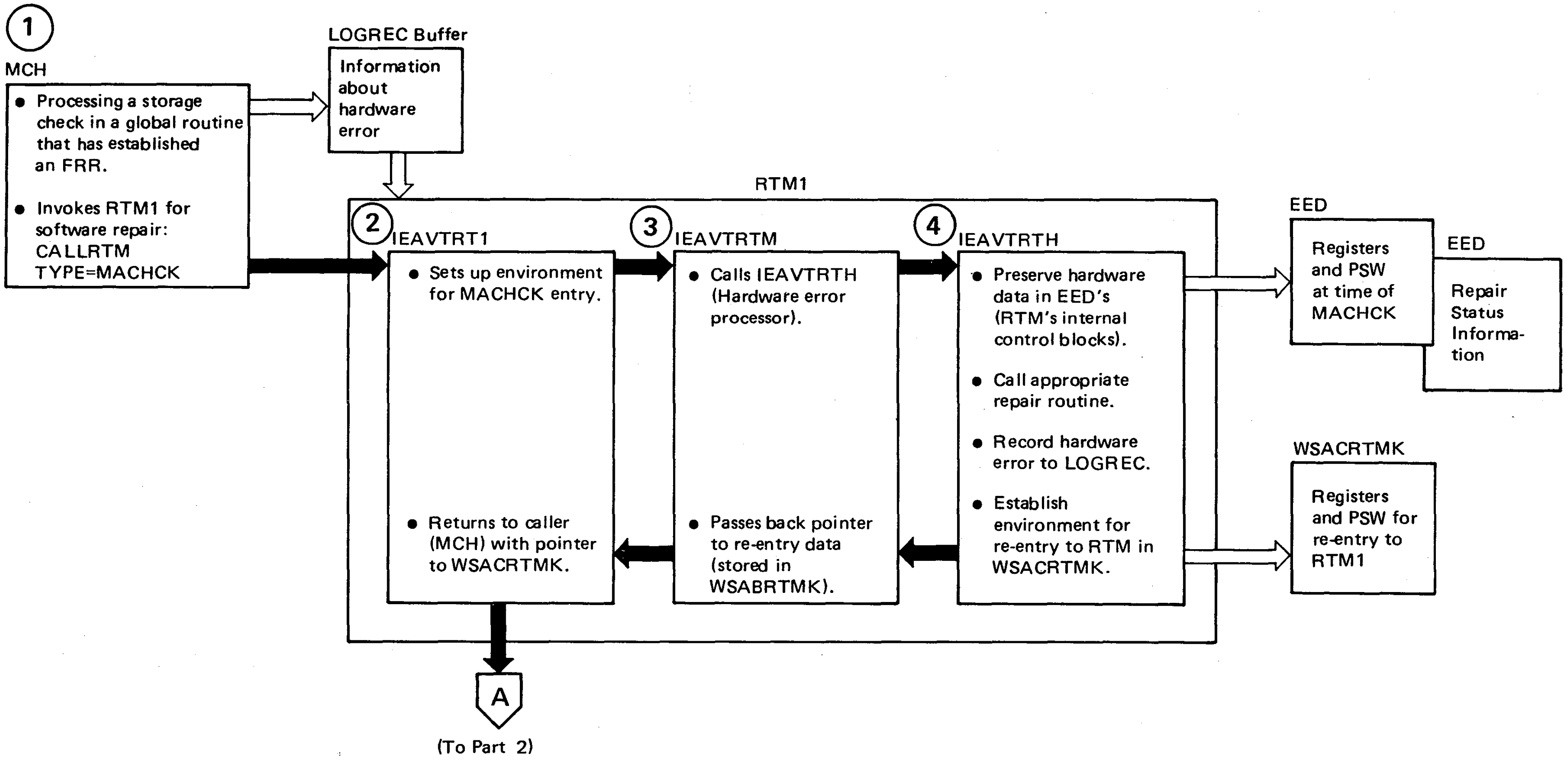

(From Part 1)
C
(From Part 1)
A
(From Part 1)
B
RTM1
8
IEAVTRT1
Set up register interface for mainline. BALR to R/TM.
Take exit to retry routine or dispatcher.
9
IEAVTRTM
Determine system mode as non-TCB mode. Call IEAVTRTS.
Retry or PERCOLATE as determined by FRR.
10
IEAVTRTS
Route control to program "A"s FRR. (Established in step 1)
SDWA
Regs and PSW from EED

Figure 2-46. Page I/O Error Processing (Part 2 of 2)


1
MCH
Processing a storage check in a global routine that has established an FRR.
Invokes RTM1 for software repair: CALLRTM TYPE=MACHCK
LOGREC Buffer
Information about hardware error
RTM1
2 IEAVTRT1
Sets up environment for MACHCK entry.
Returns to caller (MCH) with pointer to WSACRTMK.
3 IEAVTRTM
Calls IEAVTRTH (Hardware error processor).
Passes back pointer to re-entry data (stored in WSABRTMK).
4 IEAVTRTH
Preserve hardware data in EED's (RTM's internal control blocks).
Call appropriate repair routine.
Record hardware error to LOGREC.
Establish environment for re-entry to RTM in WSACRTMK.
EED
Registers and PSW at time of MACHCK
EED
Repair Status Informa-tion
WSACRTMK
Registers and PSW for re-entry to RTM1
A
(To Part 2)

This depicts the processing for a "hard" type machine check in a global routine which has FRR recovery. It shows the interfaces and control flow between the machine check handler and RTM1 for both hardware error processing and the resulting software recovery attempt by the FRR. It alludes to the fact that software recovery will continue in task mode, because in this example the FRR does not recover the error.

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

The R/TM CPU-related work save area (WSACRTMK) is used by RTM1 to alter the registers and the PSW that MCH will reload — thereby determining whether MCH will resume the interrupted process ('soft' error), or reenter RTM1 for software recovery ('hard' error).

Figure 2-47. Hardware Error Processing (Part 1 of 2)

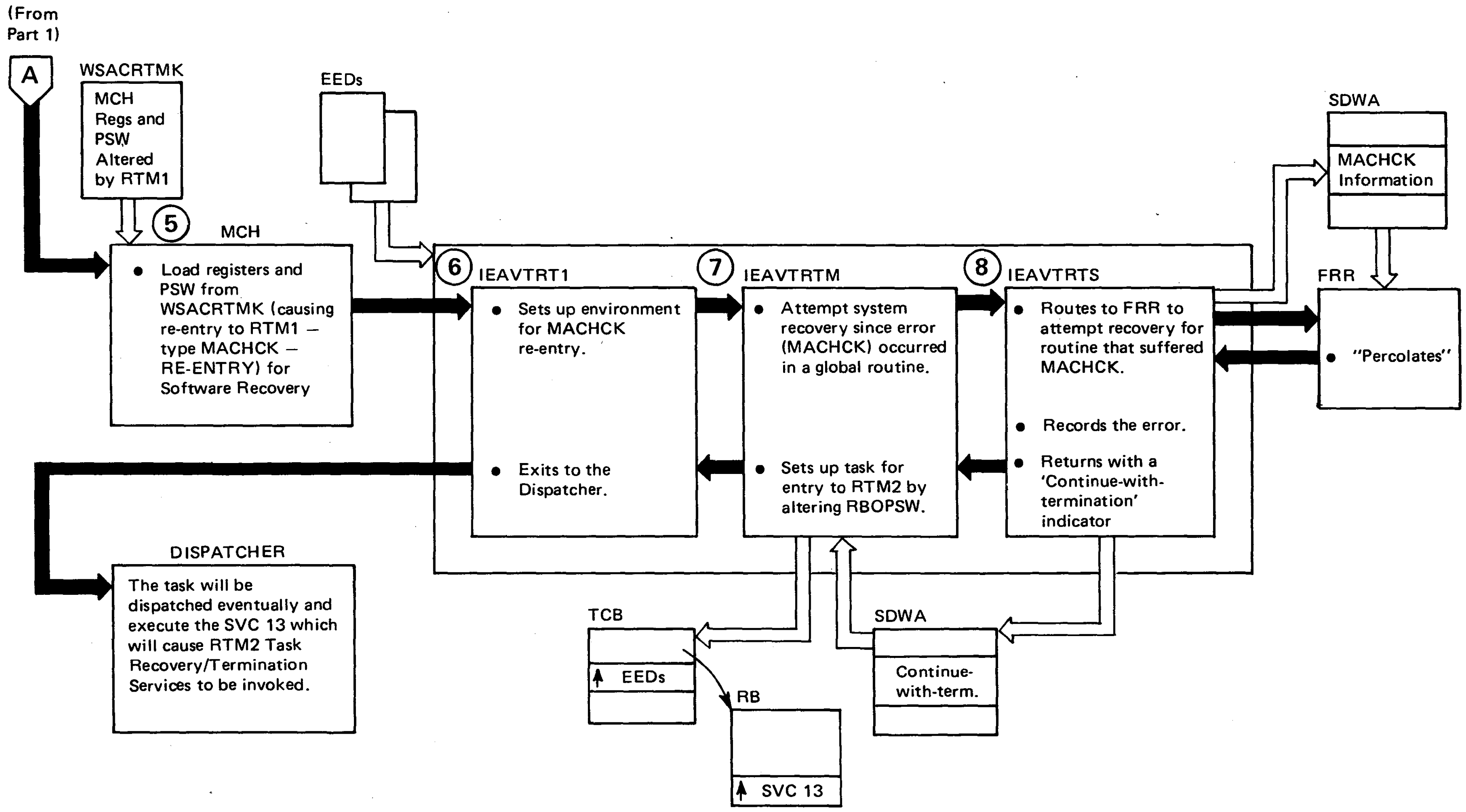

(From Part 1)
A
WSACRTMK
MCH Regs and PSW Altered by RTM1
EEDs
5
MCH
• Load registers and PSW from WSACRTMK (causing re-entry to RTM1 – type MACHCK – RE-ENTRY) for Software Recovery
6
IEAVTRT1
• Sets up environment for MACHCK re-entry.
• Exits to the Dispatcher.
7
IEAVTRTM
• Attempt system recovery since error (MACHCK) occurred in a global routine.
• Sets up task for entry to RTM2 by altering RBOPSW.
8
IEAVTRTS
• Routes to FRR to attempt recovery for routine that suffered MACHCK.
• Records the error.
• Returns with a 'Continue-with-termination' indicator
SDWA
MACHCK Information
FRR
• "Percolates"
DISPATCHER
The task will be dispatched eventually and execute the SVC 13 which will cause RTM2 Task Recovery/Termination Services to be invoked.
TCB
EEDs
RB
SVC 13
SDWA
Continue-with-term.

Figure 2-47. Hardware Error Processing (Part 2 of 2)

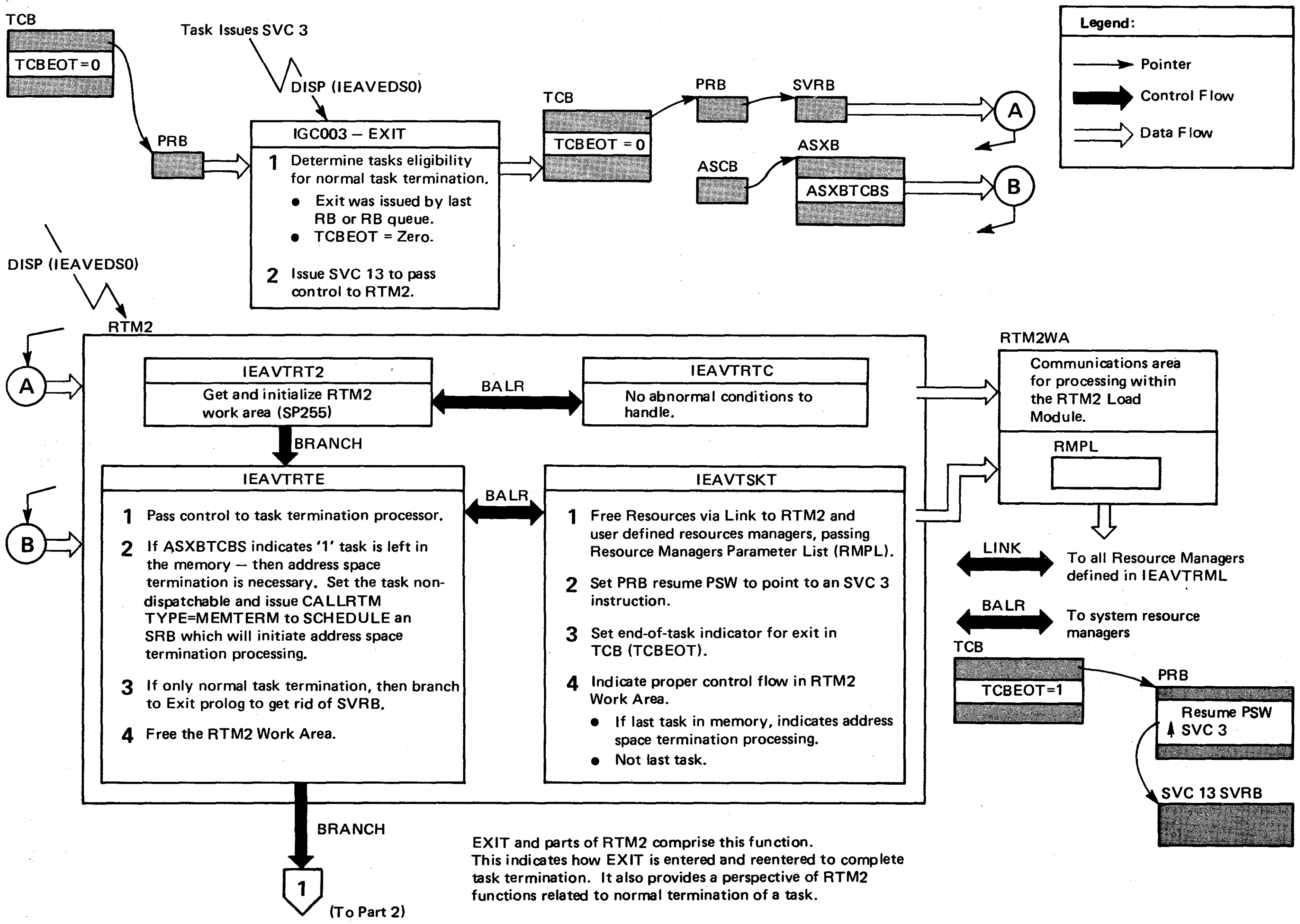

TCB
TCBEOT=0
PRB
Task Issues SVC 3
DISP (IEAVEDS0)
IGC003 – EXIT
1 Determine tasks eligibility for normal task termination.
• Exit was issued by last RB or RB queue.
• TCBEOT = Zero.
2 Issue SVC 13 to pass control to RTM2.
TCB
TCBEOT = 0
PRB
SVRB
A
ASCB
ASXB
ASXBTCBS
B
Legend:
Pointer
Control Flow
Data Flow
DISP (IEAVEDS0)
RTM2
A
B
IEAVTRT2
Get and initialize RTM2 work area (SP255)
BALR
IEAVTRTC
No abnormal conditions to handle.
BRANCH
IEAVTRTE
1 Pass control to task termination processor.
2 If ASXBTCBS indicates '1' task is left in the memory – then address space termination is necessary. Set the task non-dispatchable and issue CALLRTM TYPE=MEMTERM to SCHEDULE an SRB which will initiate address space termination processing.
3 If only normal task termination, then branch to Exit prolog to get rid of SVRB.
4 Free the RTM2 Work Area.
BALR
IEAVTSKT
1 Free Resources via Link to RTM2 and user defined resources managers, passing Resource Managers Parameter List (RMPL).
2 Set PRB resume PSW to point to an SVC 3 instruction.
3 Set end-of-task indicator for exit in TCB (TCBEOT).
4 Indicate proper control flow in RTM2 Work Area.
• If last task in memory, indicates address space termination processing.
• Not last task.
RTM2WA
Communications area for processing within the RTM2 Load Module.
RMPL
LINK
To all Resource Managers defined in IEAVTRML
BALR
To system resource managers
TCB
TCBEOT=1
PRB
Resume PSW
SVC 3
SVC 13 SVRB
BRANCH
1
(To Part 2)
EXIT and parts of RTM2 comprise this function. This indicates how EXIT is entered and reentered to complete task termination. It also provides a perspective of RTM2 functions related to normal termination of a task.

Figure 2-48. The Process of Normal Task Termination (Part 1 of 2)

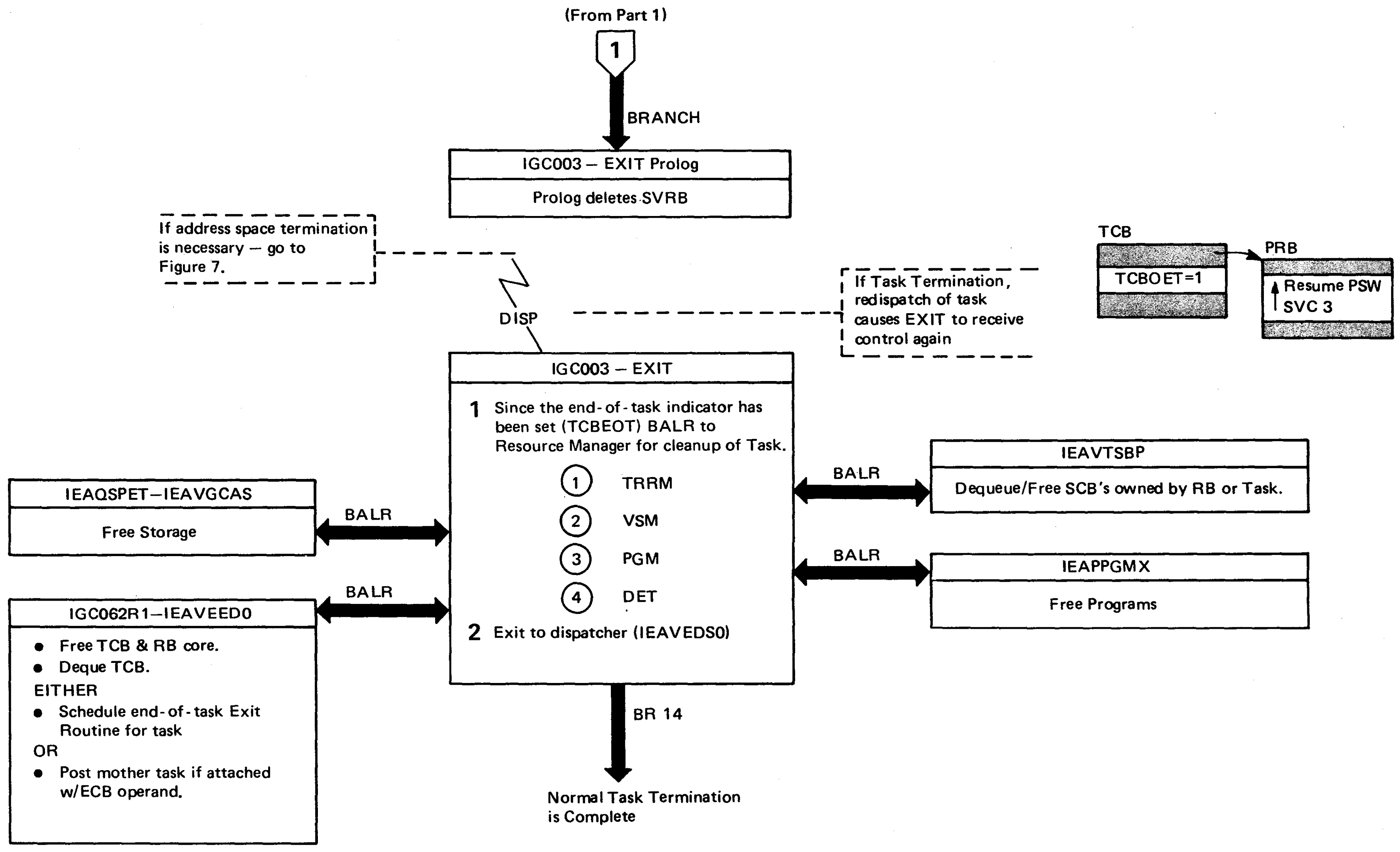
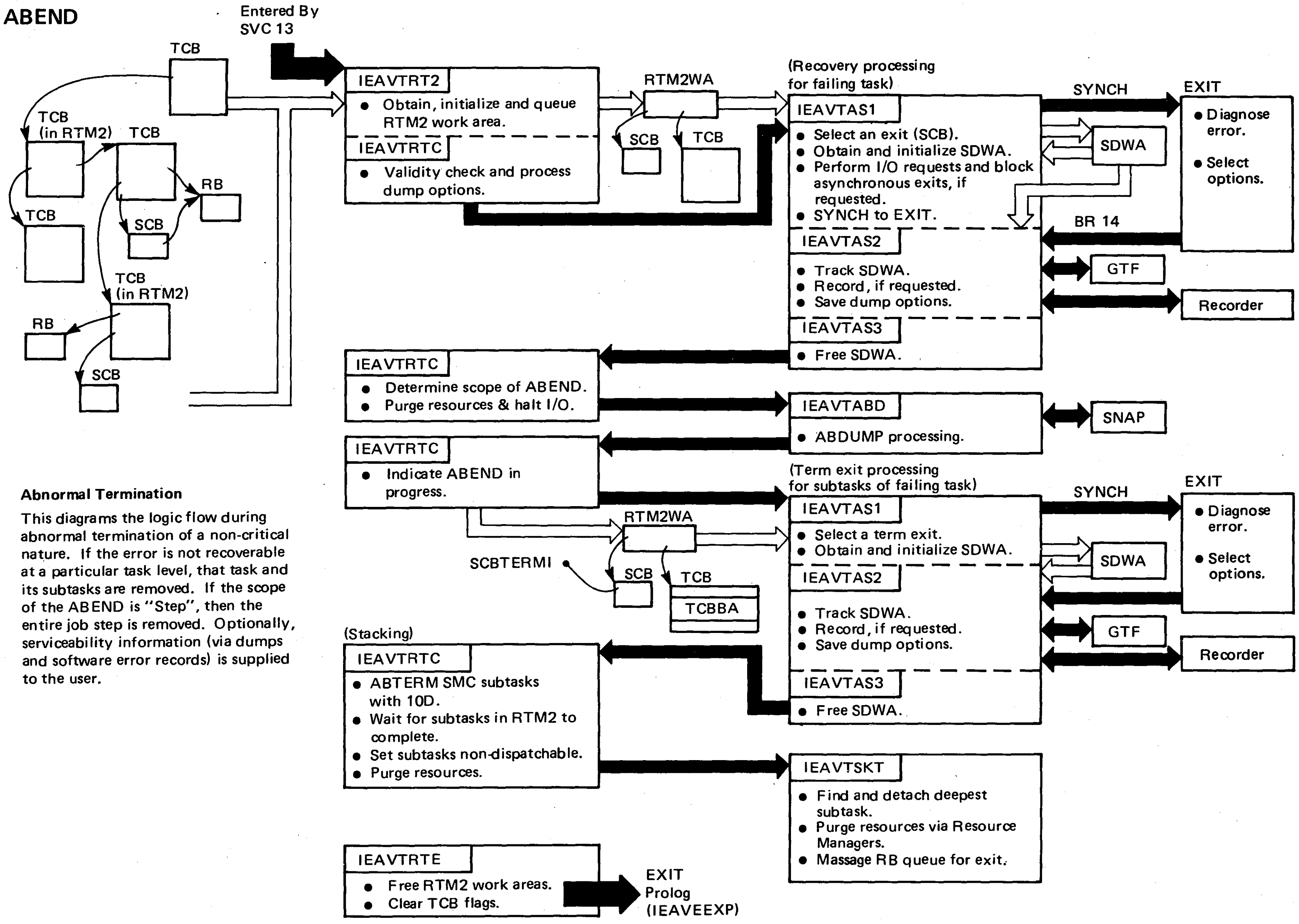

(From Part 1)
1
BRANCH
IGC003 – EXIT Prolog
Prolog deletes SVRB
If address space termination is necessary – go to Figure 7.
DISP
If Task Termination, redispatch of task causes EXIT to receive control again
TCB
TCBOET=1
PRB
Resume PSW
SVC 3
IGC003 – EXIT
1 Since the end-of-task indicator has been set (TCBEOT) BALR to Resource Manager for cleanup of Task.
1 TRRM
2 VSM
3 PGM
4 DET
2 Exit to dispatcher (IEAVEDSO)
IEAQSPET–IEAVGCAS
Free Storage
BALR
IGC062R1–IEAVEEDO
Free TCB & RB core.
Deque TCB.
EITHER
Schedule end-of-task Exit Routine for task
OR
Post mother task if attached w/ECB operand.
BALR
IEAVTSBP
Dequeue/Free SCB's owned by RB or Task.
BALR
IEAPPGMX
Free Programs
BALR
BR 14
Normal Task Termination is Complete

Figure 2-48. The Process of Normal Task Termination (Part 2 of 2)

ABEND



Abnormal Termination

This diagrams the logic flow during abnormal termination of a non-critical nature. If the error is not recoverable at a particular task level, that task and its subtasks are removed. If the scope of the ABEND is "Step", then the entire job step is removed. Optionally, serviceability information (via dumps and software error records) is supplied to the user.

Figure 2-49. Abnormal End-of-Task

This shows the flow through RTM2 when processing a potentially recoverable error. The recovery exit is supplied environmental data that describes the error, for example, completion code, register contents, PSW, system state at time of error, etc., to aid in diagnosing the error. To effect retry, the resume PSW in each RB up to and including the retry RB is modified. The retry address supplied by the exit is placed in resume PSW field of the retrying RB, and all RB's between the retry RB and the RTM2 RB have their resume PSW set to either Exit prologue or SVC 3. When RTM2 eventually returns to the system, supervisor assisted linkage will cause the retry address in the retry RB to be given control.

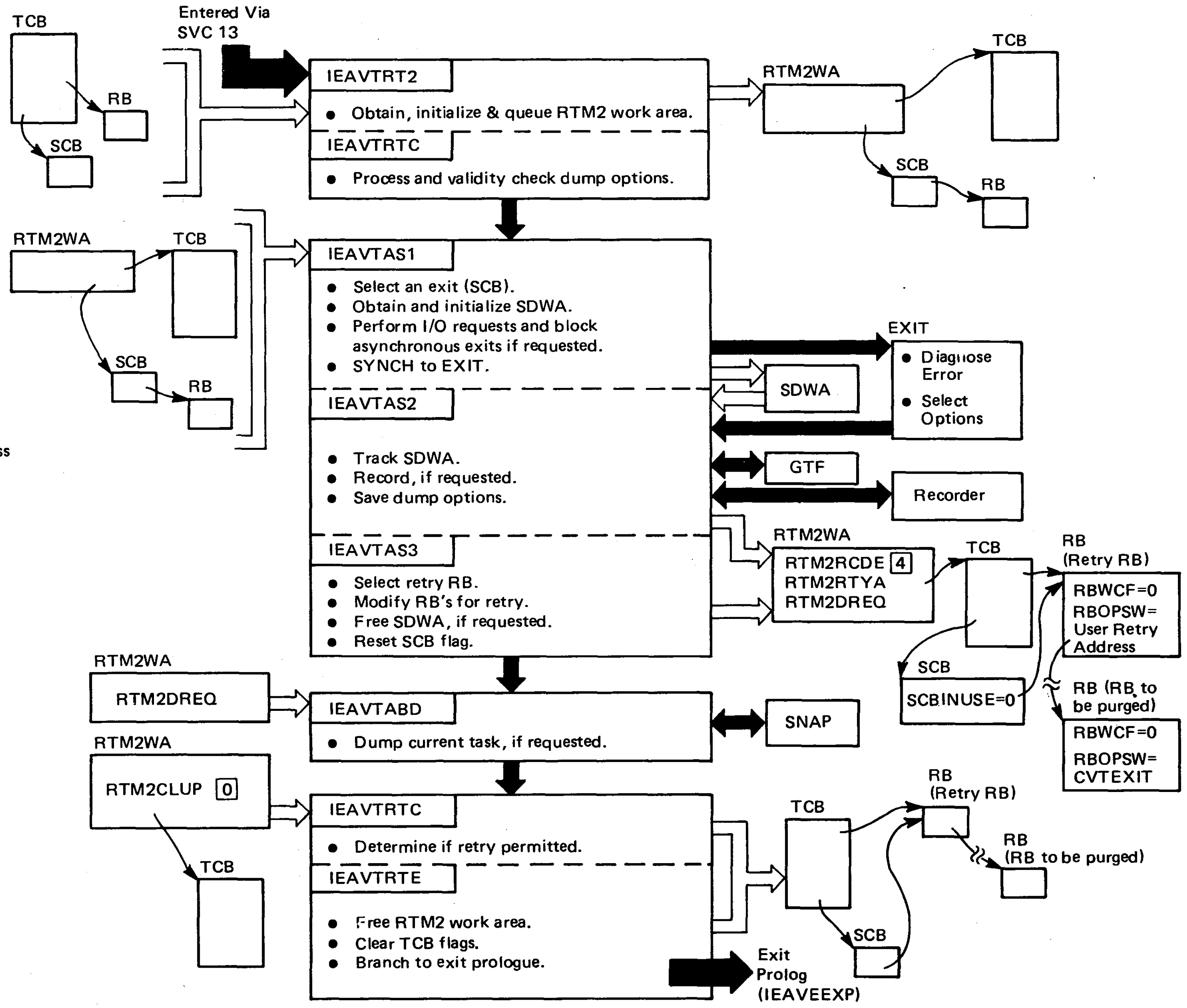
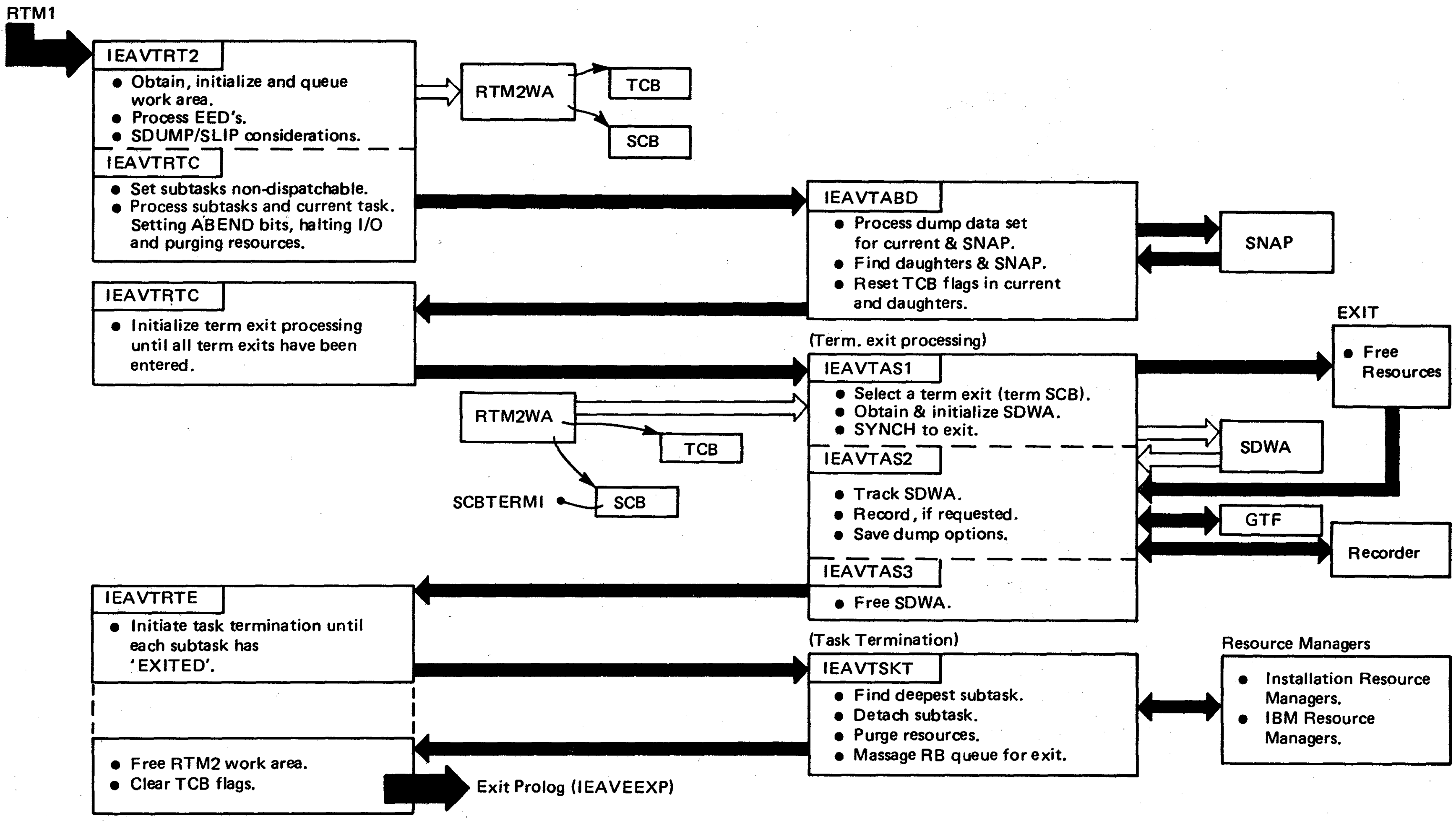


Figure 2-50. Retry


RTM1
IEAVTRT2
● Obtain, initialize and queue work area.
● Process EED's.
● SDUMP/SLIP considerations.
IEAVTRTC
● Set subtasks non-dispatchable.
● Process subtasks and current task. Setting ABEND bits, halting I/O and purging resources.
RTM2WA
TCB
SCB
IEAVTABD
● Process dump data set for current & SNAP.
● Find daughters & SNAP.
● Reset TCB flags in current and daughters.
SNAP
IEAVTRTC
● Initialize term exit processing until all term exits have been entered.
(Term. exit processing)
IEAVTAS1
● Select a term exit (term SCB).
● Obtain & initialize SDWA.
● SYNCH to exit.
EXIT
● Free Resources
RTM2WA
TCB
SCBTERMI
SCB
SDWA
IEAVTAS2
● Track SDWA.
● Record, if requested.
● Save dump options.
GTF
Recorder
IEAVTAS3
● Free SDWA.
IEAVTRTE
● Initiate task termination until each subtask has 'EXITED'.
(Task Termination)
IEAVTSKT
● Find deepest subtask.
● Detach subtask.
● Purge resources.
● Massage RB queue for exit.
Resource Managers
● Installation Resource Managers.
● IBM Resource Managers.
● Free RTM2 work area.
● Clear TCB flags.
Exit Prolog (IEAVEEXP)

This illustrates the flow of control through R/TM when a job is cancelled. The CANCEL request is indicated by specific completion codes set in the TCB by RTM1 (code='x22'). The CANCEL process is distinctive in that it is considered a strictly unrecoverable situation. Normal termination procedures are abandoned in favor of creating an 'express' path through termination. However, term exits are given control.

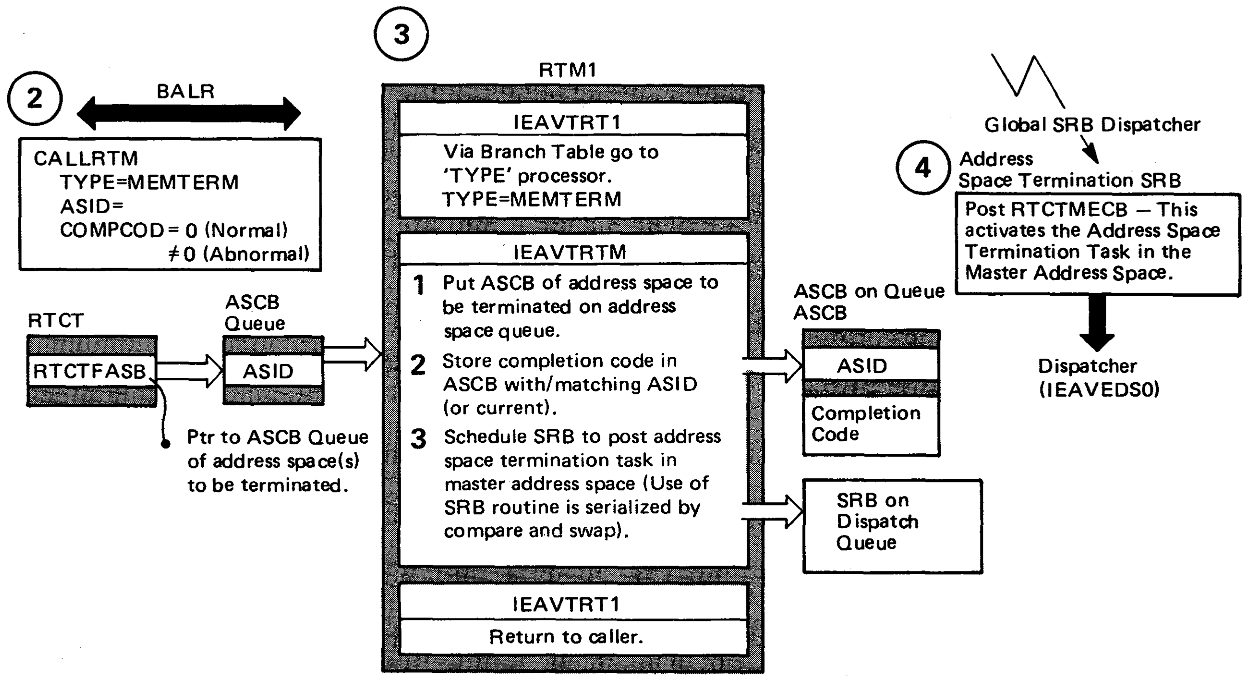
Figure 2-51. Cancel

1

Since the MEMTERM process circumvents all TASK recovery and TASK Resource Manager processing, its use is restricted to a select group of routines which can determine that task recovery and resource manager clean up is either not warranted or will not successfully operate in the address space being terminated. It therefore is restricted to the following users:

1) Paging supervisor when it determines that it cannot swap in the LSQA for an address space,
2) Address space create when it determines that an Address Space cannot be initialized,
3) The RTM or the supervisor control FRR when they determine that uncorrectable translation errors are occuring in the address space,
4) The RTM2 when it determines that task recovery and termination cannot take place in the current address space;
5) The RCT when it determines that the address space is permanently deadlocked,
6) The RTM2 when all tasks in the address space have terminated (IEAVTRTE). This is the only requestor of normal address space termination (i.e. COMPCOD=0).
7) Auxiliary Storage Management recovery routine, when it suffers an indeterminate error from which it cannot recover, while handling a swap-in or a swap-out request.
8) Auxiliary Storage Management recovery routine, when it determines that uncorrectable translation errors are occurring while ASM is using the control register of another address space to update that address space's LSQA.

Note: Since callers 4, 5, and 6 above are task-related and running in the address space to be terminated, they will set themselves non-dispatchable after issuance of CALLRTM.



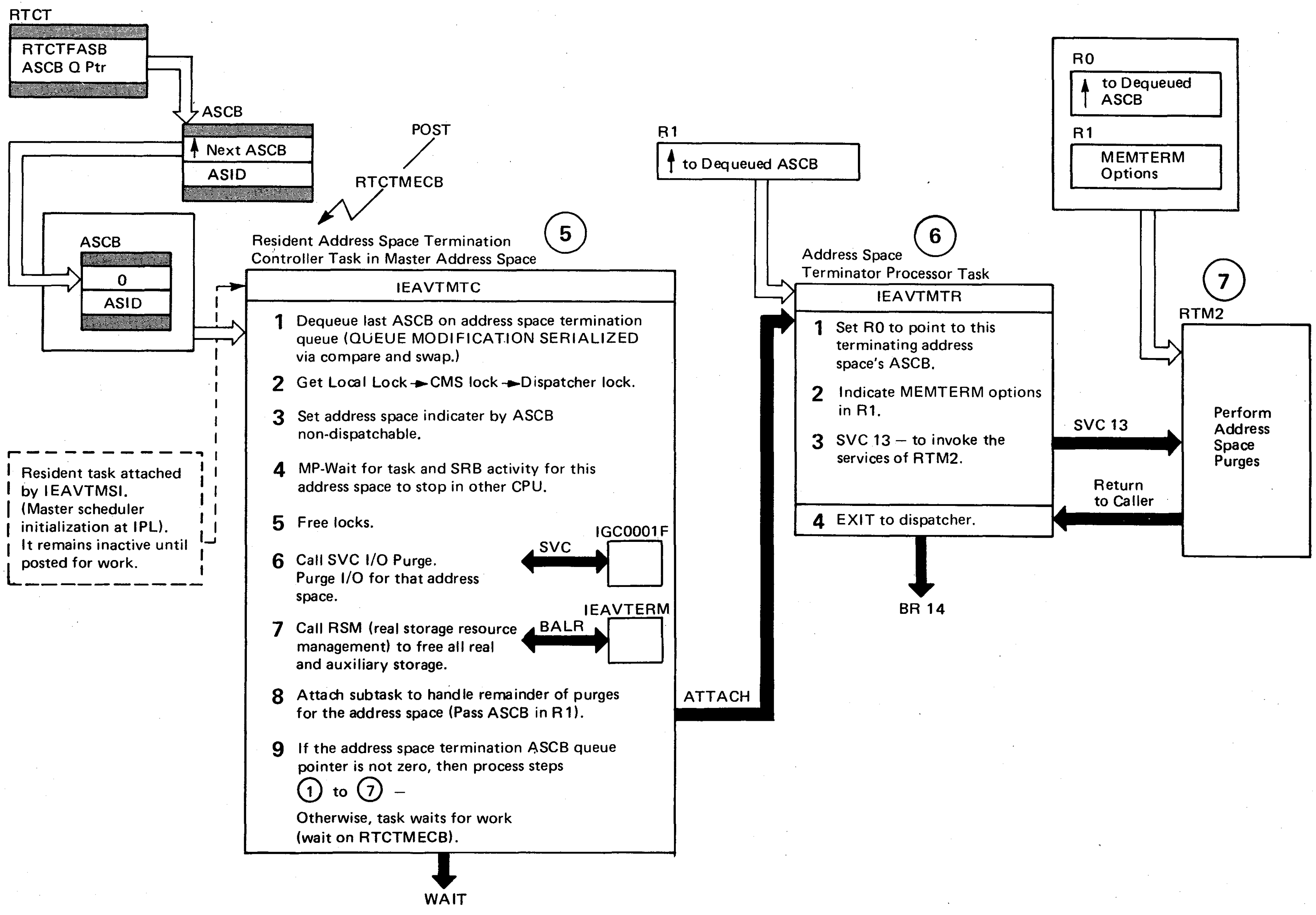
2
BALR
CALLRTM
TYPE=MEMTERM
ASID=
COMPCOD= 0 (Normal)
≠0 (Abnormal)
RTCT
RTCTFASB
ASCB Queue
ASID
Ptr to ASCB Queue of address space(s) to be terminated.
3
RTM1
IEAVTRT1
Via Branch Table go to 'TYPE' processor. TYPE=MEMTERM
IEAVTRTM
1 Put ASCB of address space to be terminated on address space queue.
2 Store completion code in ASCB with/matching ASID (or current).
3 Schedule SRB to post address space termination task in master address space (Use of SRB routine is serialized by compare and swap).
IEAVTRT1
Return to caller.
ASCB on Queue ASCB
ASID
Completion Code
SRB on Dispatch Queue
Global SRB Dispatcher
4
Address Space Termination SRB
Post RTCTMECB – This activates the Address Space Termination Task in the Master Address Space.
Dispatcher (IEAVEDSO)

The process of terminating an address space (memory) is one which cannot be isolated to one task, module or logical unit of code. This presents the many parts of the function into a coherent picture of the process, by showing control flow and data flow.

The multiple dispatches, tasks, and address spaces involved would otherwise be hidden elements.

Step 1	Identifies the Requesters
Step 2	The Request Format
Steps 3, 4	Initiate the Request
Steps 5, 6, 7	Process the Request

Figure 2-52. The Process of Terminating an Address Space (Part 1 of 2)



Resident task attached by IEAVTMSI. (Master scheduler initialization at IPL). It remains inactive until posted for work.

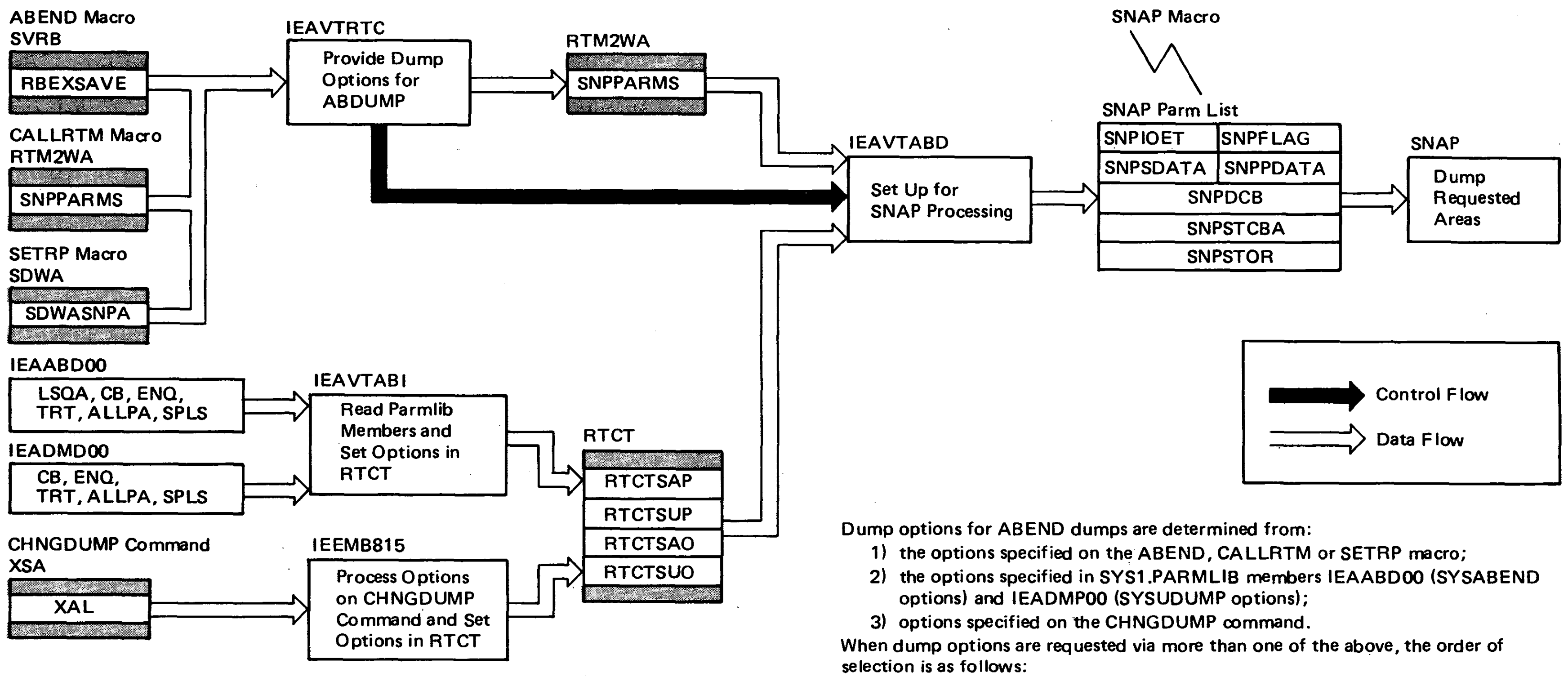
Resident Address Space Termination Controller Task in Master Address Space

1. Dequeue last ASCB on address space termination queue (QUEUE MODIFICATION SERIALIZED via compare and swap.)
2. Get Local Lock → CMS lock → Dispatcher lock.
3. Set address space indicater by ASCB non-dispatchable.
4. MP-Wait for task and SRB activity for this address space to stop in other CPU.
5. Free locks.
6. Call SVC I/O Purge. Purge I/O for that address space.
7. Call RSM (real storage resource management) to free all real and auxiliary storage.
8. Attach subtask to handle remainder of purges for the address space (Pass ASCB in R1).
9. If the address space termination ASCB queue pointer is not zero, then process steps ① to ⑦ – Otherwise, task waits for work (wait on RTCTMECB).

Address Space Terminator Processor Task

1. Set R0 to point to this terminating address space's ASCB.
2. Indicate MEMTERM options in R1.
3. SVC 13 – to invoke the services of RTM2.
4. EXIT to dispatcher.

Figure 2-52. The Process of Terminating an Address Space (Part 2 of 2)



This provides an overview of all data areas related to ABEND/SNAP dumps, the sources from which the dump options are obtained, the key modules involved and the complete scheme of data flow. It ties together the function of system initialization requestors dump options and operator intervention as all parts of the process.

Figure 2-53. ABEND/SNAP Dump Processing

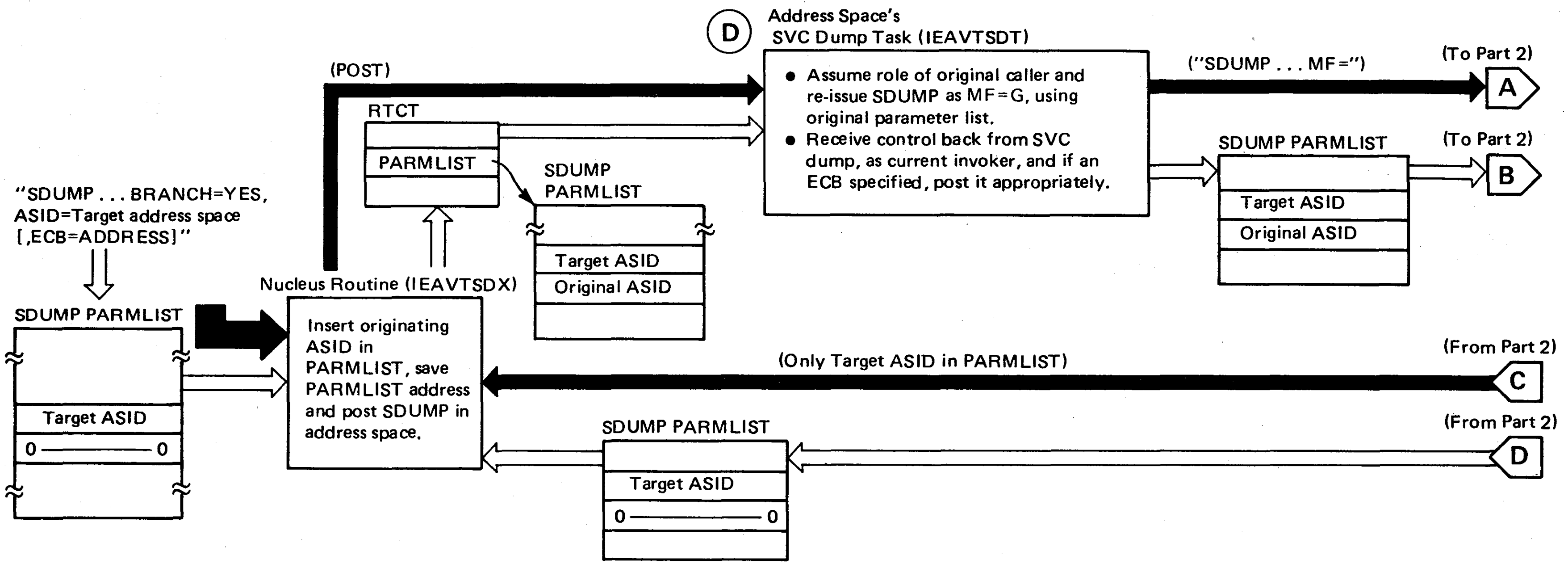
Dump options for ABEND dumps are determined from:

1) the options specified on the ABEND, CALLRTM or SETRP macro;
2) the options specified in SYS1.PARMLIB members IEAABD00 (SYSABEND options) and IEADMP00 (SYSUDUMP options);
3) options specified on the CHNGDUMP command.

When dump options are requested via more than one of the above, the order of selection is as follows:

1. CHNGDUMP options completely override any other request.
2. Lacking CHNGDUMP options, the options specified on the ABEND, CALLRTM or SETRP macros are merged with the options in IEAABD00 or IEADMP00.
3. If no options were specified on the ABEND, CALLRTM or SETRP macros, the options specified in IEAABD00 or IEADMP00 will be used. If no options are specified via CHNGDUMP, ABEND, CALLRTM, SETRP, IEAABD00 or IEADMP00 no dump will be taken.

For ABEND dumps the requestor (via ABEND, CALLRTM, and SETRP) and installation (via SYS1.PARMLIB members IEAABD00 and IEADMP00) have been given the ability to tailor dumps to the needs of the installation and the individual maintenance requirements of each type error. In addition, the CHNGDUMP command provides the facility to temporarily override options specified by the requestor and/or installation.



"SDUMP . . . BRANCH=YES, ASID=Target address space [,ECB=ADDRESS]"
SDUMP PARMLIST
Target ASID
0 —— 0
Nucleus Routine (IEAVTSDX)
Insert originating ASID in PARMLIST, save PARMLIST address and post SDUMP in address space.
(POST)
RTCT
PARMLIST
SDUMP PARMLIST
Target ASID
Original ASID
D
Address Space's SVC Dump Task (IEAVTSDT)
● Assume role of original caller and re-issue SDUMP as MF=G, using original parameter list.
● Receive control back from SVC dump, as current invoker, and if an ECB specified, post it appropriately.
("SDUMP . . . MF=")
(To Part 2)
A
SDUMP PARMLIST
Target ASID
Original ASID
(To Part 2)
B
(From Part 2)
C
(Only Target ASID in PARMLIST)
(From Part 2)
D
SDUMP PARMLIST
Target ASID
0 —— 0

This ties together system initialization, option modification by operator, multiple address space processing, and multiple tasks as parts of the SVC DUMP process. The process of how a dump is initiated for a task in a different address space from the requester, is explained by control flow and data flow.

A IEAVTSDI initializes the SVC dump data set table, and locates the SVC dump resource manager, IEAVTSDR, for use during address space termination.

B The presence of an "ASID=" parameter signifies a request to display a specific address space, and as such requires scheduling the dump request to the SVC dump task in that address space. If the parameter list contains only the address space's ASID, it indicates that scheduling has not yet taken place and so entry is made to the nucleus routine. This routine places the originating address space's ASID in the parameter list and schedules the appropriate SVC dump task, passing the parameter list (which now contains two ASID's). When the SVC dump task gains control it re-issues the SDUMP macro, using the original parameter list it received as input. When SVC dump is re-entered it realizes that scheduling has already taken place (two ASID's are present) and that the dump can now be performed. Note that the "caller" of SVC dump in this instance is the SVC dump task, not the original requestor. When SVC dump completes, it will inform the original requester of the dump's completion if an ECB was

B (continued)

supplied. Otherwise the SVC dump task in that address space bypasses ECB posting and simply returns to a wait-for-work condition.

C If the "ASID=" parameter was not specified (and therefore not present in the parameter list), the "Caller" is the original invoker and SVC dump will run under the TCB of the caller. No scheduling of the dump is performed. Likewise, no scheduling is done if both ASID's are present in the parameter list.

D The SVC dump task IEAVTSDT in each address space is attached by the RCT, except in the master address space, where it is attached by IEAVTMSI (master scheduler R/TM initialization routine).

E If CHNGDUMP dump option overrides exist, they will be used exclusively.

Figure 2-54. SVC Dump Overview (Part 1 of 2)

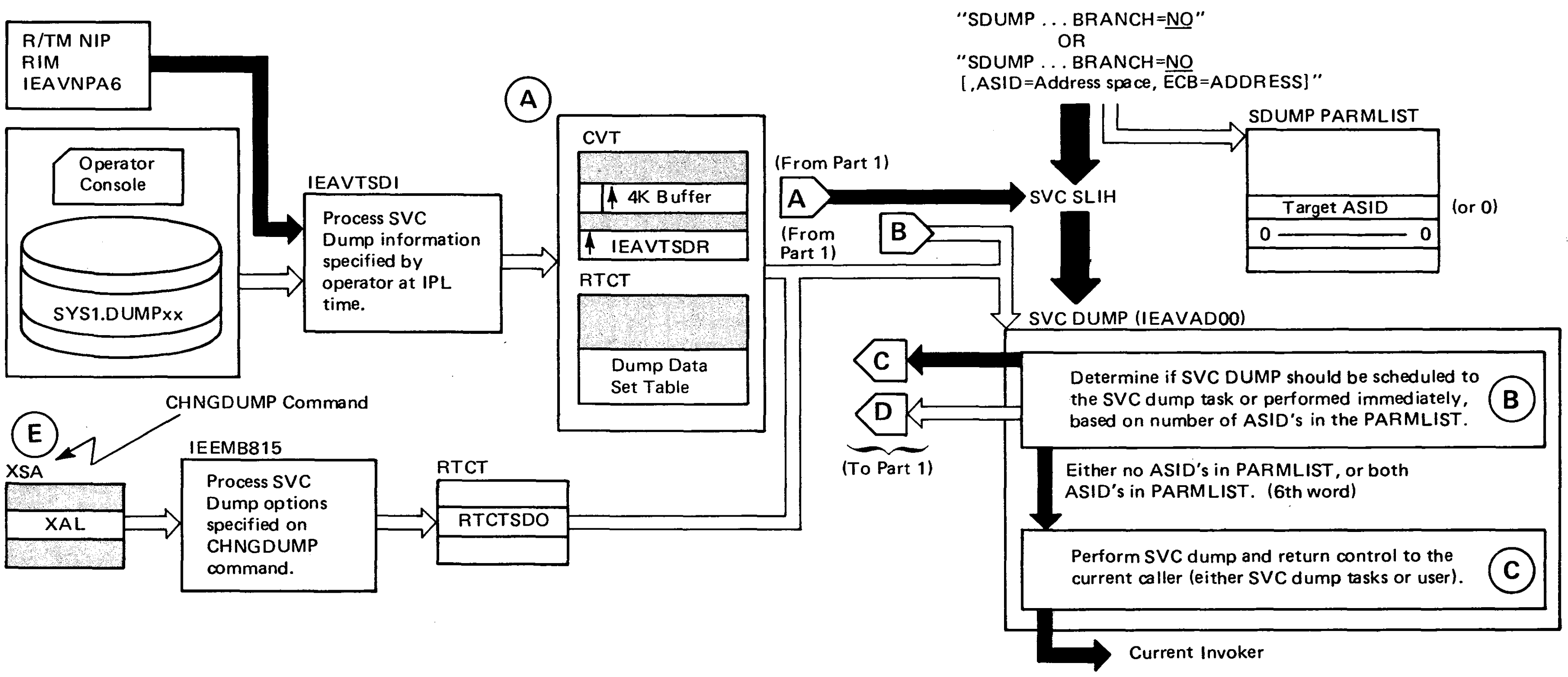

"SDUMP . . . BRANCH=NO"
OR
"SDUMP . . . BRANCH=NO
[,ASID=Address space, ECB=ADDRESS]"
SDUMP PARMLIST
Target ASID
(or 0)
0
0
R/TM NIP
RIM
IEAVNPA6
A
CVT
4K Buffer
IEAVTSDR
RTCT
Dump Data Set Table
(From Part 1)
A
SVC SLIH
B
(From Part 1)
Operator Console
SYS1.DUMPxx
IEAVTSDI
Process SVC Dump information specified by operator at IPL time.
SVC DUMP (IEAVAD00)
C
D
(To Part 1)
Determine if SVC DUMP should be scheduled to the SVC dump task or performed immediately, based on number of ASID's in the PARMLIST.
B
Either no ASID's in PARMLIST, or both ASID's in PARMLIST. (6th word)
Perform SVC dump and return control to the current caller (either SVC dump tasks or user).
C
Current Invoker
CHNGDUMP Command
E
XSA
XAL
IEEMB815
Process SVC Dump options specified on CHNGDUMP command.
RTCT
RTCTSDO

Figure 2-54. SVC Dump Overview (Part 2 of 2)

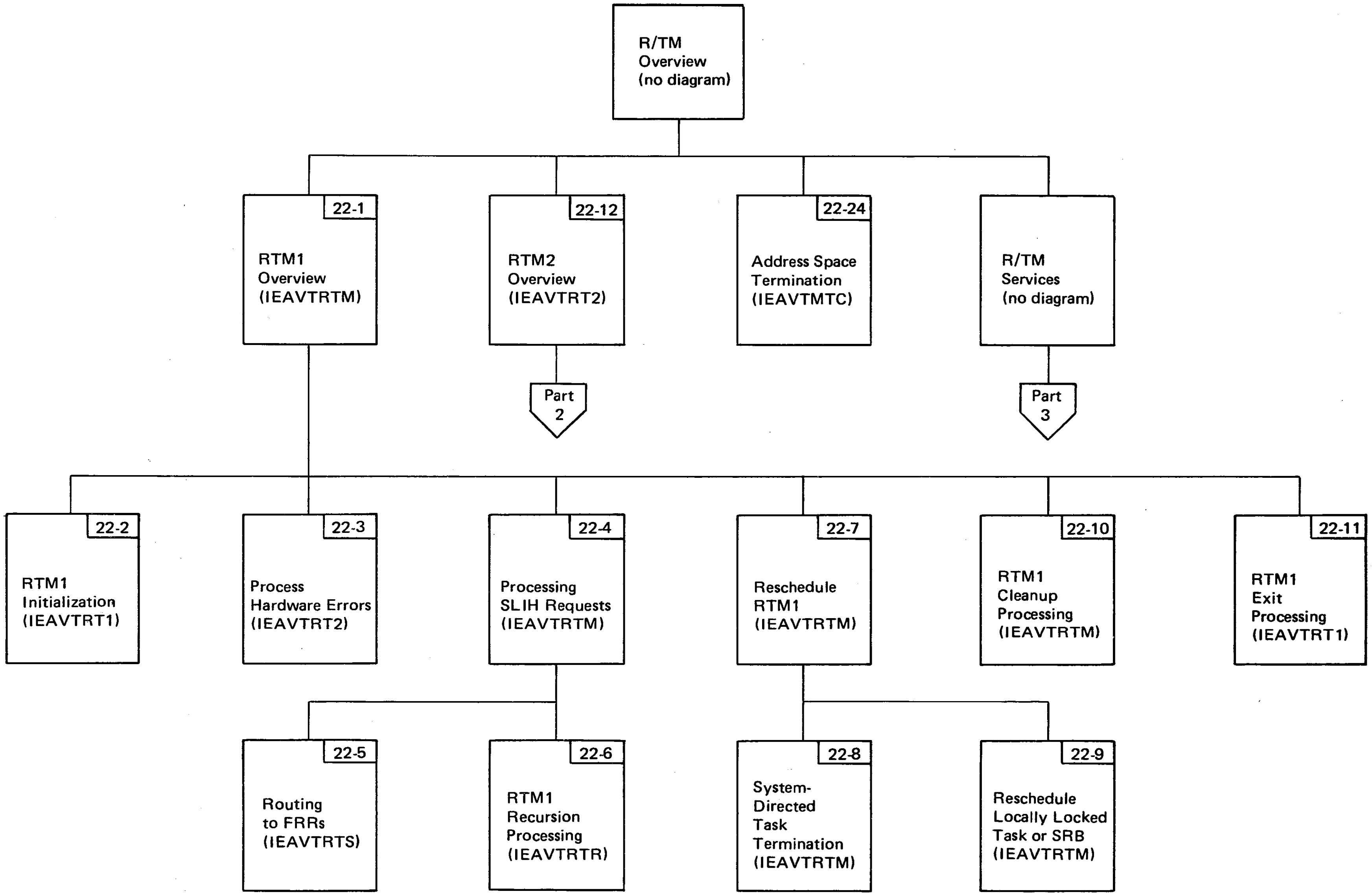

R/TM Overview (no diagram)
22-1 RTM1 Overview (IEAVTRTM)
22-12 RTM2 Overview (IEAVTRT2)
22-24 Address Space Termination (IEAVTMTC)
R/TM Services (no diagram)
Part 2
Part 3
22-2 RTM1 Initialization (IEAVTRT1)
22-3 Process Hardware Errors (IEAVTRT2)
22-4 Processing SLIH Requests (IEAVTRTM)
22-7 Reschedule RTM1 (IEAVTRTM)
22-10 RTM1 Cleanup Processing (IEAVTRTM)
22-11 RTM1 Exit Processing (IEAVTRT1)
22-5 Routing to FRRs (IEAVTRTS)
22-6 RTM1 Recursion Processing (IEAVTRTR)
22-8 System-Directed Task Termination (IEAVTRTM)
22-9 Reschedule Locally Locked Task or SRB (IEAVTRTM)

Figure 2-55. Recovery/Termination Management Visual Contents (Part 1 of 3)

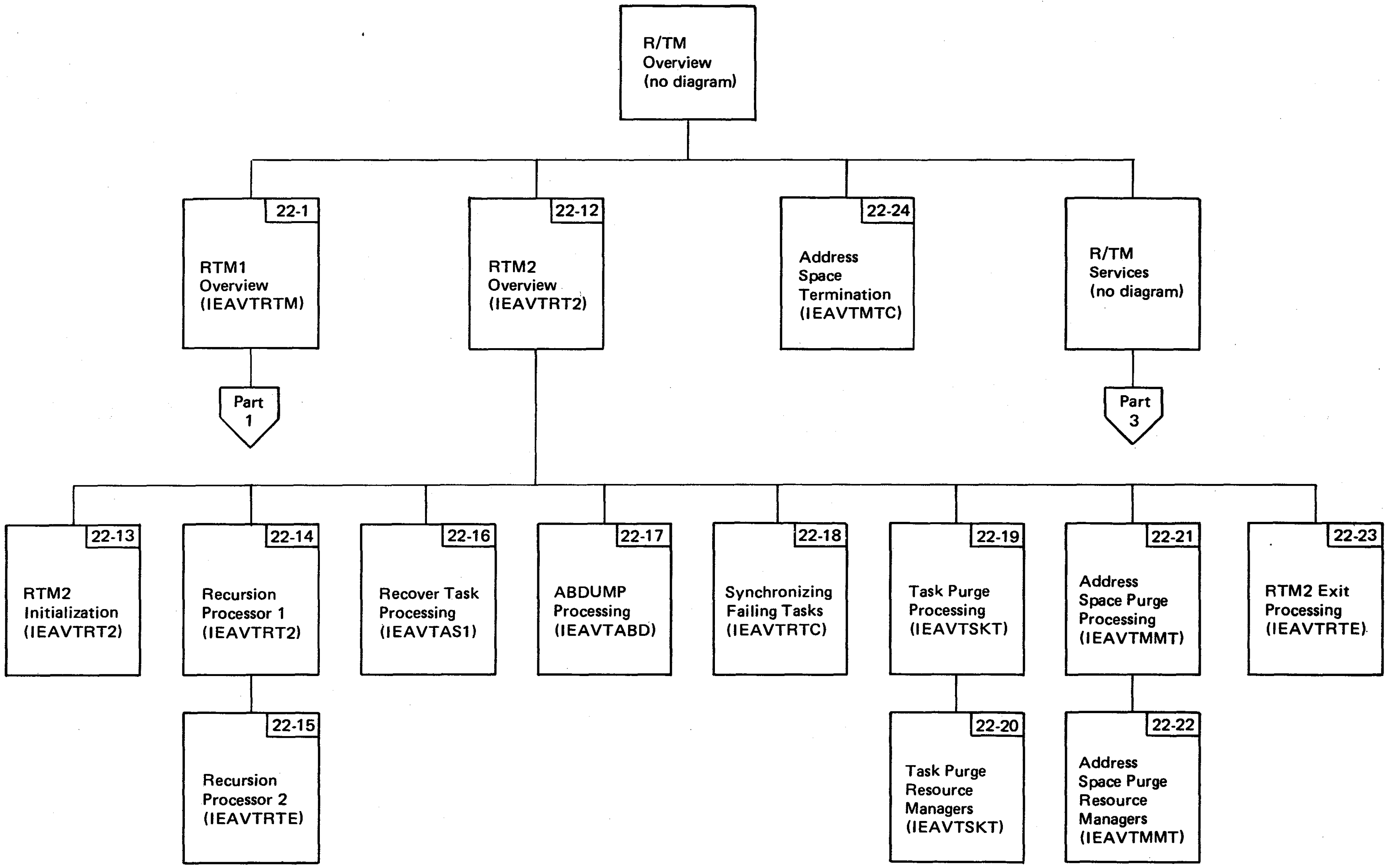
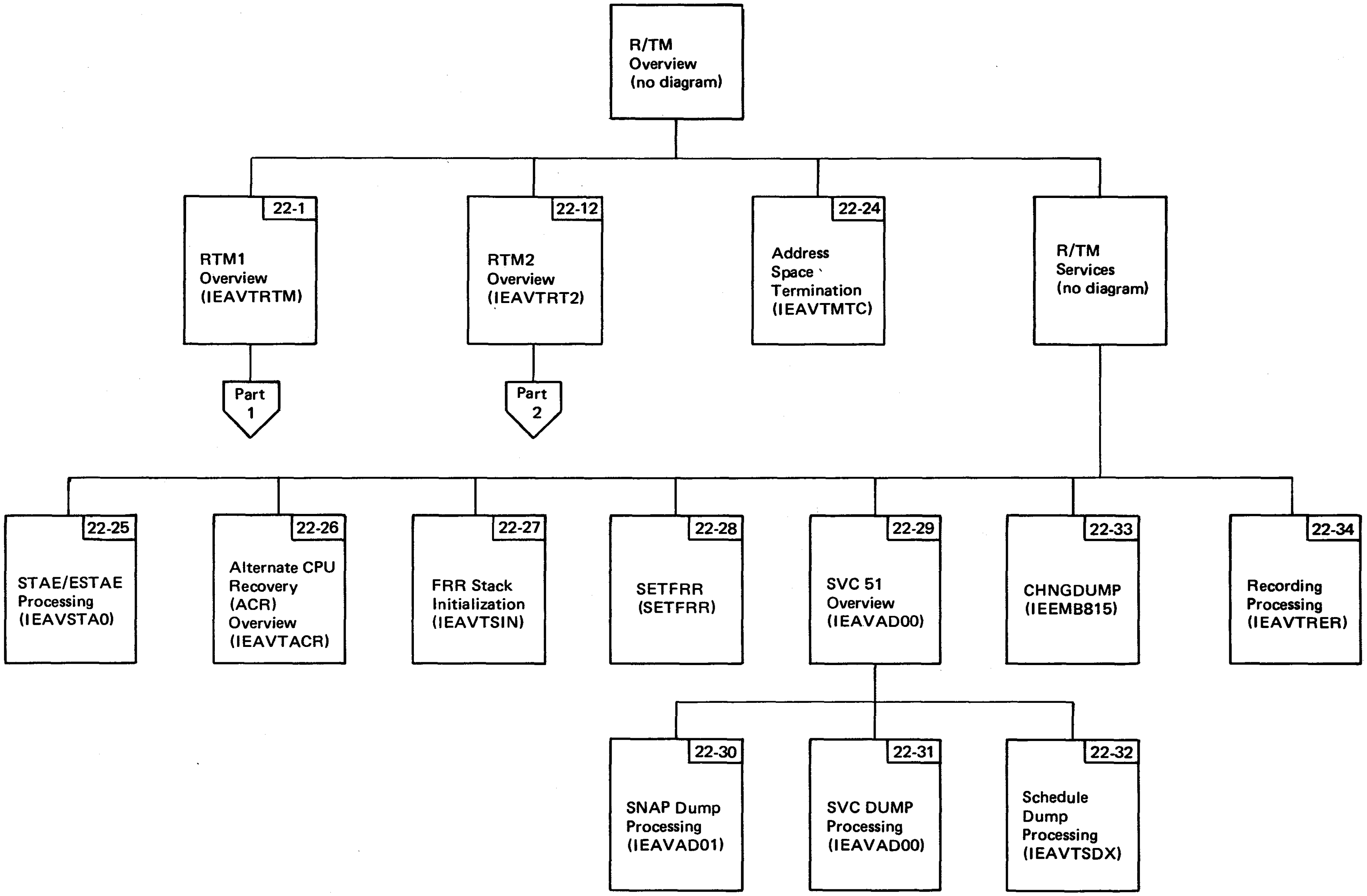

R/TM Overview (no diagram)
22-1
RTM1 Overview (IEAVTRTM)
Part 1
22-12
RTM2 Overview (IEAVTRT2)
22-24
Address Space Termination (IEAVTMTC)
R/TM Services (no diagram)
Part 3
22-13
RTM2 Initialization (IEAVTRT2)
22-14
Recursion Processor 1 (IEAVTRT2)
22-15
Recursion Processor 2 (IEAVTRTE)
22-16
Recover Task Processing (IEAVTAS1)
22-17
ABDUMP Processing (IEAVTABD)
22-18
Synchronizing Failing Tasks (IEAVTRTC)
22-19
Task Purge Processing (IEAVTSKT)
22-20
Task Purge Resource Managers (IEAVTSKT)
22-21
Address Space Purge Processing (IEAVTMMT)
22-22
Address Space Purge Resource Managers (IEAVTMMT)
22-23
RTM2 Exit Processing (IEAVTRTE)

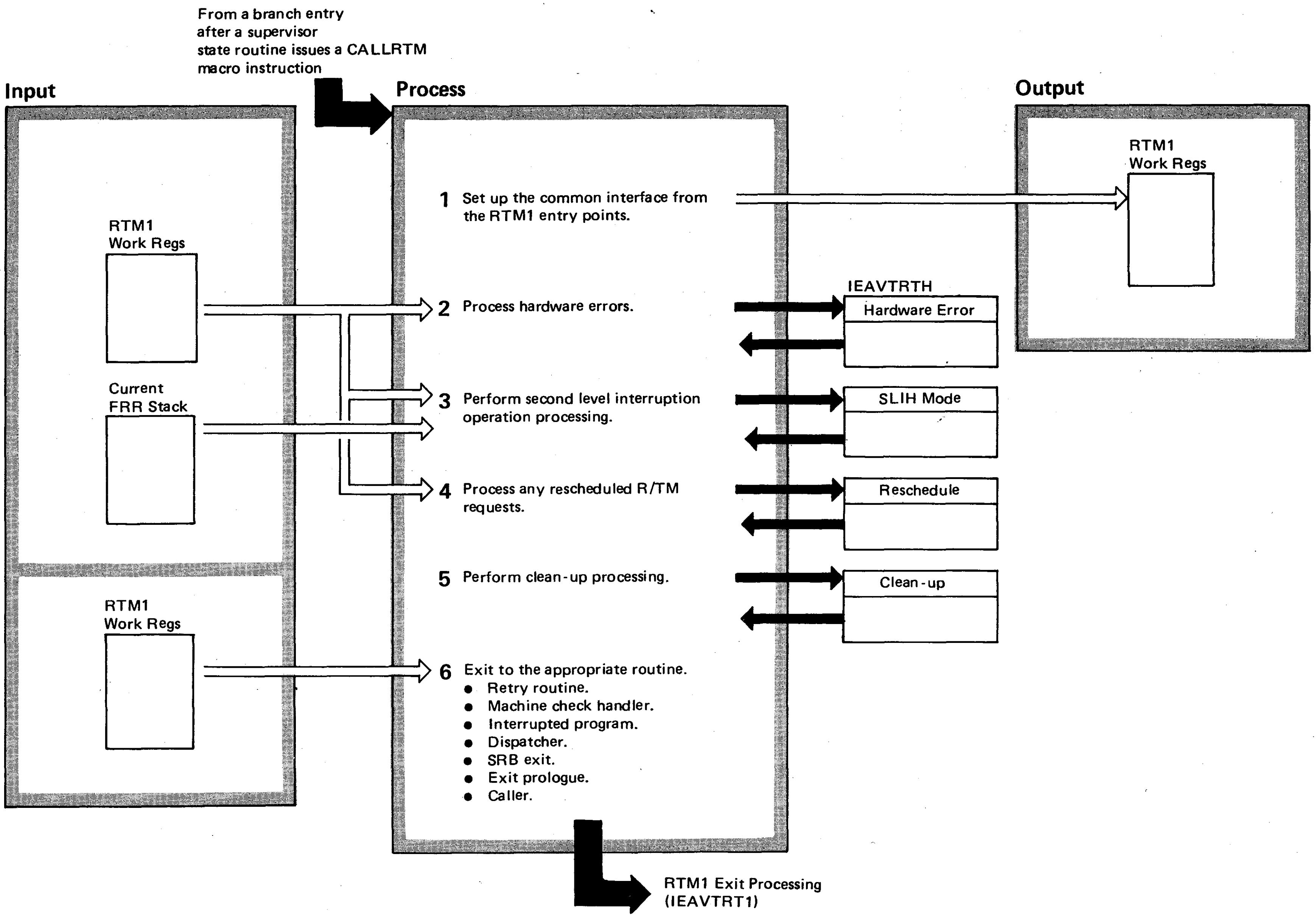
Figure 2-55. Recovery/Termination Management Visual Contents (Part 2 of 3)



R/TM Overview (no diagram)
22-1
RTM1 Overview (IEAVTRTM)
Part 1
22-12
RTM2 Overview (IEAVTRT2)
Part 2
22-24
Address Space Termination (IEAVTMTC)
R/TM Services (no diagram)
22-25
STAE/ESTAE Processing (IEAVSTA0)
22-26
Alternate CPU Recovery (ACR) Overview (IEAVTACR)
22-27
FRR Stack Initialization (IEAVTSIN)
22-28
SETFRR (SETFRR)
22-29
SVC 51 Overview (IEAVAD00)
22-33
CHNGDUMP (IEEMB815)
22-34
Recording Processing (IEAVTRER)
22-30
SNAP Dump Processing (IEAVAD01)
22-31
SVC DUMP Processing (IEAVAD00)
22-32
Schedule Dump Processing (IEAVTSDX)

Figure 2-55. Recovery/Termination Management Visual Contents (Part 3 of 3)

Diagram 22-1. RTM1 Overview (IEAVTRTM) (Part 1 of 2)



From a branch entry after a supervisor state routine issues a CALLRTM macro instruction
Input
RTM1 Work Regs
Current FRR Stack
RTM1 Work Regs
Process
1 Set up the common interface from the RTM1 entry points.
2 Process hardware errors.
3 Perform second level interruption operation processing.
4 Process any rescheduled R/TM requests.
5 Perform clean-up processing.
6 Exit to the appropriate routine.
● Retry routine.
● Machine check handler.
● Interrupted program.
● Dispatcher.
● SRB exit.
● Exit prologue.
● Caller.
IEAVTRTH
Hardware Error
SLIH Mode
Reschedule
Clean-up
Output
RTM1 Work Regs
RTM1 Exit Processing (IEAVTRT1)

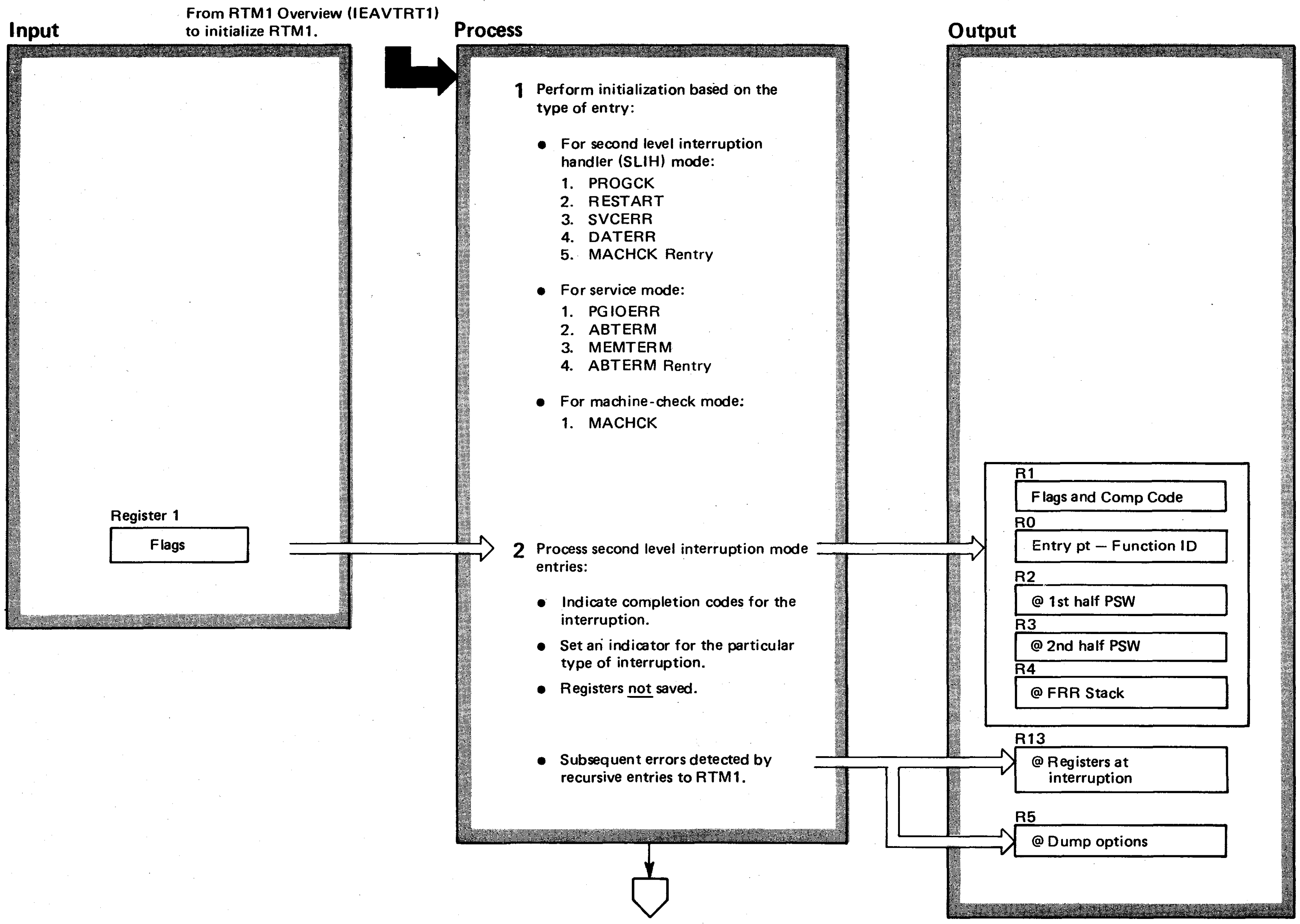
Diagram 22-1. RTM1 Overview (IEAVTRTM) (Part 2 of 2)

Extended Description	Module	Segment
The RTM1 service of recovery termination management (R/TM) provides a recovery interface with other supervisory routines. When a supervisor routine – principally the interruption handlers – detects an error situation, it passes control to RTM1, via the CALLRTM macro instruction, to initiate recovery from the error. RTM1 records the error – both hardware and software – to SYS1.LOGREC via the recording service.		
RTM1 does not perform the recovery function itself; it routes control to functional recovery routines (FRRs) established by locked, disabled or SRB routines. These FRRs are placed on an LIFO FRR "stack" by a SETFRR macro instruction issued by the routine requesting protection. The macro expansion places the FRRs on predefined stacks, that is, the FRR is placed on an appropriate stack based on its functional path through the supervisor (however, the "Super" FRR is placed on *each* stack by NIP processing). The following list shows the stacks:		
• SVC-I/O-dispatcher stack • Machine check stack • Program check stack • External interruption handler 1 stack • External interruption handler 2 stack • External interruption handler 3 stack • Restart interruption handler stack		
Additionally, a normal FRR stack contains the recovery status for other paths through the system.		
RTM1 receives control for 12 reasons. These are for:		
• Program checks. • Restart operations. • SVC errors. • Page I/O errors. • Machine checks. • DAT (dynamic address translation) errors. • Abnormal termination (ABTERM) requests for a task with an ASID (address space identifier) specified.		

*Module IEAVTRT1 contains labels; the column under "Segment" refers to label names.

Extended Description	Module	Segment
• Abnormal termination requests for a task in the current address space. • Address space termination requests. • Reentry for abnormal termination requests. • Reentry for machine checks. • Branch entries for abnormal termination requests.		
1 RTM1 creates a common interface for its sub-functions from the various entry point data and establishes recursion control for service routine requests.	IEAVTRT1	
2 When either MCH (machine check handler) or ACR (alternate CPU recovery) indicates a hardware error, control goes unconditionally to the hardware repair function, module IEAVTRTH (see M.O. diagram, Processing Hardware Errors (IEAVTRT2)). Hardware repair performs software repair, if necessary, and attempts to record all hardware errors on SYS1.LOGREC (module IEAVTRTM).	IEAVTRTM	
3 The program check IH (interruption handler), SVC IH, the restart IH, and the machine check handler (MCH) all can request RTM1 to perform second level interruption handler processing (SLIH mode). When RTM1 processes an SLIH mode entry type (that is, TYPE=PCFLIH, MACHCK reentry, SVCERR, RESTART, DATERR) it continues the processing of the interruption. SLIH mode functioning determines the state of the system at the time of the interruption, so that recovery from the interruption may be attempted in either system mode or task mode.		
4 RTM1 performs reschedule processing for a service routine entry (that is, the CALLRTM request was for ABTERM, MEMTERM, or PGIOERR). The reschedule function may also be performed as part of SLIH mode processing. This would occur if the action indicated by routing to FRRs required a reschedule service of if the CPU had been in task mode when the error interruption occurred.		
5 The clean up function frees any resources no longer necessary before determining the appropriate type of exit.		
6 RTM1 creates the final exit linkage based on an indicator established in IEAVTRTM.	IEAVTRT1	

Diagram 22-2. RTM1 Initialization (IEAVTRT1) (Part 1 of 4)



1 Perform initialization based on the type of entry:

- For second level interruption handler (SLIH) mode:
  1. PROGCK
  2. RESTART
  3. SVCERR
  4. DATERR
  5. MACHCK Rentry
- For service mode:
  1. PGIOERR
  2. ABTERM
  3. MEMTERM
  4. ABTERM Rentry
- For machine-check mode:
  1. MACHCK

2 Process second level interruption mode entries:

- Indicate completion codes for the interruption.
- Set an indicator for the particular type of interruption.
- Registers not saved.
- Subsequent errors detected by recursive entries to RTM1.

Diagram 22-2. RTM1 Initialization (IEAVTRT1) (Part 2 of 4)

Extended Description	Module	Segment
RTM1 processing receives control via the CALLRTM macro instruction. The expansion of this macro instruction locates the correct entry point address into RTM1 from the RTM1 branch table (pointed to by the CVTBTERM field of the CVT). RTM1 initialization combines the various entry point data to create a common interface for RTM1 processing.		
1 RTM1 initialization consists of saving registers, indicating completion codes, and establishing a recovery environment based upon the type of entry. RTM1 performs three types of initialization; one based on requests made by the interruption handlers; another based on a service request for an RTM1 service; and the last for machine check interruptions.	IEAVTRT1	
2 RTM1 initialization prepares the following entry points for SLIH mode:		
1. **Program check entry point** — used by the program check IH when an invalid page fault or program check occurs. When the program check IH passes RTM1 a completion code, the registers and PSW have been saved by the program check IH in the primary save areas of the PSA and LCCA (logical configuration communications area). When RTM1 does not receive a completion code, initialization processing builds one from the interruption code and the error information in the secondary save area in the LCCA. (The "Supervisor Control" section describes the program check IH, and the different save areas used.)		PROGCK
2. **Restart entry point** — used by the restart IH after the operator has requested R/TM processing. The subsequent handling of a restart request in R/TM is tailored to "loop breaking" logic, that is, a looping program cannot be allowed to retry, and a validly spinning program is allowed to request R/TM to interrupt the program that owns the resource being waited upon. The restart IH has saved the registers in the LCCA and the resume PSW in the PSA.		RESTART
3. **SVC IH entry point** — used whenever an SVC is issued by a routine that is locked, in SRB mode, or is under supervisor control (non-dispatchable supervisory functions). If the SVC was an SVC 13, RTM1 interprets the entry point as an explicit request for ABEND processing. RTM1 interprets entry from any other SVC to be an error. The SVC IH has saved the registers and the PSW. (See the "Supervisor Control" section for a complete description of the SVC IH.)		SVCERR
4. **DATERR entry point** — used by the program check IH when a recursive translation exception occurs during either the program check IH's processing, or RTM1's FRR processing. Before calling RTM1, the program check IH has attempted to circumvent any further translation failures by altering the STOR (segment table origin register) which points to the master address space's segment tables. If errors occur again, the program check IH places the system in a disabled wait state. RTM1 does not allow normal recovery processing to occur during DATERR processing since the non-common areas of the failing address space are no longer addressable. If a supervisor control routine was in control when the original error occurred, then its FRR will be given control, with a special indication to warn it that private areas are no longer addressable. The super FRR may recover the address space or terminate it (via MEMTERM). If a super FRR is not available, RTM1 bypasses all recovery, records the incident and terminates the address space.		DATERR
5. **MACHCK reentry** — used when RTM1 set up MCH (machine check handler) or ACR (alternate CPU retry) for re-entry into RTM1 after RTM1 was initially entered for a machine check. RTM1 uses this entry to attempt software recovery processing if a machine check caused software damage.		MACHCK

Diagram 22-2. RTM1 Initialization (IEAVTRT1) (Part 3 of 4)

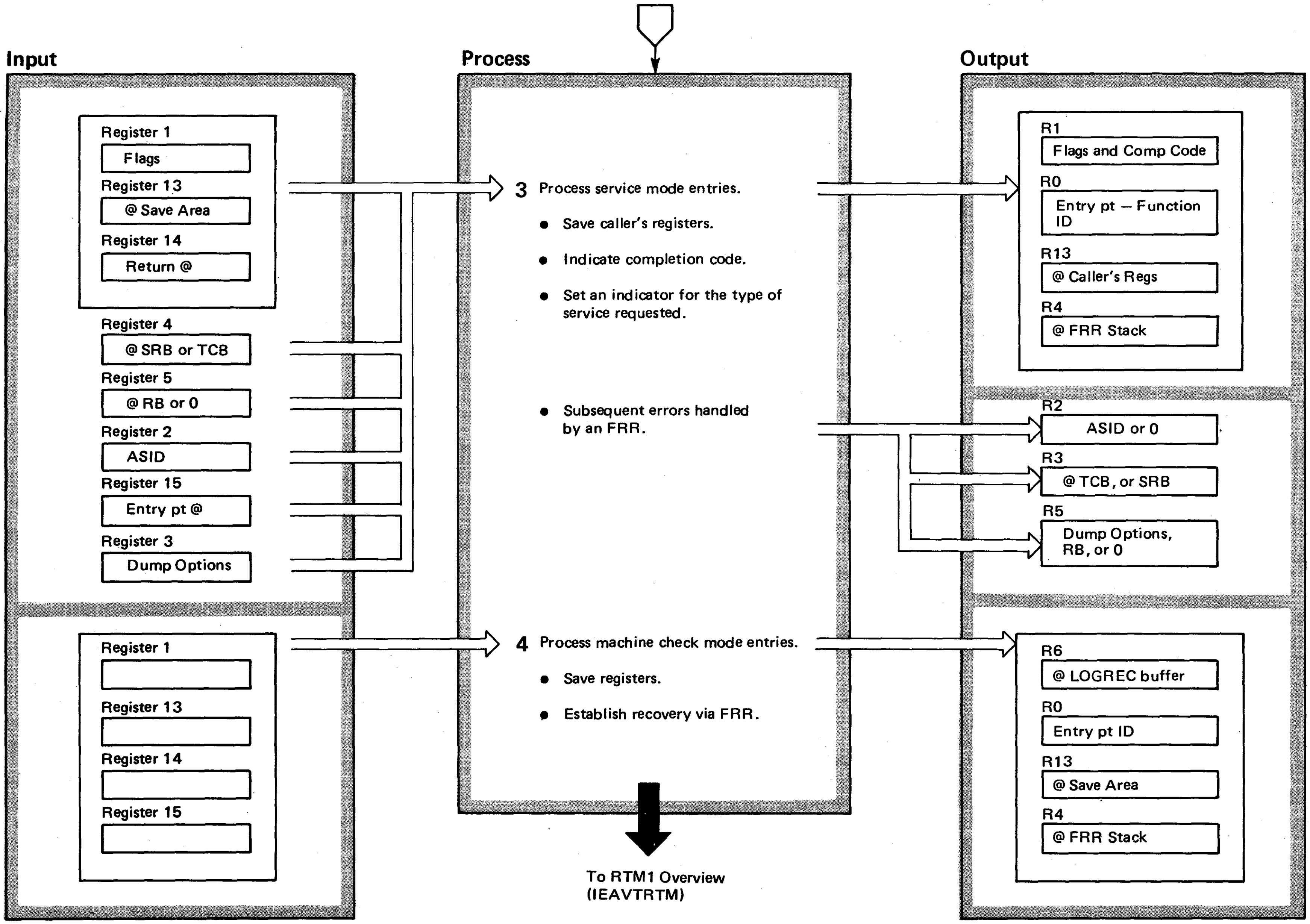


Diagram 22-2. RTM1 Initialization (IEAVTRT1) (Part 4 of 4)

Extended Description	Module	Segment
3 RTM1 initialization prepares the following entry points for service mode:		
1. **PGIOERR entry point** – used by the reset subroutine of real storage management when an error occurs while processing a page fault. The routine that suffered the paging error is forced to issue an ABEND instruction (SVC 13) to cause linkage to R/TM for recovery and termination services. Initialization processing for this entry point passes the address of the TCB or SRB that suffered the error. If a task suffered the error, the address of the RB is also passed.		PGIOERR
2. **ABTERM entry points** – used by key 0, supervisor state routines to set a task up for entry to RTM2 for ABEND. There are two types of ABTERM entry: ABTERM with ASID option; and ABTERM without ASID.		
ABTERM with ASID is a request to terminate a task in an address space other than the current one. RTM1 schedules itself as an SRB into the specified address space to perform the ABTERM request. RTM1 saves the caller's registers in a caller-supplied save area.		XABTERM
ABTERM without ASID is a request to terminate a task in the current address space. RTM1 saves the caller's registers and PSW, and performs the ABTERM request.		CABTERM

Extended Description	Module	Segment
3. **MEMTERM entry point** – used to request scheduling an address space termination. Since there are no specific lock requirements, the caller must provide a register save area. R/TM will perform the address space termination. RTM1 performs a MEMTERM asynchronously with dependencies on locks and the dispatcher. Therefore, control may or may not return to the caller, depending on the lock status when the caller issued the request.		MEMTERM
4. **ABTERM reentry** – used when RTM1 scheduled itself as an SRB during a previous entry when the caller requested ABTERM with the ASID option. When entered at this entry point, RTM1 is operating as an SRB in the specified address space.		IEAVTRTX
4 MCH (machine check handler) and ACR (alternate CPU recovery) use this entry point when requesting hardware recording and hardware damage repair. The caller passes the address of a LOGREC buffer which contains all the information about the error. If RTM1 subsequently determines that software recovery is warranted, it will establish the appropriate software interface.		IEAVTRTN

Diagram 22-3. Process Hardware Errors (IEAVTRTM) (Part 1 of 4)

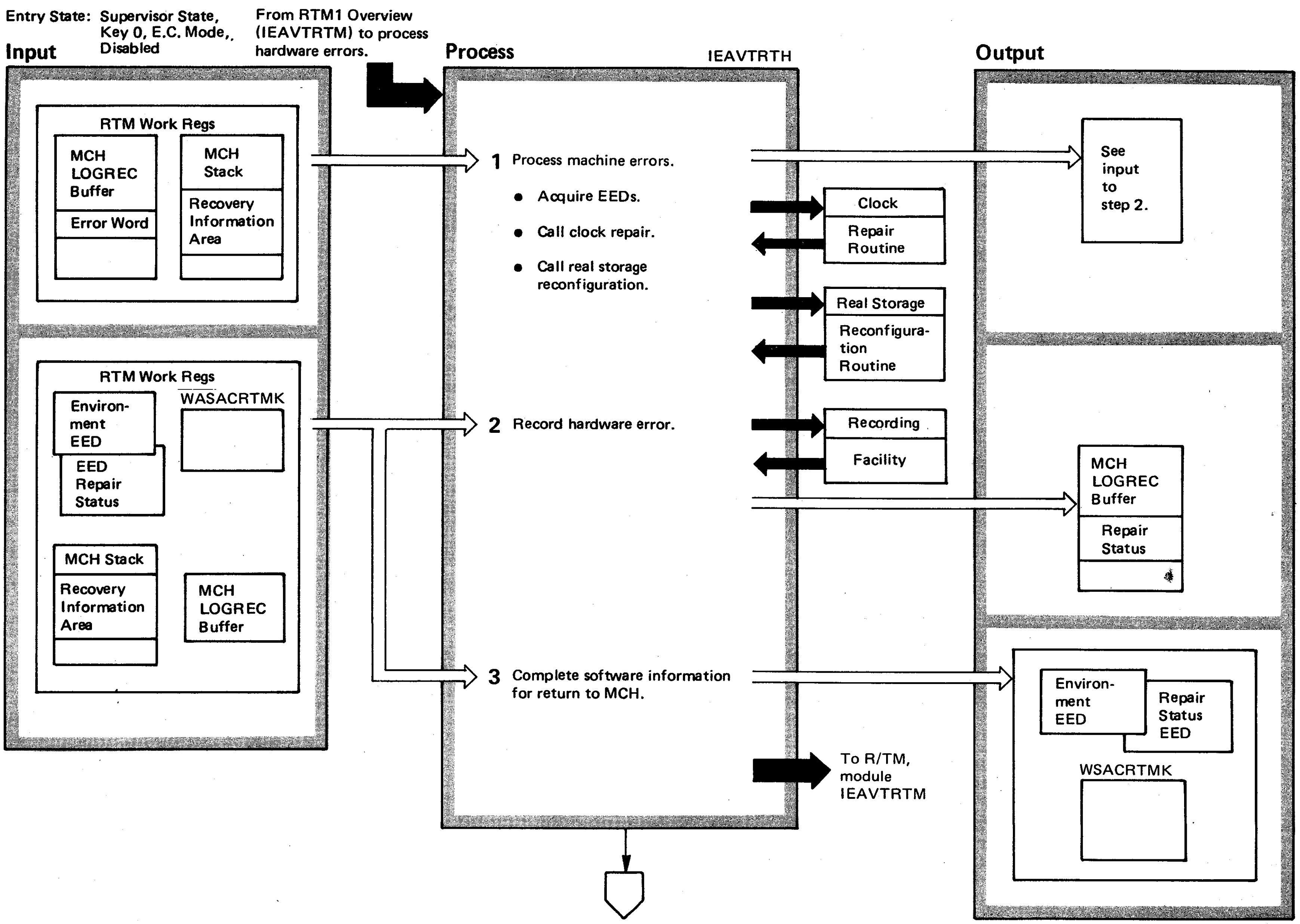


Diagram 22-3. Process Hardware Errors (IEAVTRTM) (Part 2 of 4)

Extended Description	Module	Segment
MCH uses RTM1 as a subroutine to attempt the repair of clock and storage errors, and to record all hardware errors. The main body of RTM1's hardware error processing is contained in the module IEAVTRTH. During the time RTM1 is working with the LOGREC buffer, MCH protects RTM1 from any further entry for new machine errors. When RTM1 processes ACR (alternate CPU recovery) errors, ACR provides protection from new machine check entries by disabling machine checks during the RTM1 process.		
1 For ACR and "hard" errors (that is, machine checks where hardware recovery has not been able to recover the operation) RTM1 obtains two EEDs (extended error descriptors) to pass on information concerning the error.	IEAVTRTH	EEDREQST
For ACR and timer errors, the clock repair routine (module IEAVRCLS) receives control to recover software timing functions.		CLOCKREP
For storage data checks or storage key failures, the RSR (real storage reconfiguration) routine (module IEAVRCF) receives control.		RSRECON
To attempt repair upon return from RSR, repair status is placed in the LOGREC buffer and in the EEDs.		BLDPLIST
2 RTM1 places a record of the hardware failure on SYS1.LOGREC via the recording facility for both "hard" and "soft" errors (that is, errors which were successfully recovered by the hardware).		RECORDNG
3 The WSACRTMK contains the registers and PSW that MCH needs to restore when RTM1 subsequently returns control to MCH. For "soft" errors, the routine the machine check handler interrupted has sustained no software damage and may resume its processing at the point of interruption. In this case, the information in the WSACRTMK consists of the registers and PSW at the time of the machine check. For a "hard" error, the routine in control at the time of the machine check did suffer software damage; RTM1 must be reentered to perform software recovery. Therefore, the PSW RTM1 placed in the WSACRTMK points to the machine check reentry point (IEAVTRTN) in IEAVTRT1. The registers in the WSACRTMK contain the values RTM1 expects on reentry.		SOFTINFO

Diagram 22-3. Process Hardware Errors (IEAVTRTM) (Part 3 of 4)

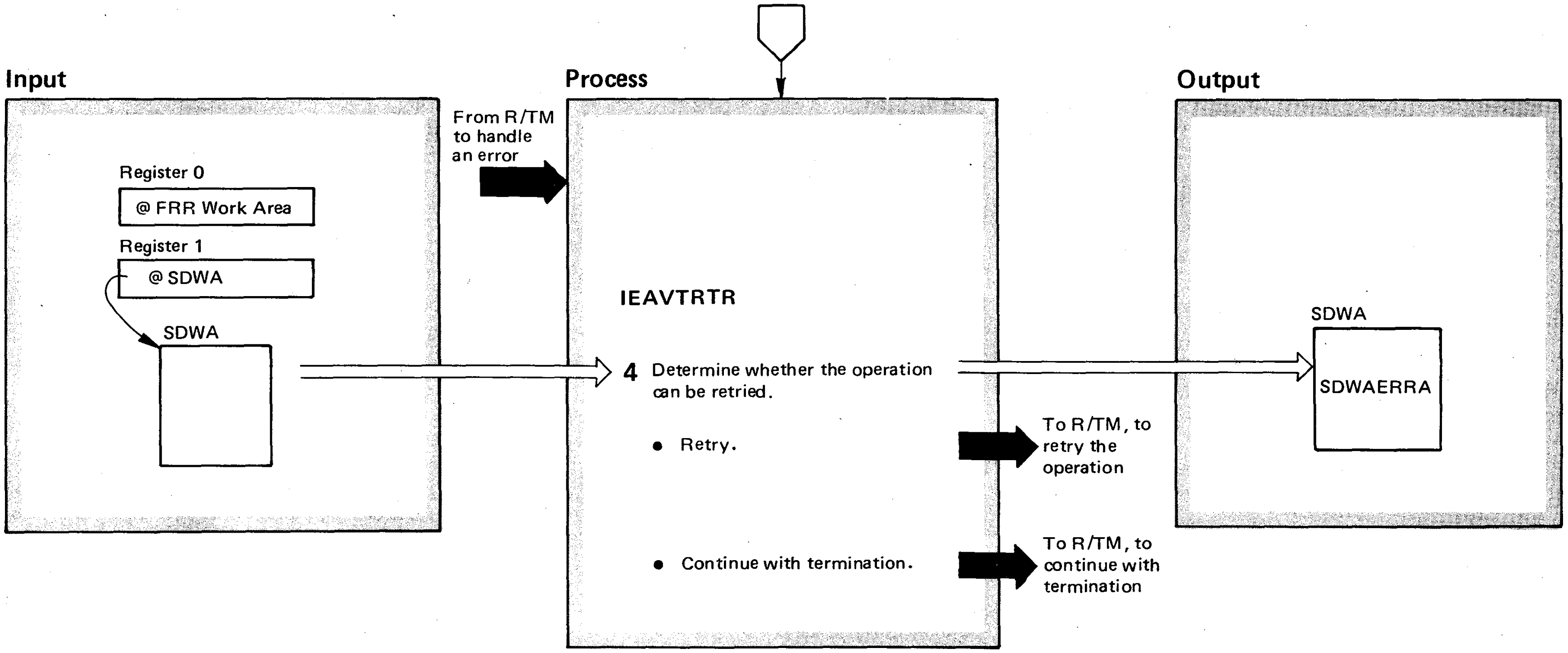
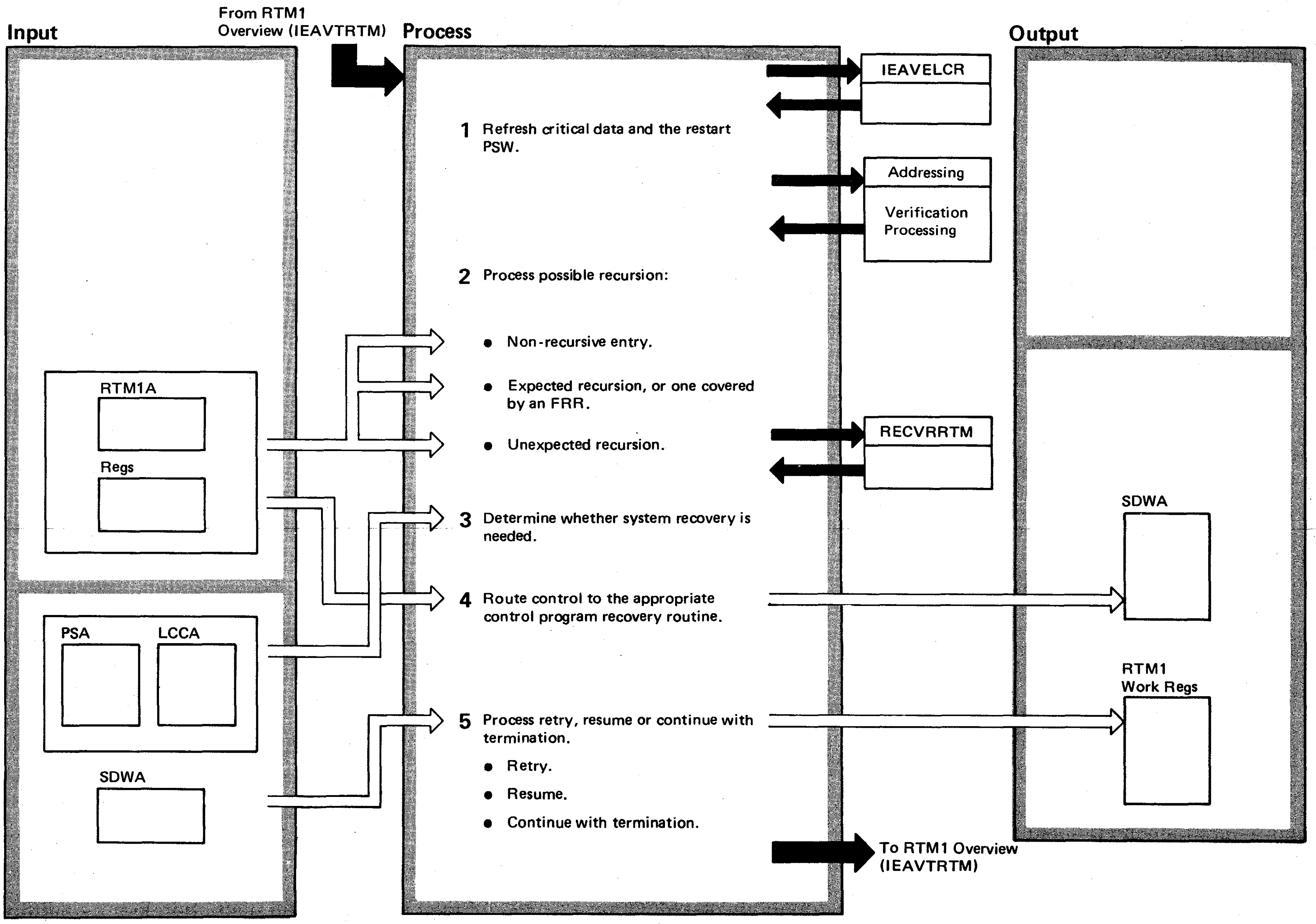

Input
Register 0
@ FRR Work Area
Register 1
@ SDWA
SDWA
Process
From R/TM to handle an error
IEAVTRTR
4 Determine whether the operation can be retried.
• Retry.
• Continue with termination.
To R/TM, to retry the operation
To R/TM, to continue with termination
Output
SDWA
SDWAERRA

Diagram 22-3. Process Hardware Errors (IEAVTRTM) (Part 4 of 4)

Extended Description	Module	Segment
4 Processing hardware error establishes an FRR (functional recovery routine) to protect itself. When the FRR receives control, it examines the error information in the SDWA — pointed to by register 1 — to determine the cause for the entry and to determine whether the function can be retried. For DAT (dynamic address translation) and restart errors, the FRR continues with termination, while all others can be retried. When the function cannot be retried, the FRR frees any EEDs acquired during process hardware error operation, and gives control back to R/TM to continue with termination. If the function can be retried, the process hardware error operation will be given control again.	IEAVTRTR	RTHFRR

Diagram 22-4. Processing SLIH Requests (IEAVTRTM) (Part 1 of 2)



1 Refresh critical data and the restart PSW.

2 Process possible recursion:

- Non-recursive entry.
- Expected recursion, or one covered by an FRR.
- Unexpected recursion.

3 Determine whether system recovery is needed.

4 Route control to the appropriate control program recovery routine.

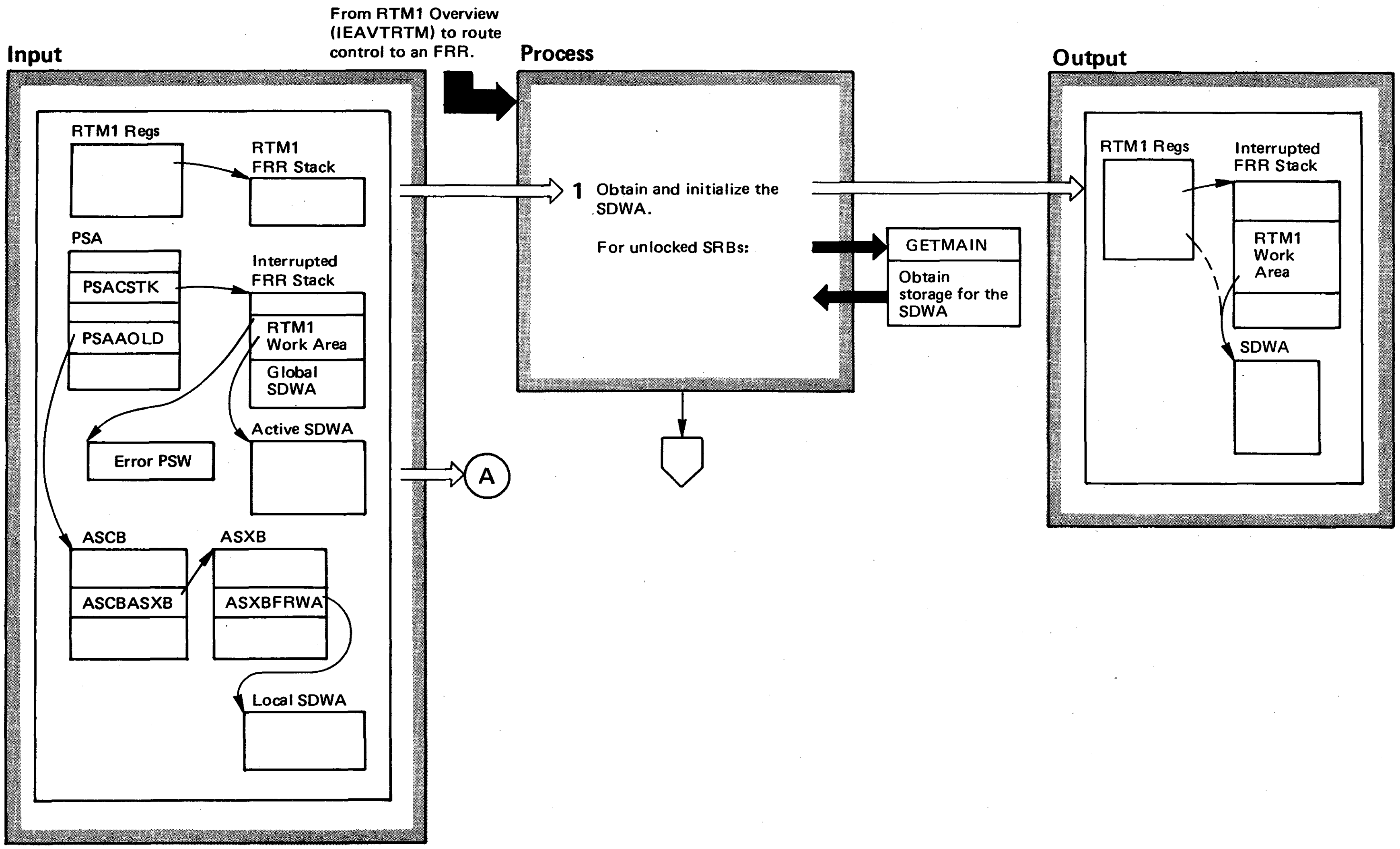
5 Process retry, resume or continue with termination.

- Retry.
- Resume.
- Continue with termination.

Diagram 22-4. Processing SLIH Requests (IEAVTRTM) (Part 2 of 2)

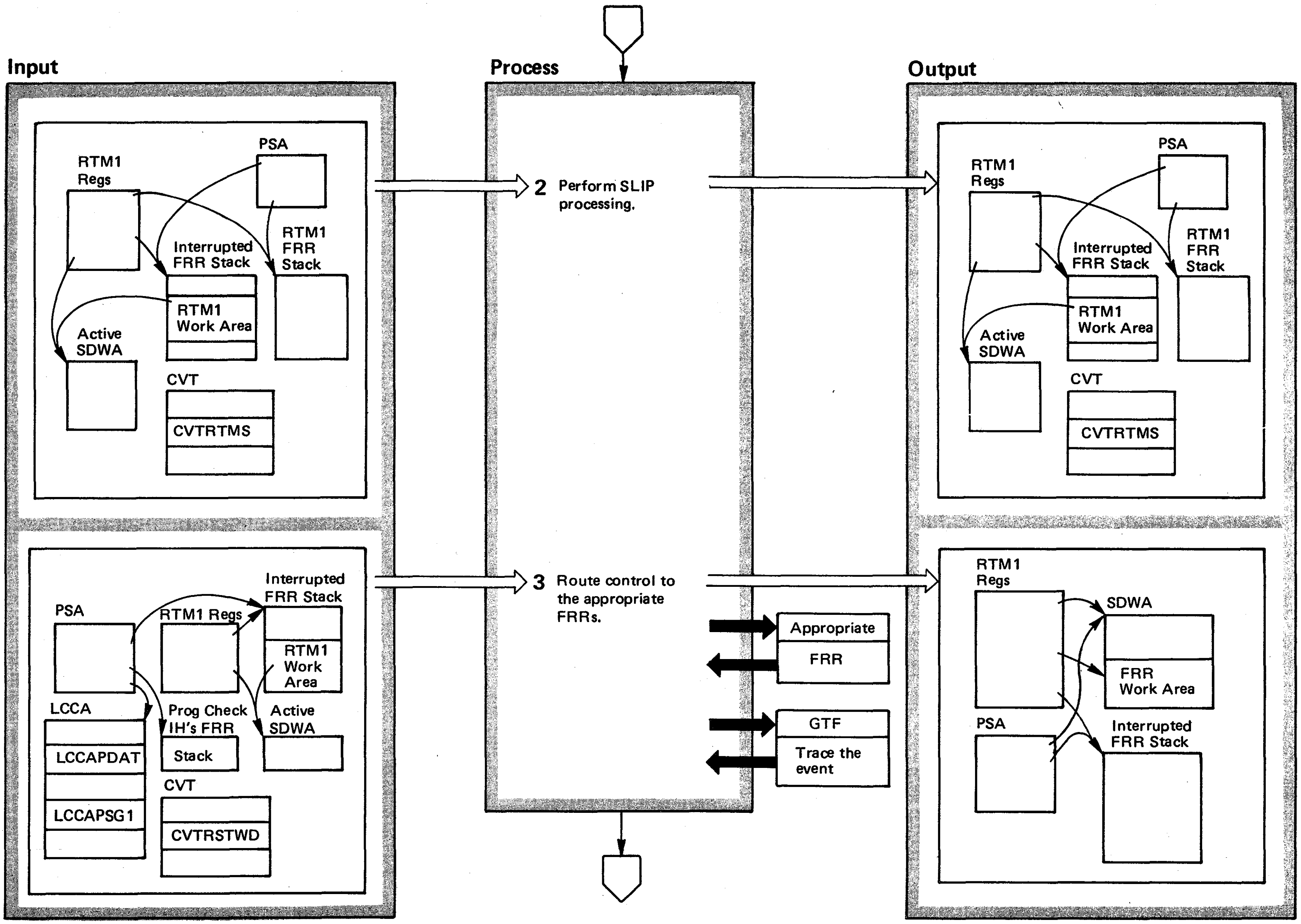
Extended Description	Module	Segment
This chart illustrates the flow of control during RTM1's SLIH processing.		
1 Whenever RTM1 performs SLIH processing, RTM1 first attempts to refresh critical common fixed constants. RTM1 refreshes low storage (via IEAVELCR) and attempts, on its own, to refresh the restart new PSW.	IEAVTRTM	REFRESH
2 RTM1 continues SLIH processing for non-recursive entry into RTM1; for anticipated recursive entry; or for recursion covered by one of RTM1's FRRs. Otherwise, RTM1 processes an unanticipated recursive entry by routing control to a recovery routine (RECVRRTM in module IEAVTRTR) that determines whether any recovery of this recursive error can be performed.		RECURSE
3 RTM1 determines the system state at the time of the interruption by examining indicators in the PSA and LCCA. The succeeding flow of control during SLIH mode processing depends on the system state (system mode or task mode).		SYSTATE
4 For errors in global, local, SRB, or supervisor control code (that is, the state determined in step 3 is system mode), control program recovery must be performed. To effect this recovery, routing FRR processing (module IEAVTRTS) receives control and routes control to any appropriate recovery routine (FRR) associated with the failing routine.		SYSRCVR
For errors in task mode when the interrupt occurred, RTM1 skips this step and the following step and sets the work registers to reschedule the interrupted task for entry to RTM2.		SETUPABT
5 RTM1 analyzes the output from routing to FRRs. For retry requests, control goes to R/TM's clean-up and exit processing. For valid resume requests, RTM1 establishes an interface to the reschedule CPU function. Otherwise RTM1 continues with termination, setting its work registers to establish the correct interface to the reschedule function.		SYSRCVR DATPERC
For DATERR entries to RTM1, RTM1 establishes the address space termination interface.		
When the system is in SRB mode, RTM1 establishes the ABTERM interface to terminate the task associated with the failing SRB.		SRBPERC

Diagram 22-5. Routing to FRRs (IEAVTRTS) (Part 1 of 8)



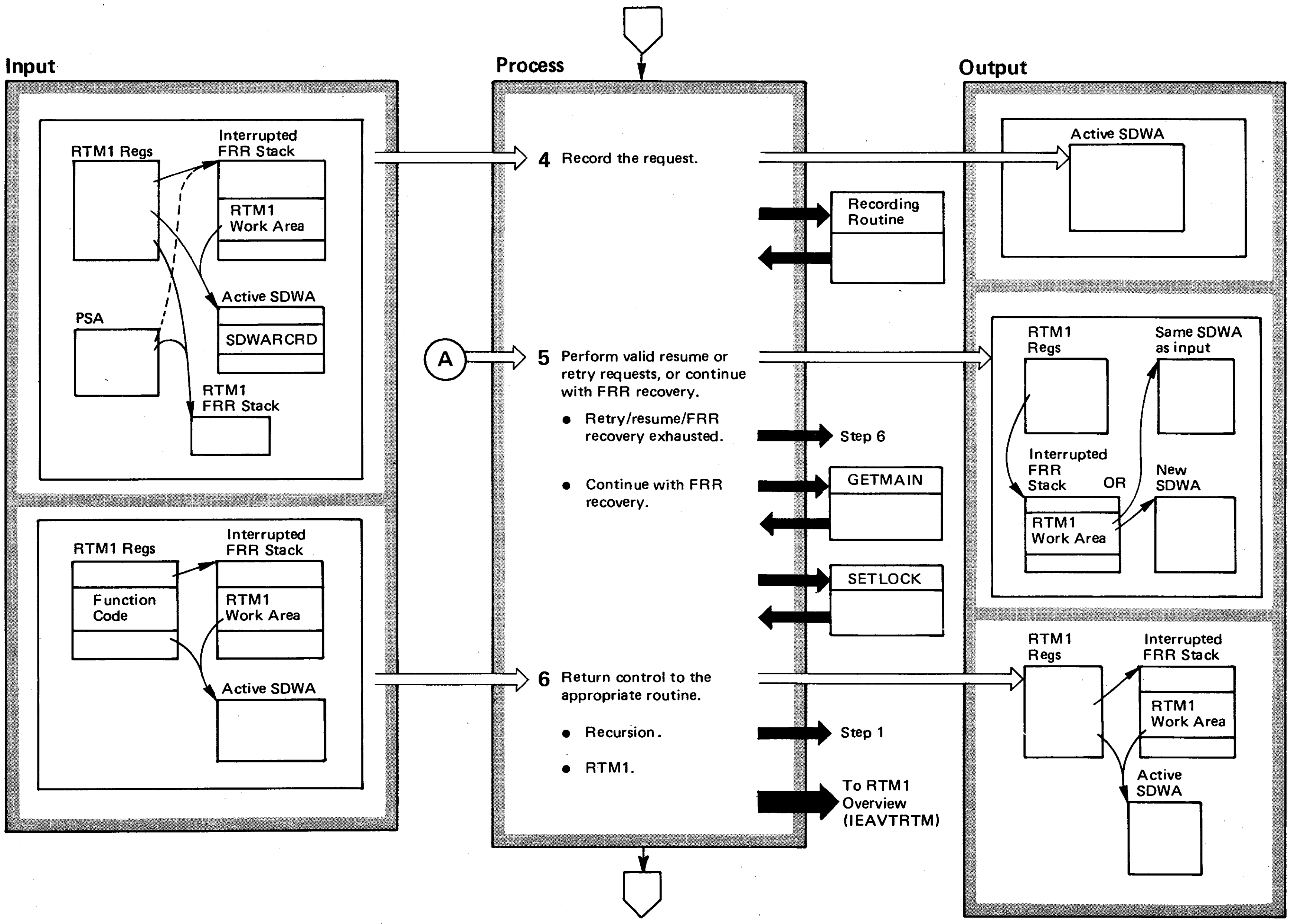
Extended Description	Module	Segment
RTM1 routes control to FRRs (functional recovery routines (IEAVTRTS)) defined by supervisor routines to protect themselves from errors. The function provides the interface and control between failing supervisor routines and their FRRs. The FRRs reside on "stacks." Allocated as predefined areas in SQA (system queue area), consists of a header (used to control the contents of the stack), a workarea (used by RTM1 when performing FRR routing), and a fixed number of FRR entries. (See Initializing FRR Stacks (IEAVTSIN)). Each FRR stack defines a path through the supervisor as follows:		
• SVC/I/O-dispatcher stack. Defines the path through the supervisor used when servicing SVC interruptions or I/O interruptions, or during dispatcher processing. (One stack can be used for all of these three functions, since the processing for any one function is not dependent on the processing of the other two functions.) Those supervisor functions servicing I/O or SVC interrupts as well as those functions comprising the dispatcher place their FRRs on this stack.		
• Machine check stack. Defines the path through the supervisor taken when a machine check interruption occurs. Supervisor functions processing machine checks place their FRRs on this stack.		
• Program check stack. Defines the path through the supervisor taken when a program check occurs. Supervisor functions processing program checks place their FRRs on this stack.		
• External interruption handler 1 stack. Defines the path through the supervisor when an external interruption occurs, and there are no recursions. Supervisor functions processing external interruptions place their FRRs on this stack. (See the M.O. diagram, External Interruption Handler (IEAVEEXT) in the *Supervisor Control* section for a complete description of the external interruption handler and its method of handling recursions.)		
• External interruption handler stack 2. Defines the path through the supervisor when an external interruption occurs for a second time, while the external interruption handler is processing a previous interruption. Supervisor functions processing external interruptions place their FRRs on this stack.		
• External interruption handler 3 stack. Defines the path through the supervisor for an external interruption when one recursion has occurred already and is being processed and this is the second one. Supervisor functions processing external interruptions place their FRRs on this stack.		
• Restart interruption handler stack. Defines the path through the supervisor when a restart interruption occurs. Supervisor functions processing restarts place their FRRs on this stack.		
• Normal stack. Defines the path through the supervisor used when processing normal requests for supervisor services made directly (or indirectly) by problem programs.		
When an error occurs in a supervisor function covered by an FRR, routing to FRRs gives control to the appropriate FRR defined on the stack protecting that function. Routing to FRRs supplies the FRR receiving control with a complete description of the error in the SDWA (system diagnostic work area). Routing to FRRs acquires an SDWA based on the system state at the time the error occurred:		
• Global SDWA – associated with the FRR stack defining the supervisor path that failed when the system operates physically disabled (globally locked or supervisor control mode).		
• Local SDWA – associated with the supervisor path that failed when the system operates logically disabled (locally locked).		
• GETMAIN SDWA – an SDWA obtained via a GETMAIN request and associated with the supervisor path that failed when the system operates only in SRB mode.		
1 Routing to FRRs acquires an SDWA, and initializes it with error informations obtained from the input registers. These registers contain values set in the RTM1 mainline module (IEAVTRTM), as shown in M.O. diagram, RTM1 Initialization (IEAVTRTM).	IEAVTRTS	

Diagram 22-5. Routing to FRRs (IEAVTRTS) (Part 3 of 8)



Extended Description	Module	Segment
2 SLIP (serviceability level indicator processing) uses the CVTRTMS field of the CVT as input to determine whether additional serviceability processing should occur. This field contains indicators set manually when additional serviceability is desired for system errors. R/TM determines the serviceability level requested (modules IEAVTRTR and IEAVTRT2). SLIP processing takes an SVC dump, or places the system in a wait state.	IEAVTRTR IEAVTRT2 IEAVTRTR	SLIPPER SLIP IEAVTRTL
3 Control goes to the appropriate FRR via an LPSW (load PSW) instruction, passing the SDWA as input.	IEAVTRTS	ROUTE
Routing to FRRs gives control to GTF (generalized trace facility) to trace the FRR recovery event.		TRACEFRR

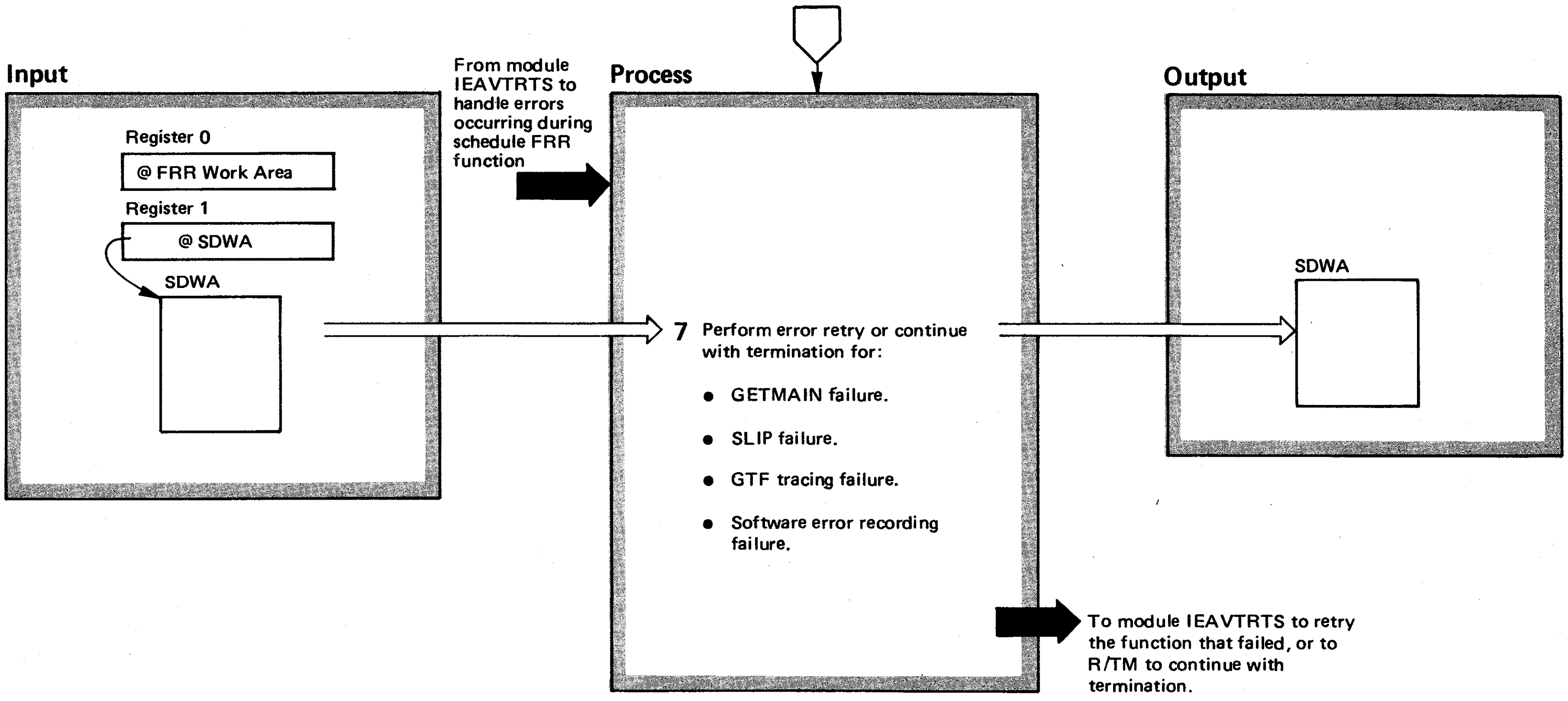
Diagram 22-5. Routing to FRRs (IEAVTRTS) (Part 5 of 8)



Extended Description	Module	Segment
4 Routing to FRR processing conditionally records the SDWA describing the error and the actions taken if:		RECORD
• The FRR that received control requests recording.		
• No FRRs exist on the stack defining the supervisor path that failed.		
• The FRR that received control had an error while attempting recovery.		
5 Routing to FRR processing honors valid requests from the FRR to:		CHKRCDE
• Resume processing of the interrupted supervisor path at the point immediately following the interruption.		
• Retry the interrupted supervisor path at a point specified by the FRR.		
• Continue with FRR recovery when the FRR in control fails to completely recover from the error.		

Extended Description	Module	Segment
When FRR recovery continues, routing FRR processing prepares to route to additional FRRs on the stack. This is called 'percolation,' and it means continue with termination. Since the FRR stack defines a supervisor path that failed, however, and since each FRR corresponds one-to-one with a function in the path, the FRR executes in the same system state as the function it protects. When an FRR must continue with termination, the FRR receiving control (to continue the termination) must clean up or request the clean up of any resources associated with the function it protects. Because of a potential change in system state resulting from clean up, routing FRR processing involves:		
• Insuring that the SDWA contains valid error information.		
• Locating the next FRR to receive control, in a LIFO manner, and adjusting the stack header to indicate the next FRR to receive control.		
• Releasing any locks as specified by the FRR requesting to continue with termination.		
6 Routing FRR processing returns to M.O. diagram, RTM1 Overview (IEAVTRTM), to honor resume or retry requests, or after all FRRs on the stack have been exhausted.		EXIT
For recursive entries where an FRR has had an error, FRR recovery continues.		

Diagram 22-5. Routing to FRRs (IEAVTRTS) (Part 7 of 8)


Input
Register 0
@ FRR Work Area
Register 1
@ SDWA
SDWA
From module IEAVTRTS to handle errors occurring during schedule FRR function
Process
Output
SDWA
To module IEAVTRTS to retry the function that failed, or to R/TM to continue with termination.

7 Perform error retry or continue with termination for:

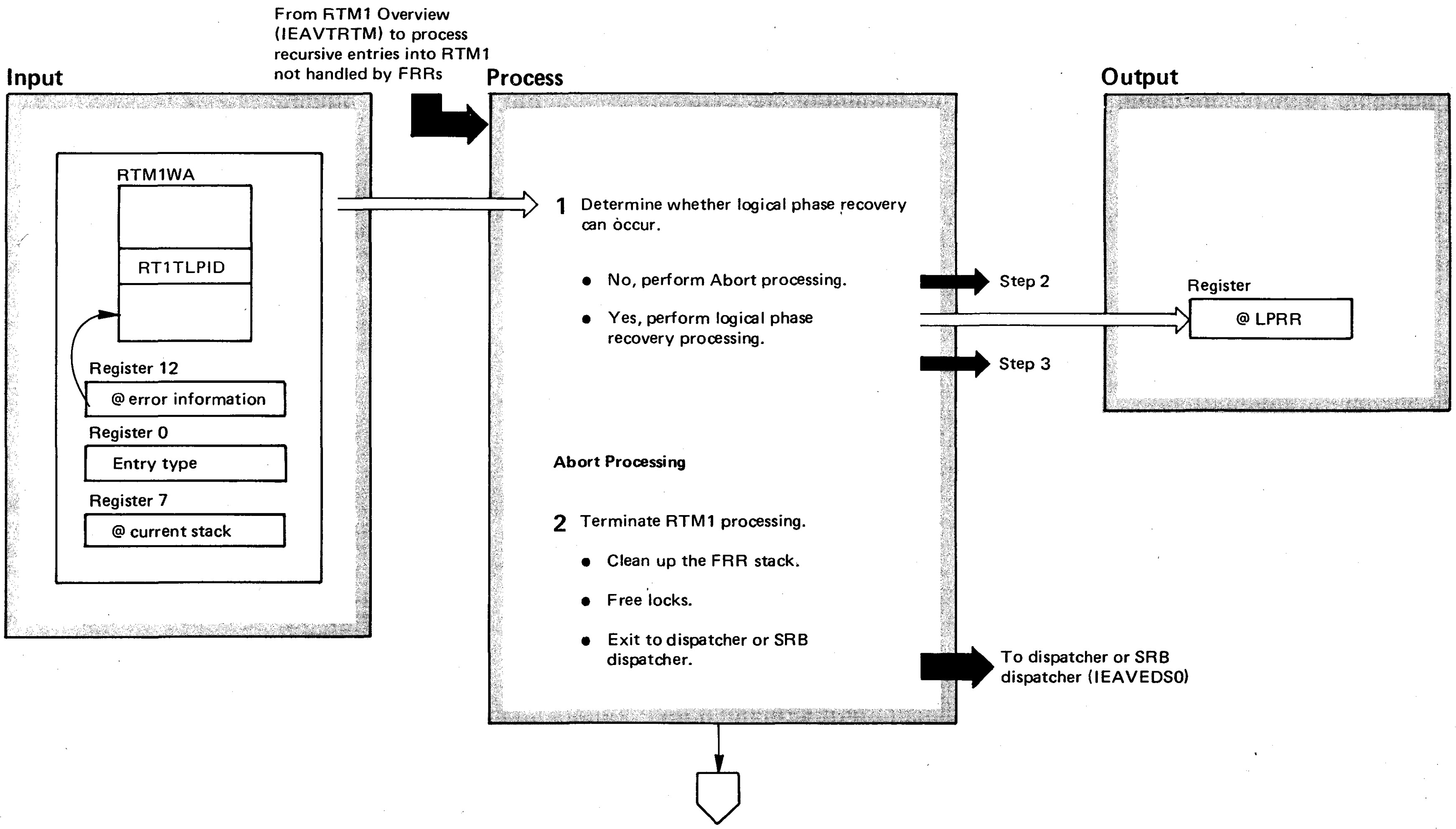
- GETMAIN failure.
- SLIP failure.
- GTF tracing failure.
- Software error recording failure.

Diagram 22-5. Routing to FRRs (IEAVTRTS) (Part 8 of 8)

Extended Description	Module	Segment
7 The routing to FRR function protects itself from errors with several FRRs. These FRRs protect against:	IEAVTRTR	
• Failures occurring in GETMAIN processing.		
• Failures occurring during SLIP processing.		
• Failures occurring while GTF traces an event.		
• Failures occurring while software errors are being recorded.		
RCOVSLP1, RCOVRGTF, and RCOVRCRD may set up ABORT processing for double errors occurring in IEAVTRTS processing. (See the M.O. diagram RTM1 Recursion Processing (IEAVTRTR), for a description of ABORT processing.)		
When an error occurs during GETMAIN processing while attempting to acquire an SDWA for an unlocked SRB mode failure, this FRR gets control if GETMAIN recovery is unsuccessful. The FRR retries all errors except DAT (dynamic address translation) and restart errors. Retry will occur at the point in routing to FRRs where the local SDWA is acquired for this SRB failure. DAT and restart errors cause continue with termination to be requested by this FRR.		RCOVGETM
The SDWA contains indicators explaining what happened during this FRR's processing, as follows:		
RCOVGETM places the following messages in the variable recording area of the SDWA:		
• Retry IEAVTRTS after failure in GETMAIN attempting to acquire an SRB SDWA for use by IEAVTRTS.		
• Percolate on DATERR or restart error occurring while attempting to acquire SRB SDWA via GETMAIN.		

Extended Description	Module	Segment
An FRR protects SLIP processing. The FRR retries all errors except DAT and restart errors. Retry will occur at the point past SLIP processing. For DAT and restart, the FRR indicates continue with termination.		RCOVSLP1
The SDWA contains indicators explaining what happened during the FRR processing.		
This FRR receives control if the SLIP2ACT entry for SLIP fails. This FRR frees resources obtained by SLIP, and indicates continue with termination.		SLIP2FRR
An FRR protects routing to FRRs from an error occurring while GTF traces another FRR's actions. The FRR retries all errors except DAT and restart errors. Retry will occur at the point past GTF processing. For DAT and restart, the FRR indicates continue with termination.		RCOVRGTF
The SDWA contains indicators explaining what happened during the FRR processing.		
RCOVRGTF places the following messages in the variable recording area of the SDWA:		
• Retry IEAVTRTS after GTF failure attempting to trace SDWA returned by FRR.		
• Percolate on DATERR or restart error occurring while attempting to trace SDWA via GTF.		
An FRR protects software error recording of errors being already handled by another FRR. The FRR retries all errors except DAT and restart errors. Retry will occur at the point past software error recording. For DAT and restart, the FRR indicates continue with termination.		RCOVRCRD
The SDWA contains indicators explaining what happened during the FRR processing.		
RCOVRCRD places the following messages in the variable recording area of the SDWA:		
• Retry IEAVTRTS after failure in software recording facility attempting to record the SDWA.		
• Percolate on DATERR or restart error occurring while attempting to record the SDWA.		

Diagram 22-6. RTM1 Recursion Processing (IEAVTRTR) (Part 1 of 4)



1 Determine whether logical phase recovery can occur.

- No, perform Abort processing.
- Yes, perform logical phase recovery processing.

Abort Processing

2 Terminate RTM1 processing.

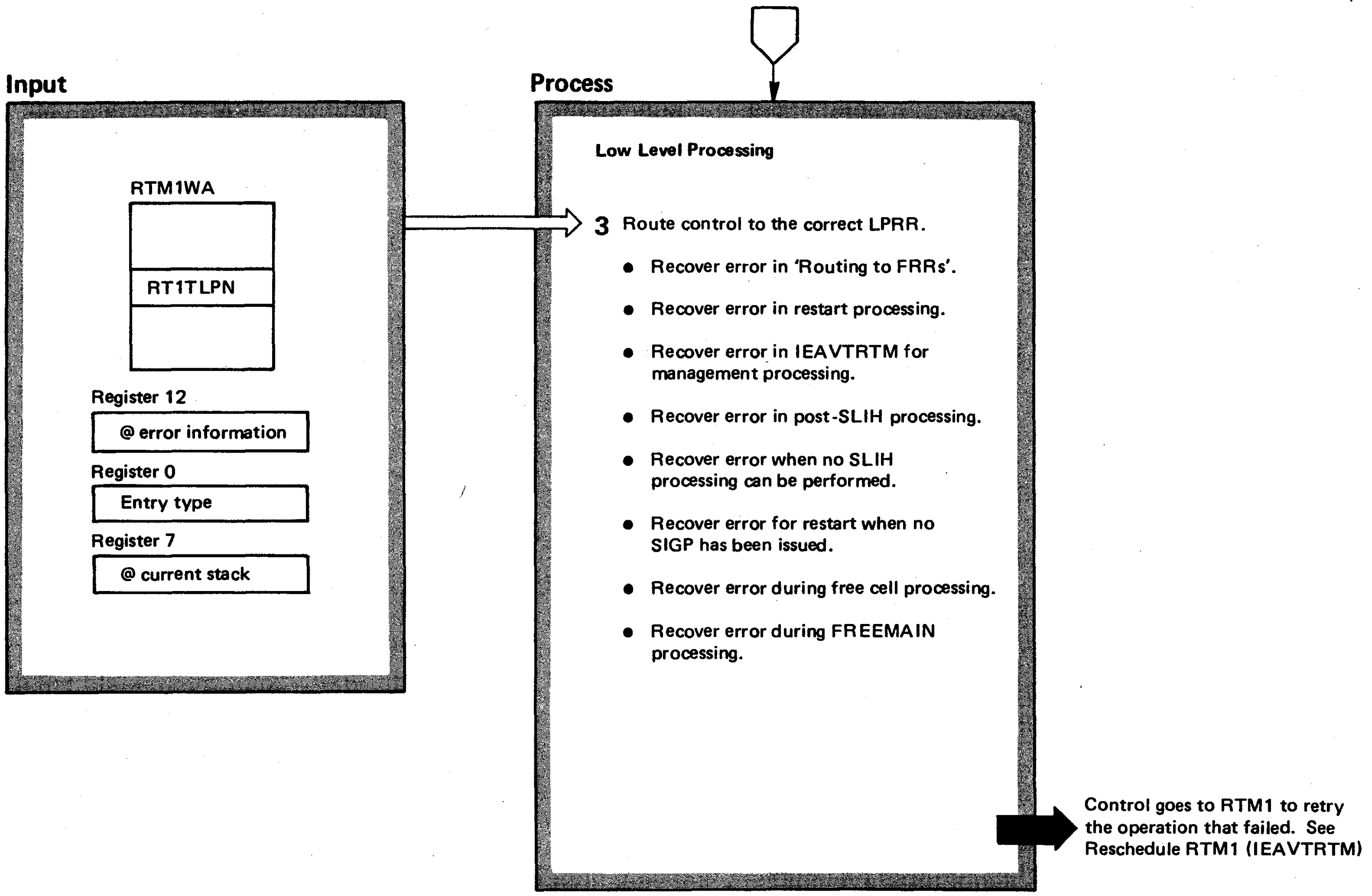
- Clean up the FRR stack.
- Free locks.
- Exit to dispatcher or SRB dispatcher.

Diagram 22-6. RTM1 Recursion Processing (IEAVTRTR) (Part 2 of 4)

Extended Description	Module	Segment
In certain paths through RTM1 processing, recursions cannot be processed by FRRs (functional recovery routines). For example, the phase of the module that actually routes control to FRRs (module IEAVTRTS) cannot be protected by an FRR – if this phase does not work, it cannot route to an FRR to protect itself. To handle these situations where certain phases cannot be protected with an FRR, RTM1 uses LPRRs (logical phase recovery routines). To use LPRRs, RTM1 tracks its processing. The tracking information consists of two items: • An LPID – a logical phase ID that identifies the LPRR that can process the recursion. • An LPN – a logical phase number that identifies the phase of RTM1's processing in control at the time of the error. Recursion processing routes control to the LPRR identified by the LPID.		

Extended Description	Module	Segment
1 After an RTM1 process, IEAVTRTM has discovered a recursive condition, control goes to the recursion processing routine. Recursion processing first determines whether a logical phase identifier exists, by checking the RT1TLPID field of the RTM1WA. Any time an RTM1 logical phase uses an LPRR for recovery, it sets the RT1TLPID to a non-zero number. The recursion processing routine gives control to the correct LPRR if it finds a non-zero number in the field. If it finds a zero, this means that no specific LPRR exists, and the Abort LPRR must receive control.	IEAVTRTR	RECVRRTM
2 The Abort processing routine handles recursions by performing clean up processing. Abort processing releases any locks and resets any FRR stack pointer values. In general, Abort processing removes any traces of the original error. Control goes to either the dispatcher or the SRB dispatcher, depending on the mode at the time of error.		ABORT

Diagram 22-6. RTM1 Recursion Processing (IEAVTRTR) (Part 3 of 4)



Low Level Processing

3 Route control to the correct LPRR.

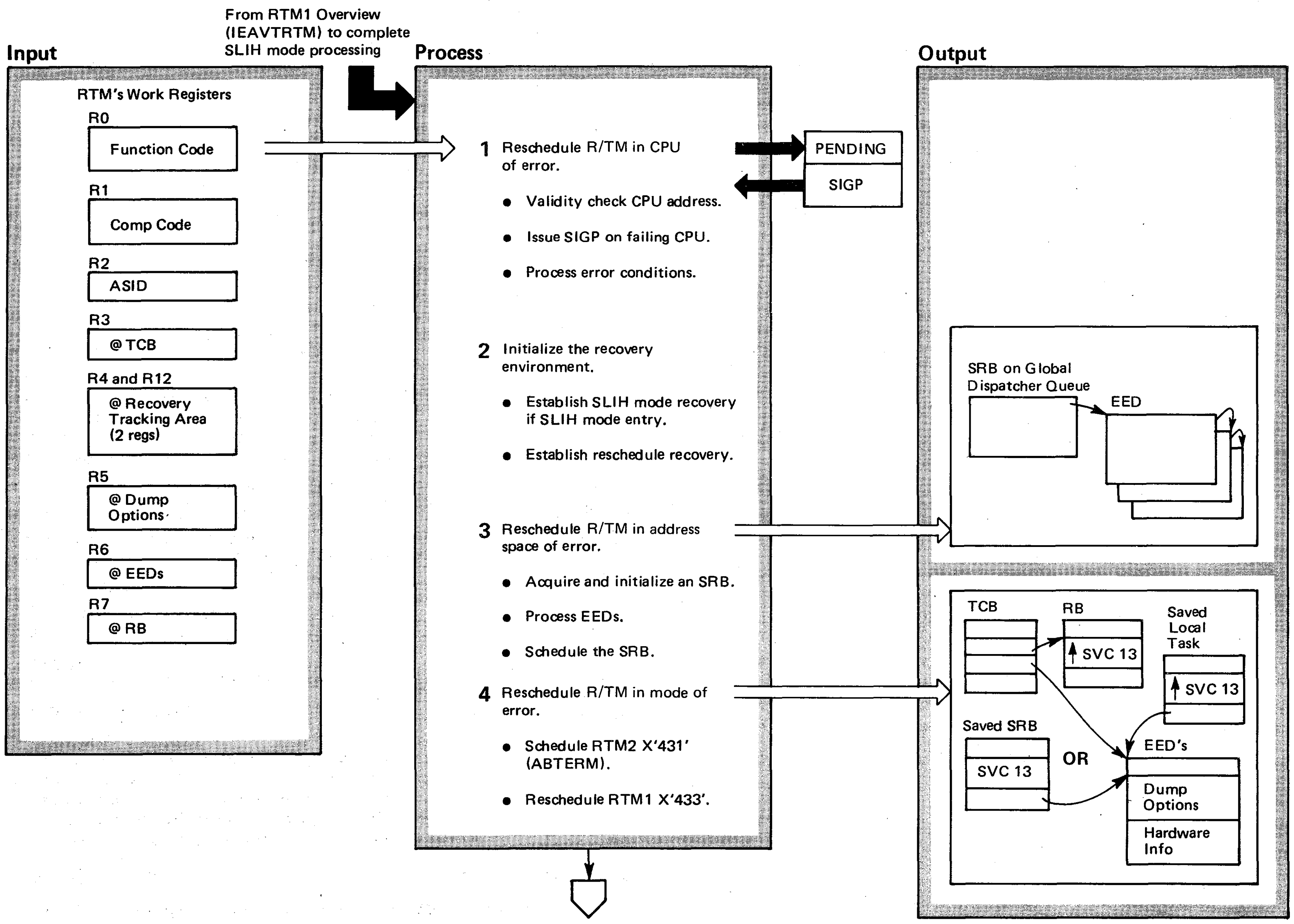
- Recover error in 'Routing to FRRs'.
- Recover error in restart processing.
- Recover error in IEAVTRTM for management processing.
- Recover error in post-SLIH processing.
- Recover error when no SLIH processing can be performed.
- Recover error for restart when no SIGP has been issued.
- Recover error during free cell processing.
- Recover error during FREEMAIN processing.

Control goes to RTM1 to retry the operation that failed. See Reschedule RTM1 (IEAVTRTM)

Diagram 22-6. RTM1 Recursion Processing (IEAVTRTR) (Part 4 of 4)

Extended Description	Module	Segment
3 When the RT1TLPID indicates a non-zero number, an LPRR exists. The recursion processing routine routes control to the various LPRRs according to the type of recovery desired. (The RT1TLPN field of the RTM1WA indicates the logical phase in control.) RTM1 LPRRs recover from the following:		LPRECOV1
• Errors in routing to FRRs.	IEAVTRTS	SRMDRCOV
• Errors in mainline SLIH post-processing after routing to FRRs.		RVPOSTSR
• Errors occurring in mainline SLIH when no routing to FRR processing has been performed.		RVNORTS
• Errors in restart processing.		RVRSTRT
• Errors in restart processing when no SIGP (signal processor) macro instruction was issued.		RVNORST
• Errors occurring during FREECELL processing.		RVEEDFRE
• Errors occurring during FREEMAIN processing.		RVFREEMN
If the LPRR can recover from the recursive error, control returns to either IEAVTRTS or IEAVTRTM to resume processing of the original error. Otherwise, the LPRR will return to RTM1 main processing, to continue processing the new error.		
• Errors in the management and control routing of RTM1 (IEAVTRT1).		

Diagram 22-7. Reschedule RTM1 (IEAVTRTM) (Part 1 of 4)



1 Reschedule R/TM in CPU of error.

- Validity check CPU address.
- Issue SIGP on failing CPU.
- Process error conditions.

2 Initialize the recovery environment.

- Establish SLIH mode recovery if SLIH mode entry.
- Establish reschedule recovery.

3 Reschedule R/TM in address space of error.

- Acquire and initialize an SRB.
- Process EEDs.
- Schedule the SRB.

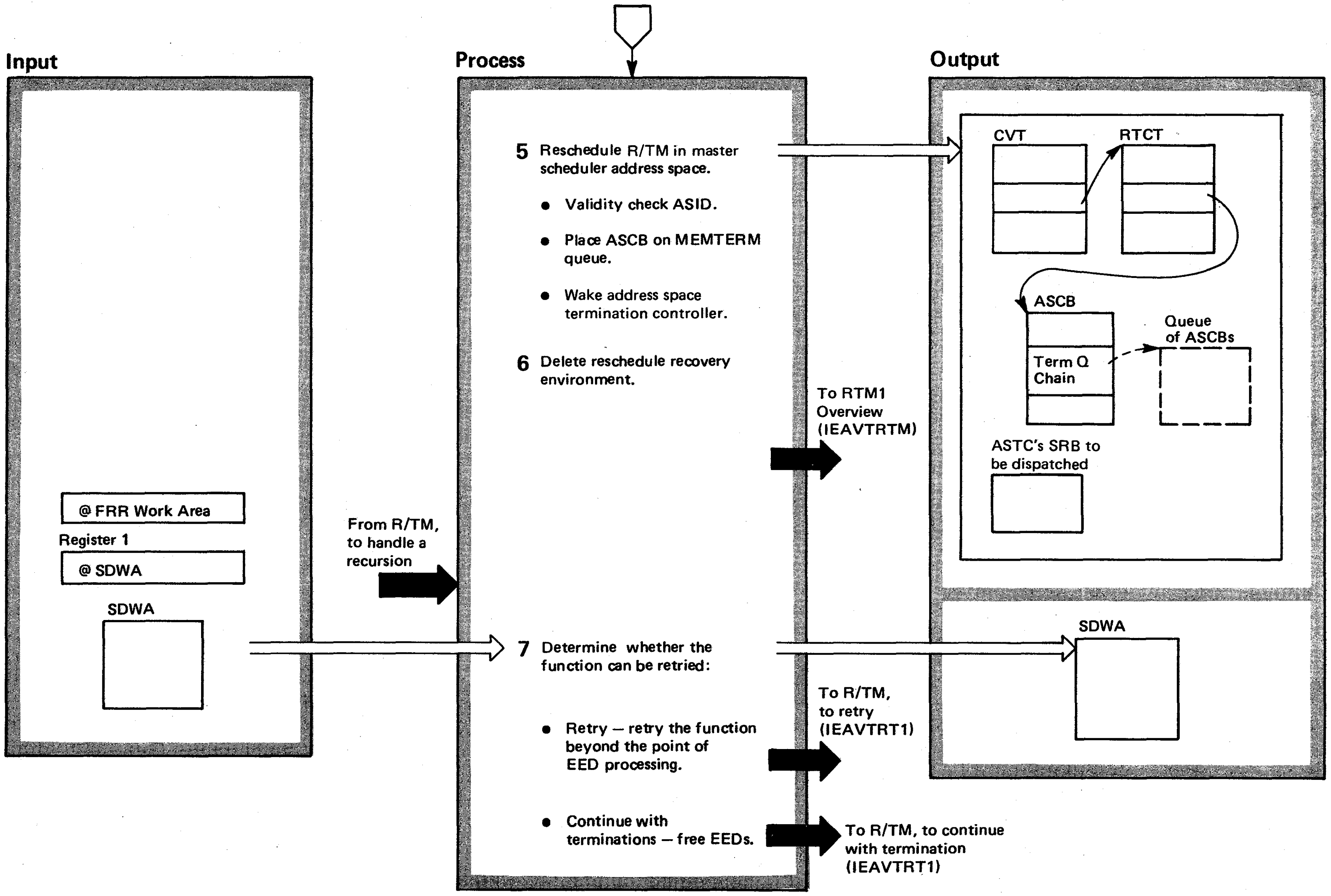
4 Reschedule R/TM in mode of error.

- Schedule RTM2 X'431' (ABTERM).
- Reschedule RTM1 X'433'.

Diagram 22-7. Reschedule RTM1 (IEAVTRTM) (Part 2 of 4)

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

Diagram 22-7. Reschedule RTM1 (IEAVTRTM) (Part 3 of 4)


Input
Process
Output
@ FRR Work Area
Register 1
@ SDWA
SDWA
From R/TM, to handle a recursion
CVT
RTCT
ASCB
Term Q Chain
Queue of ASCBs
ASTC's SRB to be dispatched
To RTM1 Overview (IEAVTRTM)
SDWA
To R/TM, to retry (IEAVTRT1)
To R/TM, to continue with termination (IEAVTRT1)

5 Reschedule R/TM in master scheduler address space.

- Validity check ASID.
- Place ASCB on MEMTERM queue.
- Wake address space termination controller.

6 Delete reschedule recovery environment.

7 Determine whether the function can be retried:

- Retry — retry the function beyond the point of EED processing.
- Continue with terminations — free EEDs.

Diagram 22-7. Reschedule RTM1 (IEAVTRTM) (Part 4 of 4)

Extended Description	Module	Segment
5 RTM1 attempts to schedule the address space termination controller part of R/TM, which resides in the master scheduler address space — if the address space termination function has been requested.		MEMTERM
• For a valid, specified ASID, RTM1 places the corresponding ASCB on the address space termination queue.		
• RTM1 schedules the address space termination controller's SRB to process the ASCB termination queue.		WAKEMTC
6 RTM1 deletes the SLIH mode FRR, if applicable, and the reschedule FRR. If RTM1 had been entered in SLIH mode, recovery now reverts to the scheme of logical phase recovery routines. (See M.O. Diagram RTM1 Recursion Processing (IEAVTRTR) for a description of logical phases.)		RESCHED
7 The reschedule RTM1 function protects itself with an FRR (functional recovery routine). The FRR determines whether the reschedule function can retry past the portion of code where the error occurred, or whether to continue with termination. The FRR requests retry only for errors that occur during processing non-essential to RTM1's handling of the original error; one such example of non-essential processing is EED processing. If the FRR must continue with termination, the FRR cleans the resources used during the reschedule function.	IEAVTRTR	RTMRSFRR
This provides an additional parameter area used by the reschedule RTM1 FRR (RTMRSFRR). This FRR passes a continue with termination request, when entered.		RTMSMFRR

Diagram 22-8. System–Directed Task Termination (IEAVTRTM) (Part 1 of 2)

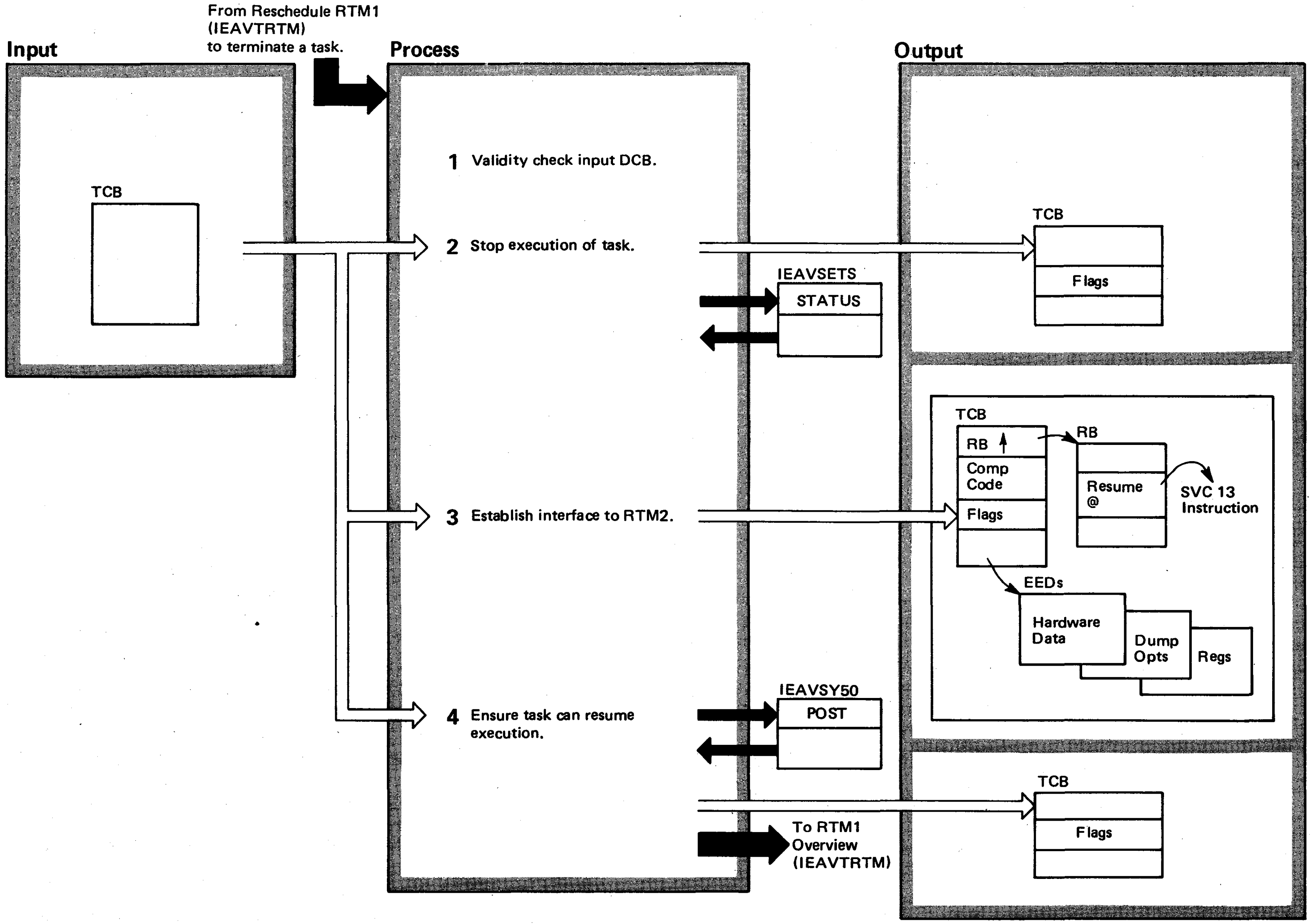

From Reschedule RTM1 (IEAVTRTM) to terminate a task.
Input
Process
Output
TCB
1 Validity check input DCB.
2 Stop execution of task.
3 Establish interface to RTM2.
4 Ensure task can resume execution.
IEAVSETS
STATUS
IEAVSY50
POST
To RTM1 Overview (IEAVTRTM)
TCB
Flags
TCB
RB
Comp Code
Flags
RB
Resume @
SVC 13 Instruction
EEDs
Hardware Data
Dump Opts
Regs
TCB
Flags

Diagram 22-8. System–Directed Task Termination (IEAVTRTM) (Part 2 of 2)

Extended Description	Module	Segment
This illustrates the processing which parallels the ABTERM function of earlier OS systems. Since the task recovery and termination process (RTM2) must operate under the TCB being serviced, RTM1 must modify the task control block structure (TCB/RB) so that the RTM2 (via SVC 13) receives control as an RB on the effected TCB. RTM1 performs validity checking to prevent erroneous modification of key 0 storage and unnecessary ABEND processing. The task must be stopped because in an MP (multiprocessing) environment, the task may be operating on another CPU. The resetting of the tasks non-dispatchability indicators and wait indicators prevents deadlock situations.		
1 First, RTM1 ensures that the task passed as input by the invoker exists on the TCB priority queue of the address space. (RTM1 does not check the priority queue if the "current" task is being terminated.) RTM1 also checks whether or not the task had previously been passed to ABTERM but has not yet executed the SVC 13 instruction. RTM1 bypasses scheduling the ABTERM function if the TCB is invalid or the ABTERM is already in progress.	IEAVTRTM	VALIDCK
2 RTM1 calls the STATUS routine to stop the execution of the task on another CPU in a multiprocessing environment. The task will not be redispatched while RTM1 holds the local lock. For current tasks, no call is necessary since the task has already stopped execution.		CKNONCUR

Extended Description	Module	Segment
3 RTM1 alters the resume address of the task so that when the task subsequently receives control it will execute an SVC 13 instruction to enter RTM2. The information concerning the error resides in the TCB/RB and EED(s) for use by RTM2.		TCBRB
4 These considerations affect the dispatchability of the TCB/RB being terminated: 1) The wait count in the RB. 2) The non-dispatchability flags in the TCB. POST is entered via a branch to reduce the wait count. POST is reissued until it takes the RB out of a wait condition (when the wait count becomes 0). STATUS sets the task forced-dispatchable by resetting all non-dispatchability flags. This function allows for the breaking of deadlock situations caused by routines which set tasks non-dispatchable and neglect to reset them.		SCHDRTM2

Diagram 22-9. Reschedule Locally Locked Task or SRB (IEAVTRTM) (Part 1 of 2)

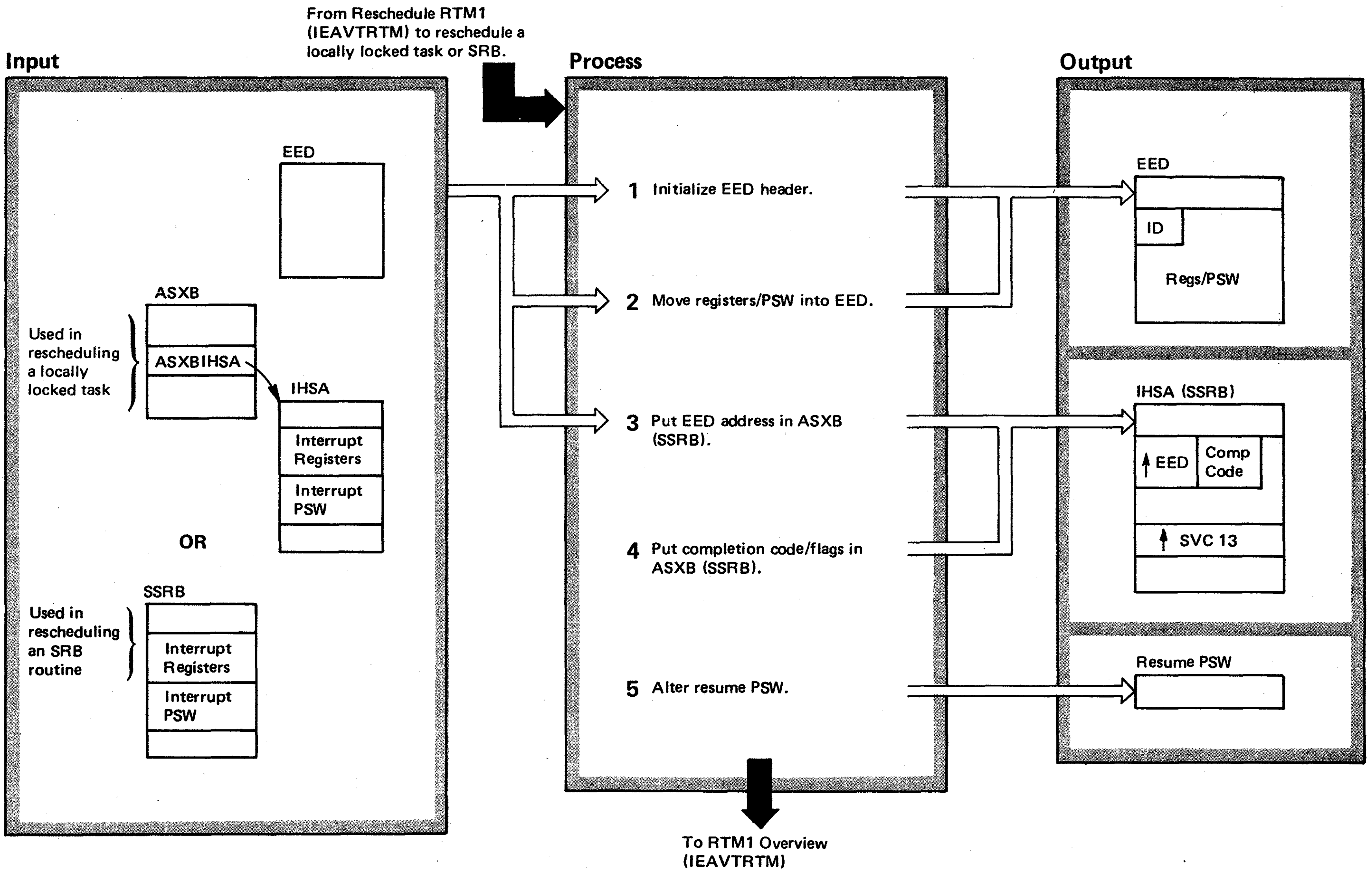
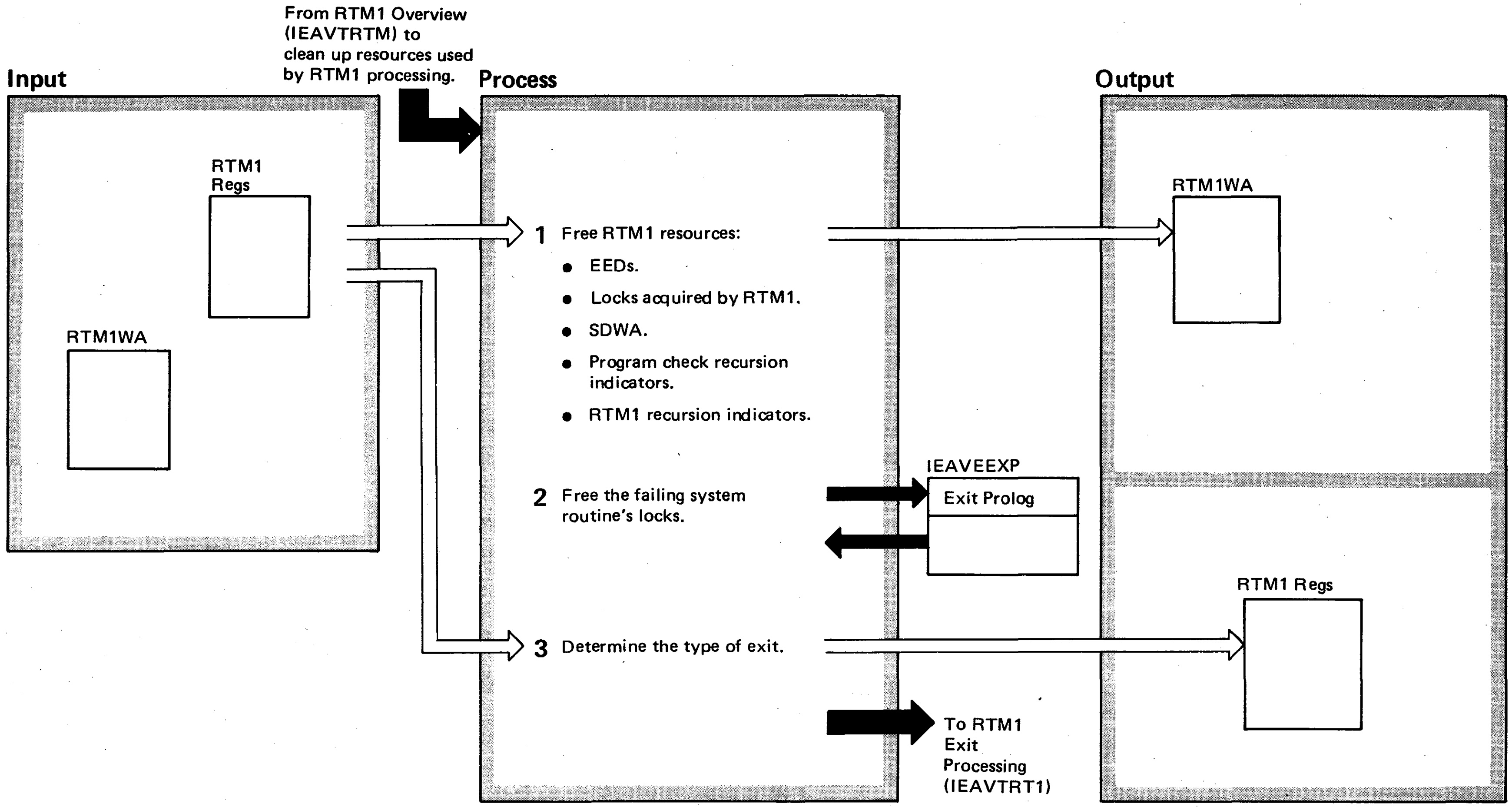

From Reschedule RTM1 (IEAVTRTM) to reschedule a locally locked task or SRB.
Input
EED
ASXB
ASXBIHSA
Used in rescheduling a locally locked task
IHSA
Interrupt Registers
Interrupt PSW
OR
SSRB
Used in rescheduling an SRB routine
Interrupt Registers
Interrupt PSW
Process
1 Initialize EED header.
2 Move registers/PSW into EED.
3 Put EED address in ASXB (SSRB).
4 Put completion code/flags in ASXB (SSRB).
5 Alter resume PSW.
To RTM1 Overview (IEAVTRTM)
Output
EED
ID
Regs/PSW
IHSA (SSRB)
EED
Comp Code
SVC 13
Resume PSW

Diagram 22-9. Reschedule Locally Locked Task or SRB (IEAVTRTM) (Part 2 of 2)

Extended Description	Module	Segment
When an error occurs during page fault processing for a locally locked task (or SRB routine) RTM1 sets the task to be redispatched from the IHSA (or the SSRB) with an SVC 13 instruction as the first instruction to be executed. When the SVC IH subsequently becomes dispatched, it will issue a "CALLRTM TYPE=SVCERR" macro instruction since it would appear that an ineligible routine (i.e. locked task or SRB routine) has issued an SVC.		
1 RTM1 zeroes the EED and sets the I.D. field to indicate a register type.	IEAVTRTM	SCHDRTM1
2 The registers and PSW in the IHSA for a task (IHSAGPRS and IHSACPSW and in the SSRB for an SRB – SSRBGPRS and SSRBCPSW) stored at the time the task (or the SRB) was pre-empted by the page fault are preserved in the EED.		SCHDRTM1
3 RTM1 alters register 0 in the IHSAGPRS (or SSRBGPRS for SRBs) field to point to the EED (this becomes input to the RTM1 upon re-entry).		SCHDRTM1
4 RTM1 places the completion code and options flags in the register 1 slot in the IHSAGPRS (or SSRBGPRS for SRBs) field.		SCHDRTM1
5 RTM1 alters the IHSACPSW (or SSRBCPSW for SRBs) field to point an SVC 13 instruction within the RTM's module (this technique allows the RTM1 to uniquely identify the re-entry as a reschedule function as opposed to another routine issuing the ABEND macro instruction).		SCHDRTM1

Diagram 22-10. RTM1 Clean-up Processing (IEAVTRTM) (Part 1 of 2)

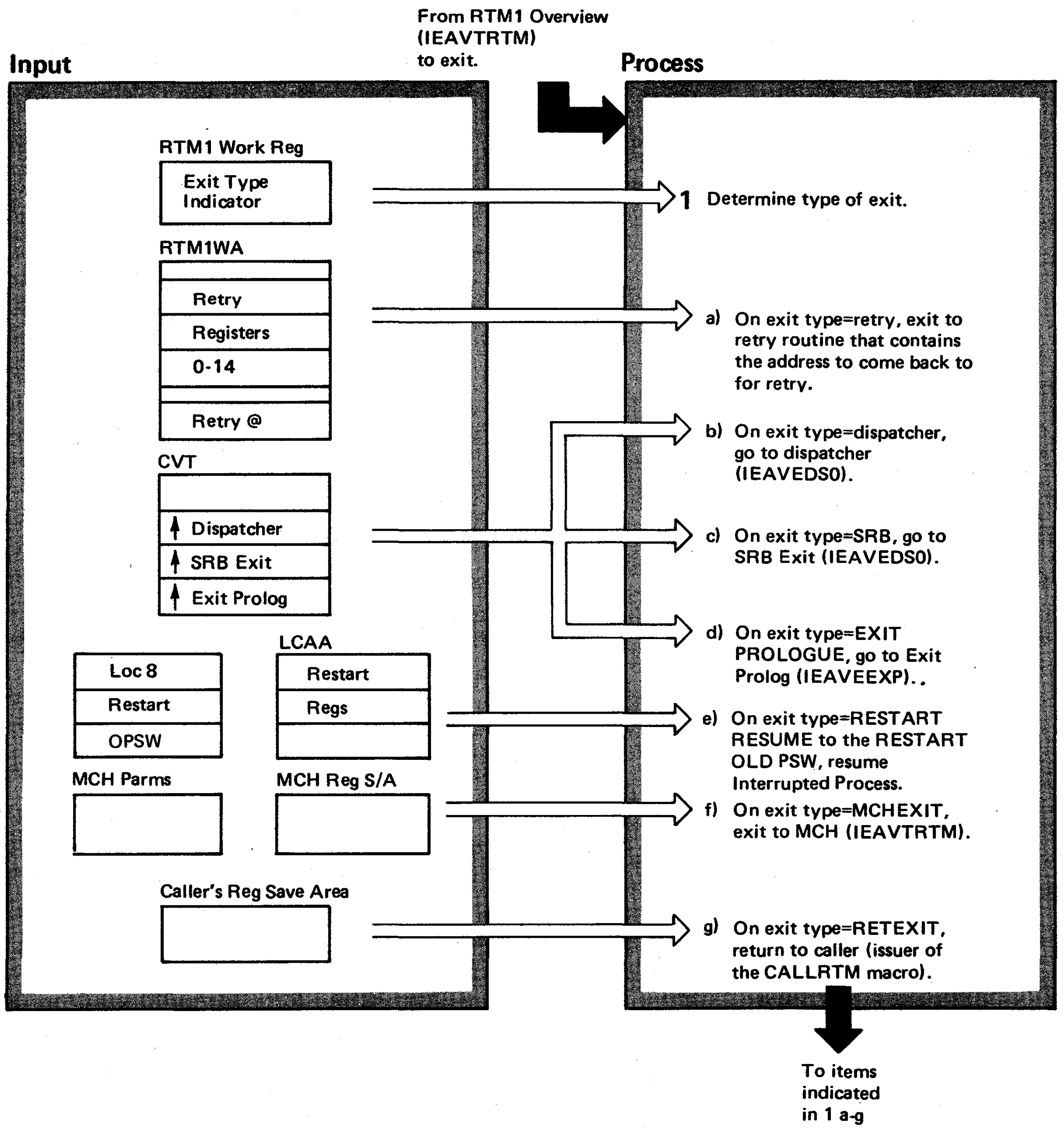


From RTM1 Overview (IEAVTRTM) to clean up resources used by RTM1 processing.
Input
Process
Output
RTM1 Regs
RTM1WA
1 Free RTM1 resources:
● EEDs.
● Locks acquired by RTM1.
● SDWA.
● Program check recursion indicators.
● RTM1 recursion indicators.
RTM1WA
2 Free the failing system routine's locks.
IEAVEEXP
Exit Prolog
3 Determine the type of exit.
RTM1 Regs
To RTM1 Exit Processing (IEAVTRT1)

Diagram 22-10. RTM1 Clean-up Processing (IEAVTRTM) (Part 2 of 2)

Extended Description	Module	Segment
This illustrates the functions performed by RTM1 during clean-up processing.		
1 The clean-up processing frees any locks, EED's or an SDWA acquired during the RTM1 processing, which are no longer needed.	IEAVTRTM	SYSCLEAN
2 Clean-up frees all locks currently held by the failing routine. Exit Prologue (EP Name=IEAVFRLK) performs this function.		
3 Recursion indicators in the RTM1WA or the current FRR are deleted. Control is returned to the entry point/exit point processor with an indication of the type of exit to effect.		EXIT

Diagram 22-11. RTM1 Exit Processing (IEAVTRT1) (Part 1 of 2)



1 Determine type of exit.

a) On exit type=retry, exit to retry routine that contains the address to come back to for retry.

b) On exit type=dispatcher, go to dispatcher (IEAVEDS0).

c) On exit type=SRB, go to SRB Exit (IEAVEDS0).

d) On exit type=EXIT PROLOGUE, go to Exit Prolog (IEAVEEXP)..

e) On exit type=RESTART RESUME to the RESTART OLD PSW, resume Interrupted Process.

f) On exit type=MCHEXIT, exit to MCH (IEAVTRTM).

g) On exit type=RETEXIT, return to caller (issuer of the CALLRTM macro).

Diagram 22-11. RTM1 Exit Processing (IEAVTRT1) (Part 2 of 2)

Extended Description	Module	Segment
RTM1 routines exit from a common exit routine within module IEAVTRT1.		
1 RTM1 exit processing uses the exit type determined by the module IEAVTRTM to perform the appropriate exit procedure, as follows:	IEAVTRT1	IEAVTRTZ
a. Exit processing loads registers 0 through 15 from the RTM1 work area. Register 15 will now contain the retry address. Finally a branch on register 15 is executed.		
b. The dispatcher's exit point is placed in register 15 from the CVTODS field of the CVT. A branch on register 15 is executed.		RT1EXIT2
c. The SRB exit point is placed in register 15 from the CVTSRBRT field of the CVT. A BR 15 instruction is executed.		RT1EXIT4
d. Register 15 is loaded with the contents of the CVTEXPRO field in the CVT. This points to the exit prolog routine, via a BR 15 instruction.		RT1EXIT6
e. Registers 0-15 are loaded from the restart save area (LCCARSGR). A LPSW instruction is issued to cause the restart old PSW to be loaded.		RT1EXIT8
f. A pointer to the interrupt PSW and registers is placed in the MCH parameter list. Register 2-0 (all but Register 1) are reloaded from the MCH save area. A branch on register 14 is executed.		RT1EXITC
g. Registers 0-14 are reloaded from the register save area and a branch on register 14 is executed.		RT1EXITE

Diagram 22-12. RTM2 Overview (IEAVTRT2) (Part 1 of 4)

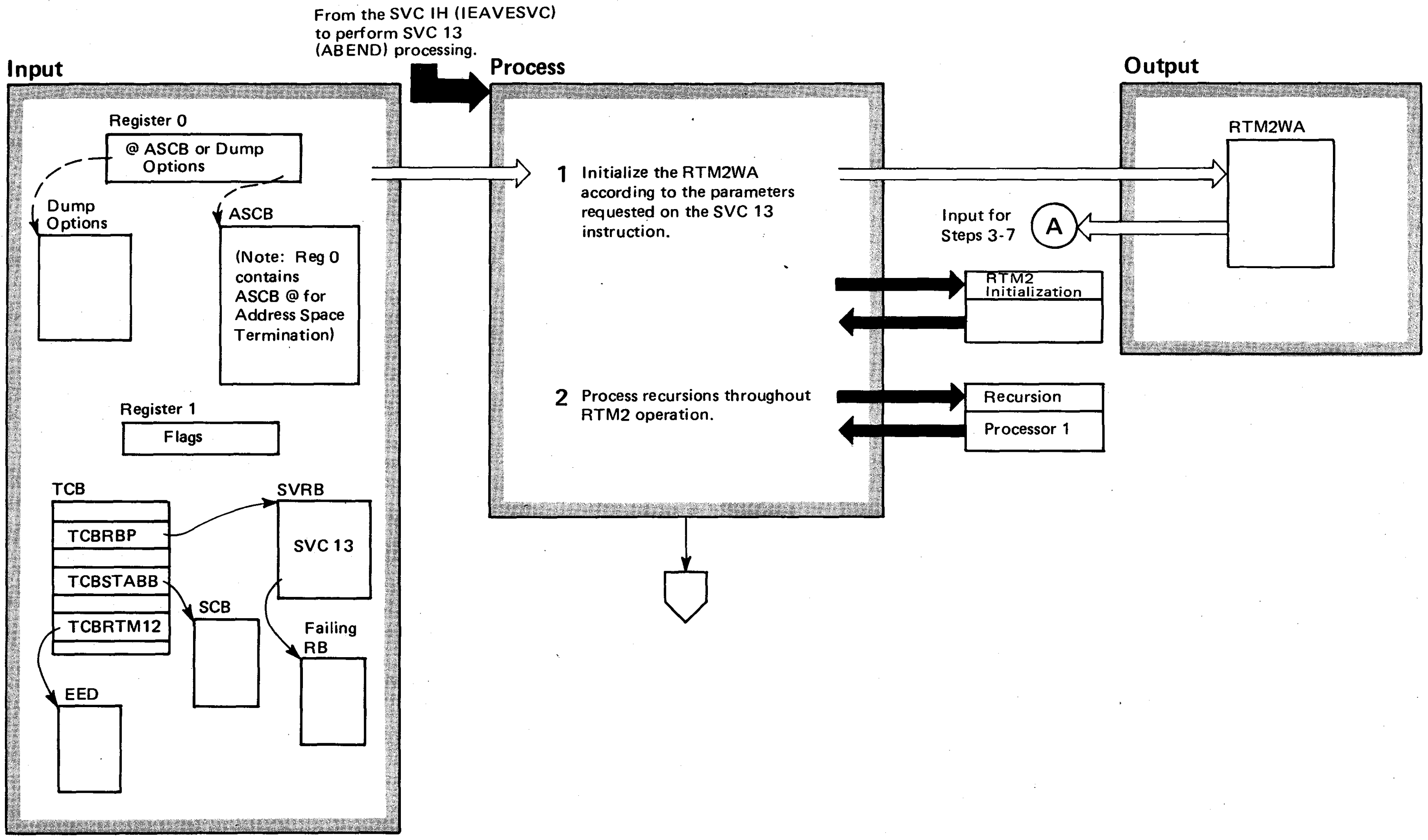

From the SVC IH (IEAVESVC) to perform SVC 13 (ABEND) processing.
Input
Register 0
@ ASCB or Dump Options
Dump Options
ASCB
(Note: Reg 0 contains ASCB @ for Address Space Termination)
Register 1
Flags
TCB
TCBRBP
TCBSTABB
TCBRTM12
SVRB
SVC 13
SCB
Failing RB
EED
Process
1 Initialize the RTM2WA according to the parameters requested on the SVC 13 instruction.
2 Process recursions throughout RTM2 operation.
RTM2 Initialization
Recursion
Processor 1
Output
RTM2WA
Input for Steps 3-7
A

Diagram 22-12. RTM2 Overview (IEAVTRT2) (Part 2 of 4)

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

Diagram 22-12. RTM2 Overview (IEAVTRT2) (Part 3 of 4)

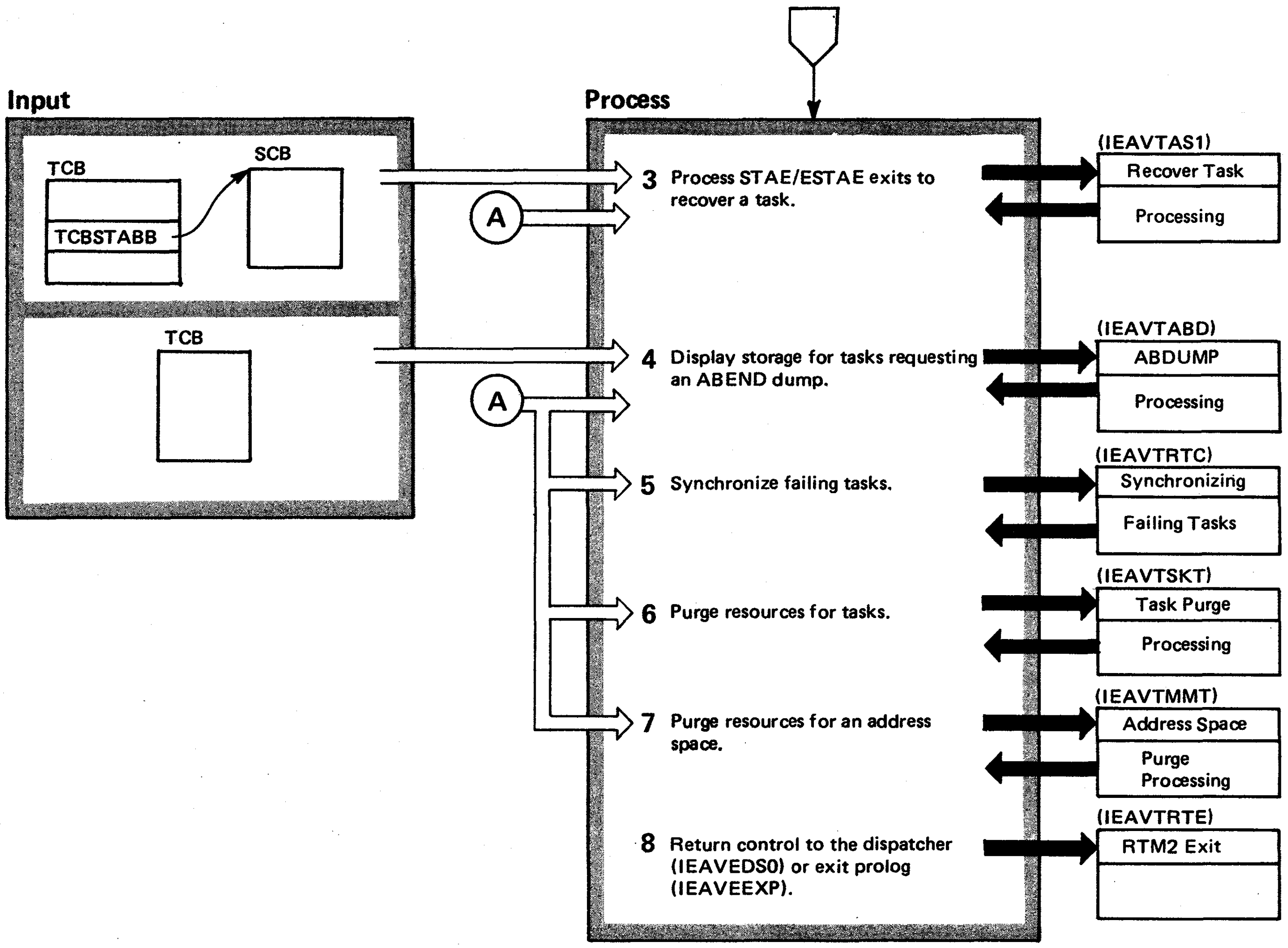

Input
TCB
TCBSTABB
SCB
TCB
A
A
Process
3 Process STAE/ESTAE exits to recover a task.
4 Display storage for tasks requesting an ABEND dump.
5 Synchronize failing tasks.
6 Purge resources for tasks.
7 Purge resources for an address space.
8 Return control to the dispatcher (IEAVEDS0) or exit prolog (IEAVEEXP).
(IEAVTAS1)
Recover Task Processing
(IEAVTABD)
ABDUMP Processing
(IEAVTRTC)
Synchronizing Failing Tasks
(IEAVTSKT)
Task Purge Processing
(IEAVTMMT)
Address Space Purge Processing
(IEAVTRTE)
RTM2 Exit

Diagram 22-12. RTM2 Overview (IEAVTRT2) (Part 4 of 4)

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

Diagram 22-13. RTM2 Initialization (IEAVTRT2) (Part 1 of 2)

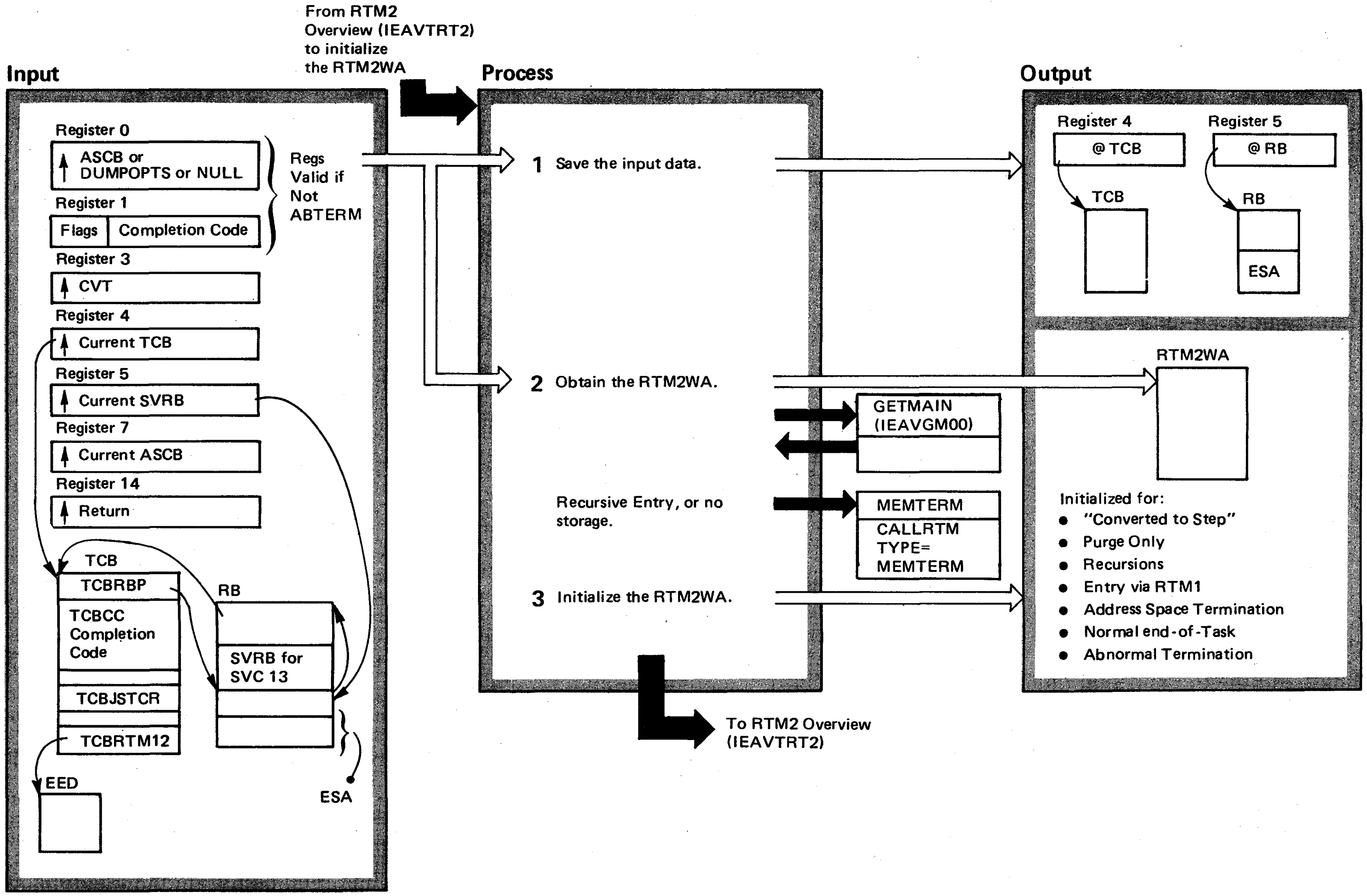
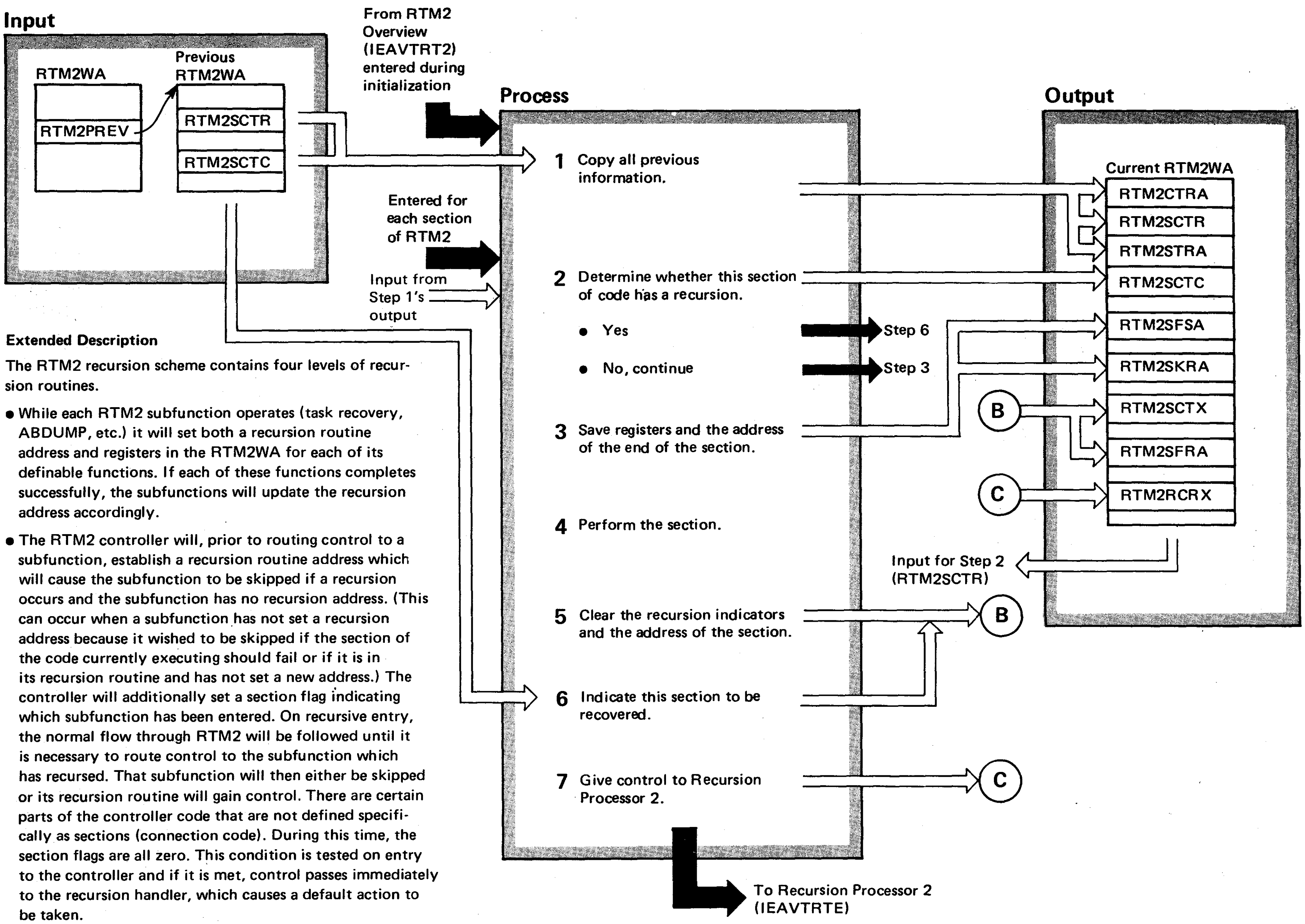


Diagram 22-13. RTM2 Initialization (IEAVTRT2) (Part 2 of 2)

Extended Description	Module	Segment
RTM2 communicates between its various routines via the RTM2WA. RTM2 initialization processing creates and initializes the RTM2WA for subsequent use by the RTM2 routines. The RTM2WA contains the following types of information: • Address of TCB, RB, CVT, ASCB, SDWA. • Registers and PSW at the time of error, and flags indicating system state for ABTERM and ABEND requests. • Machine check information. • DUMP options if any were passed. • Address of any previous workarea, and indicators, for recursive entries.		
Control goes from initialization to the RTM2 controller (represented by the M.O. diagram RTM2 Overview (IEAVTRT2)) to continue processing.		
Register 0 contains (1) the address of ASCB representing the address space to be terminated if address space termination is requested or (2) the address of dump options if dump options were supplied and entry is not via the RTM1 ABTERM function.		
Register 1 contains the completion code and flags indicating the type of request and options if the entry is not via the RTM1 ABTERM function. If entry is via the RTM1 ABTERM function, the dump options, completion code, and type of request, are passed via TCB fields.		
1 Initialization processing saves the input registers and TCB flags in the ESA. Those TCB fields set by RTM1 are cleared to prevent confusion in case of recursion. The TCB fields necessary for recursion tracking are set. Asynchronous exits are blocked. If this is a recursive entry, the recursion flags are copied from previous ESA.	IEAVTRT2	RT2INESA
2 If the ESACTS flag is on, this ABEND is on a jobstep task: RTM2 converted an ABEND to the step level. If so, the work area required for the initial ABEND has been queued to this TCB and no new work area should be acquired. If the flag is off, storage is acquired for an RTM2WA.		RT2GETWA
If it is not possible to obtain storage (RC=4, no virtual, RC=8, no real from GETMAIN), initialization processing passes control to the critical error routine which attempts to take an SVC dump and terminate this address space. This is done since no storage remains in LSQA or SQA, and the termination of the address space (which takes place in the master scheduler's address space) should at least cause SQA to be released, thereby enabling the rest of the system to process.		RT2CRERR RT2TMRY
3 Initialization processing places the critical error routine address in the RTM2WA (RTM2CTRA) and sets an initialization phase recursion indicator (ESAINREC) in the ESA. If this is not a purge — only entry or an entry on a jobstep TCB, the step conversion recursion handler address is also placed in the RTM2WA (RTM2STRA). The initialization processing routine initializes the RTM2WA, using data found originally in the input registers, the TCB, the RB queue, and, if the entry is from RTM1, the extended error descriptors (EEDS). If this is a recursive entry and if the ESAINREC flag is on, initialization processing terminates the address space. If control returns normally from initialization, the ESAINREC flag is reset.		RT2INWA RT2INCNV RT2INCM RT2INEOT RT2INABD RT2INRT1 RT2INMT RT2CYEED RT2MODE RT2INPG RT2INRCR

Diagram 22-14. Recursion Processor 1 (IEAVTRT2) (Part 1 of 2)


Input
RTM2WA
RTM2PREV
Previous RTM2WA
RTM2SCTR
RTM2SCTC
From RTM2 Overview (IEAVTRT2) entered during initialization
Entered for each section of RTM2
Input from Step 1's output
Process
1 Copy all previous information.
2 Determine whether this section of code has a recursion.
● Yes
● No, continue
Step 6
Step 3
3 Save registers and the address of the end of the section.
4 Perform the section.
5 Clear the recursion indicators and the address of the section.
6 Indicate this section to be recovered.
7 Give control to Recursion Processor 2.
To Recursion Processor 2 (IEAVTRTE)
Output
Current RTM2WA
RTM2CTRA
RTM2SCTR
RTM2STRA
RTM2SCTC
RTM2SFSA
RTM2SKRA
RTM2SCTX
RTM2SFRA
RTM2RCRX
B
C
Input for Step 2 (RTM2SCTR)

Extended Description

The RTM2 recursion scheme contains four levels of recursion routines.

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

Diagram 22-14. Recursion Processor 1 (IEAVTRT2) (Part 2 of 2)

Extended Description

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

On recursive entries no attempt is made to determine the cause of the error by these recursion handling routines.

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

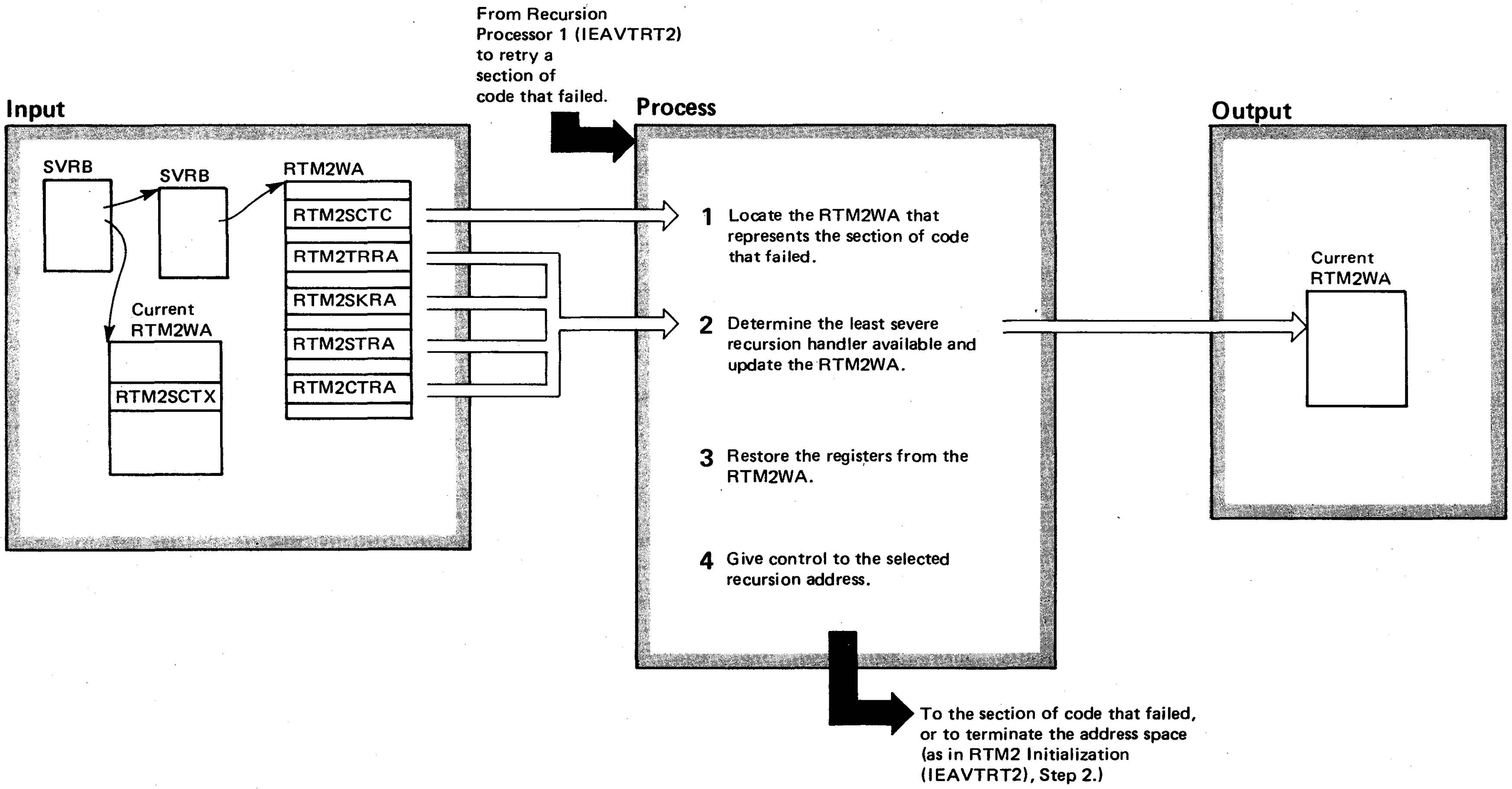
RTM2 uses three sets of flags to maintain control during recursion. RTM2 sets the RTM2SCTC flags as it enters each section and sets them to zero when the section is complete. When one of these flags is set, there is generally a "skip address" which will cause the section to be bypassed if it does suffer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

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

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

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

Diagram 22-15. Recursion Processor 2 (IEAVTRTE) (Part 1 of 2)


From Recursion Processor 1 (IEAVTRT2) to retry a section of code that failed.
Input
SVRB
SVRB
RTM2WA
RTM2SCTC
RTM2TRRA
RTM2SKRA
RTM2STRA
RTM2CTRA
Current RTM2WA
RTM2SCTX
Process
Output
Current RTM2WA
To the section of code that failed, or to terminate the address space (as in RTM2 Initialization (IEAVTRT2), Step 2.)

1 Locate the RTM2WA that represents the section of code that failed.

2 Determine the least severe recursion handler available and update the RTM2WA.

3 Restore the registers from the RTM2WA.

4 Give control to the selected recursion address.

Diagram 22-15. Recursion Processor 2 (IEAVTRTE) (Part 2 of 2)

Extended Description	Module	Segment
The recursion processor 2 function routes control to a recursion handler for the section of code that failed.		
1 The recursion processor 2 locates the RTM2WA for the failed section. (Recursion processor 2 uses this RTM2WA for the processing described in this M.O. diagram.) It does this by matching the RTM2SCTX field passed as input with the RTM2SCTC fields in the various RTM2WA's that represent the failed sections of the code.	IEAVTRTE	RTERCREX
2 The recursion processor 2 checks for a non-zero value, in order of increasing severity, in four fields in the RTM2WA: • RTM2TRRA — skip a small RTM2 function, such as a resource manager routine. • RTM2SKRA — skip a major RTM2 function, such as synchronizing failing tasks or task recovery. • RTM2STRA — terminate the job step • RTM2CTRA — terminate the address space. to find the *least* severe recursion handler.		

Extended Description	Module	Segment
For RTM2TRRA: The recursion processor 2 clears the section indicator in RTM2SCTR and allows the section to retry; it copies the address and registers that will skip the failing section from that RTM2WA to the current RTM2WA passed as input. This enables the recursion processor 2 to skip this section if it fails again. The fields set in the current RTM2WA are RTM2SKRA and RTM2SFSA.		RTESFRE
For RTM2SKRA: The recursion processor 2 *does not* clear the section indicator in RTM2SCTR; this section must be skipped every time it is reached and not be allowed to execute.		
For RTM2STRA and RTM2CTRA: The recursion processor 2 clears *no* fields. These fields contain the addresses of special recursion routines that handle serious errors.		
3 Prior to giving control to the section of code, the recursion processor 2 restores the registers from the RTM2SFSA field for RTM2SKRA processing, or from RTM2RREG for RTM2TRRA processing.		
4 Control goes to the appropriate section of code, using the address selected in step 2.		

Diagram 22-16. Recover Task Processing (IEAVTAS1) (Part 1 of 4)

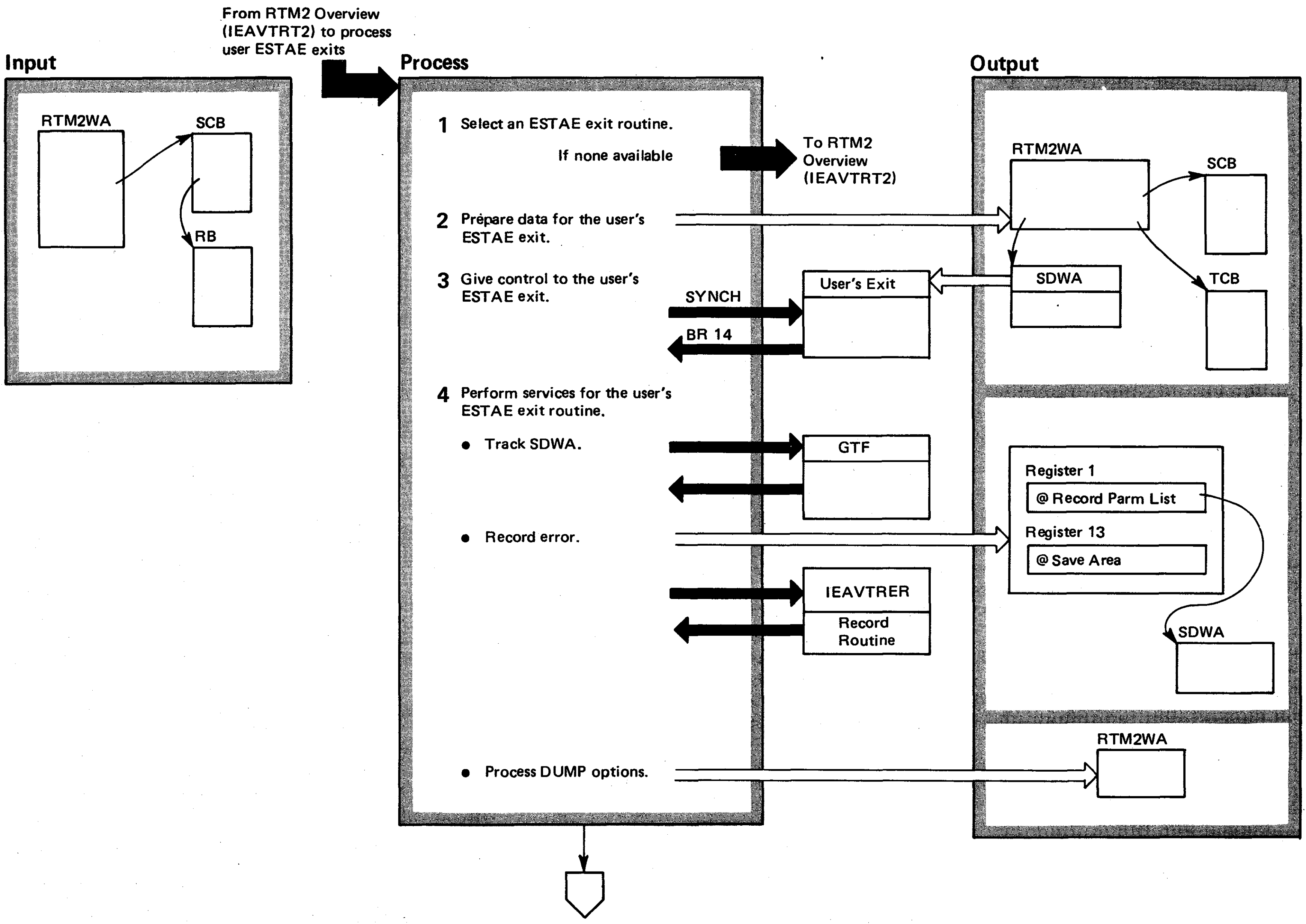
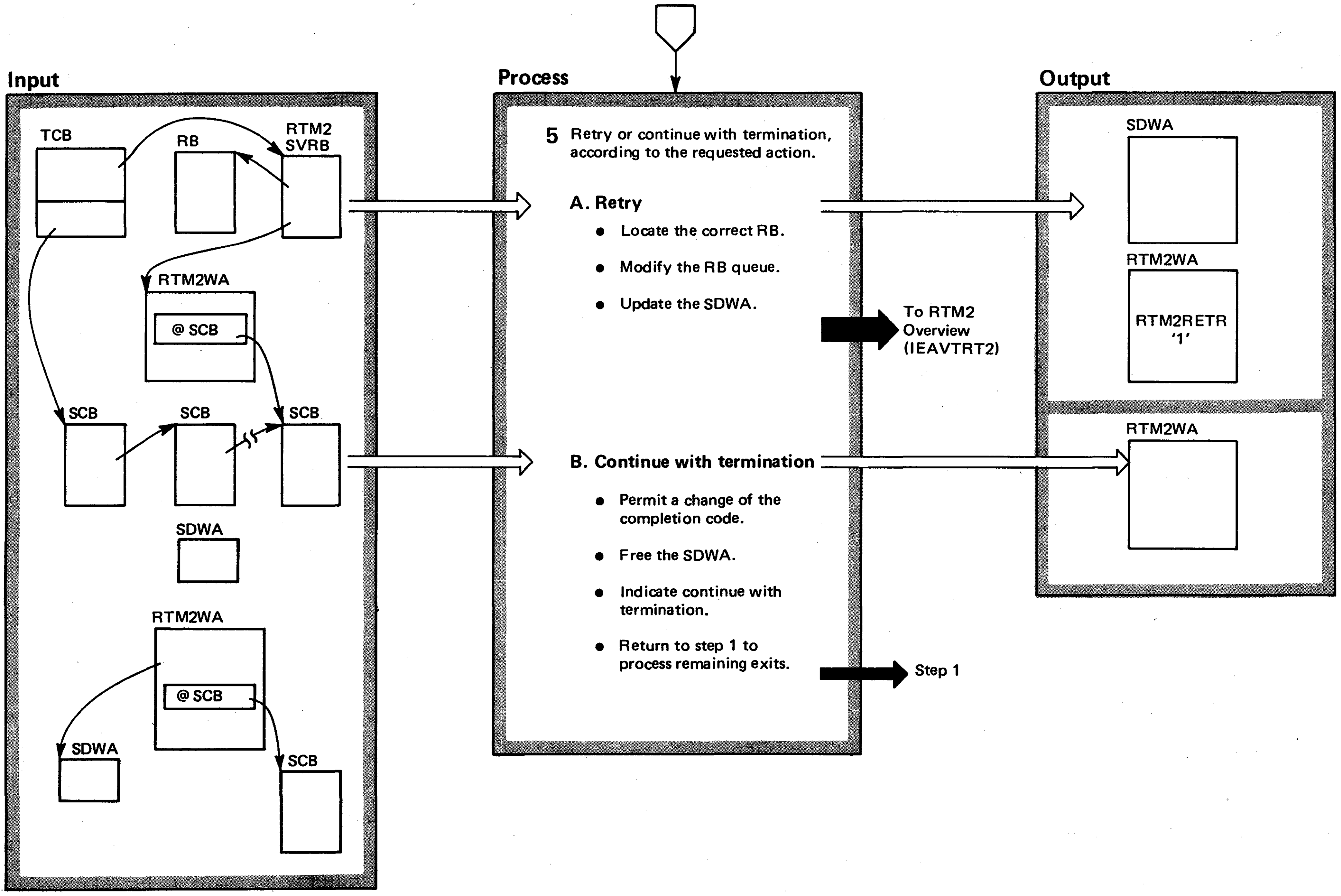

From RTM2 Overview (IEAVTRT2) to process user ESTAE exits
Input
RTM2WA
SCB
RB
Process
1 Select an ESTAE exit routine.
If none available
To RTM2 Overview (IEAVTRT2)
2 Prepare data for the user's ESTAE exit.
3 Give control to the user's ESTAE exit.
SYNCH
BR 14
User's Exit
4 Perform services for the user's ESTAE exit routine.
● Track SDWA.
GTF
● Record error.
IEAVTRER
Record Routine
● Process DUMP options.
Output
RTM2WA
SCB
SDWA
TCB
Register 1
@ Record Parm List
Register 13
@ Save Area
SDWA
RTM2WA

Diagram 22-16. Recover Task Processing (IEAVTAS1) (Part 2 of 4)

Extended Description	Module	Segment
RTM2 routes control to user-written exit routines before it terminates a task. These exit routines – either STAE (specify task asynchronous exit) or ESTAE (extended STAE) – receive control to attempt to recover an abnormally termination task. (See the M.O. diagram STAE/ESTAE Processing (IEAVSTA0) for a description of how the user creates a STAE control block (SCB)). See the publication "Supervisor Services and Macro Instructions", order number GC28-0683, for a description of how a user creates an ESTAE routine.)		
RTM2 selects an ESTAE/STAE routine from the SCB queue, and branches to it to allow it to process. If the terminating task can recover after the ESTAE/STAE routine processes, RTM2 will perform any processing necessary for a retry condition, and the terminating task will resume processing. Otherwise, the terminating task will be terminated.		
RTM2 places diagnostic information in the SDWA during ESTAE/STAE processing.		
1 RTM2 searches the SCB queue to select the exit to be given control. The searching sequence follows:	IEAVTAS1	FINDSCB
• On initial entry, the most recently established exit will be selected. • During "percolation", (a previously selected exit has not elected to retry) – the next exit on the queue will be selected. • During "percolation" only one STAE (as opposed to ESTAE) will be selected, all others will be bypassed. • During TERM processing, only those exits with the TERM option (TERM=YES on ESTAE macro instruction) will be selected. • If the queue is exhausted with no exit requesting retry, control returns to RTM2 and the task will be terminated.		
2 RTM2 initializes some fields in the internal RTM2WA (RTM2 work area) to ensure the accuracy of the SDWA during percolation.		WKUPDAT
RTM2 obtains and initializes an SDWA with information that will aid the user in diagnosing the error.		SDWAINIT
User options indicated on the ESTAE macro instruction will be performed. Asynchronous exit processing may be blocked and active I/O may be halted or quiesced. I/O options will be performed only for the first exit selected; all subsequent exits will receive an indication of I/O status.		USEROPTS
3 RTM2 initializes parameter registers for the exit routine. Additionally, RTM2 sets the interface with the SYNCH macro (used to give control to the exit).		EXITINTR
4 On return from the exit routine, RTM2 traces the SDWA, or return information if no SWDA was obtained, via the HOOK macro.	IEAVTAS2	GTFHOOK
RTM2 writes the SDWA to SYS1.LOGREC via the RECORD macro if so requested by the user exit, constrained only by SDWA's existence and availability.		RCRDSDWA
RTM2 initializes the RTM2 work area with user dump options if any exist. RTM2 combines any dump parameters with existing options; it adds storage ranges to the end of the existing storage range list, wrapping around to the top again if necessary. (A maximum of four storage ranges can be accumulated.) If the user requested no dump, RTM2 zeroes existing options.		DUMPOPTS

Diagram 22-16. Recover Task Processing (IEAVTAS1) (Part 3 of 4)



5 Retry or continue with termination, according to the requested action.

A. Retry

- Locate the correct RB.
- Modify the RB queue.
- Update the SDWA.

B. Continue with termination

- Permit a change of the completion code.
- Free the SDWA.
- Indicate continue with termination.
- Return to step 1 to process remaining exits.

Diagram 22-16. Recover Task Processing (IEAVTAS1) (Part 4 of 4)

Extended Description	Module	Segment
5A If retry can be performed (this is not term exit processing), RTM2 selects a retry RB. For STAE/ESTAE retry, the SCB contains the RB address. For ESTAR retry, RTM2 uses the oldest RB. For STAI/ESTAI, RTM2 performs retry under the PRB for the last STAE/ESTAE or STAI/ESTAI exit routine if one exists. Otherwise, RTM2 purges the RB queue until only PRBs remain and the STAI/ESTAI retry routine will run under the newest PRB left on the queue.	IEAVTAS3	FINDRB
RTM2 prepares the RB queue for retry. Resources are purged and open, embedded data sets are closed. RBs to be purged (those between the retry RB and the ABEND SVRB) have their resume PSW pointed to EXIT and their wait count zeroed.		RBPRGE
If register update was requested on the retry, the retry register values are inserted to ensure that the correct registers are passed to the retrying RB. If register update was not requested, RTM2 initializes error registers to be passed to the retry RB. In either case, if a dump is also requested on the retry, the register and PSW fields in the dump will contain the retry information rather than the values at entry to ABEND. The registers and PSW at entry to ABEND can still be found in the RTM2 work area. This work area resides in LSQA and is pointed to by the TCBRTWA field of the TCB.		RTRYSDWA
According to the user's request, RTM2 either updates the SDWA to be passed to the retry routine, or frees it. Task Recovery returns control to RTM for further preparation for retry.		
5B RTM2 saves information to be passed to the next exit during percolation (changed completion code or a serviceability indicator) in the RTM2 work area and frees the SDWA. In addition, RTM2 initializes percolation information in the RTM2 work area.	IEAVTAS3	SCBPERC

Diagram 22-17. ABDUMP Processing (IEAVTABD) (Part 1 of 4)

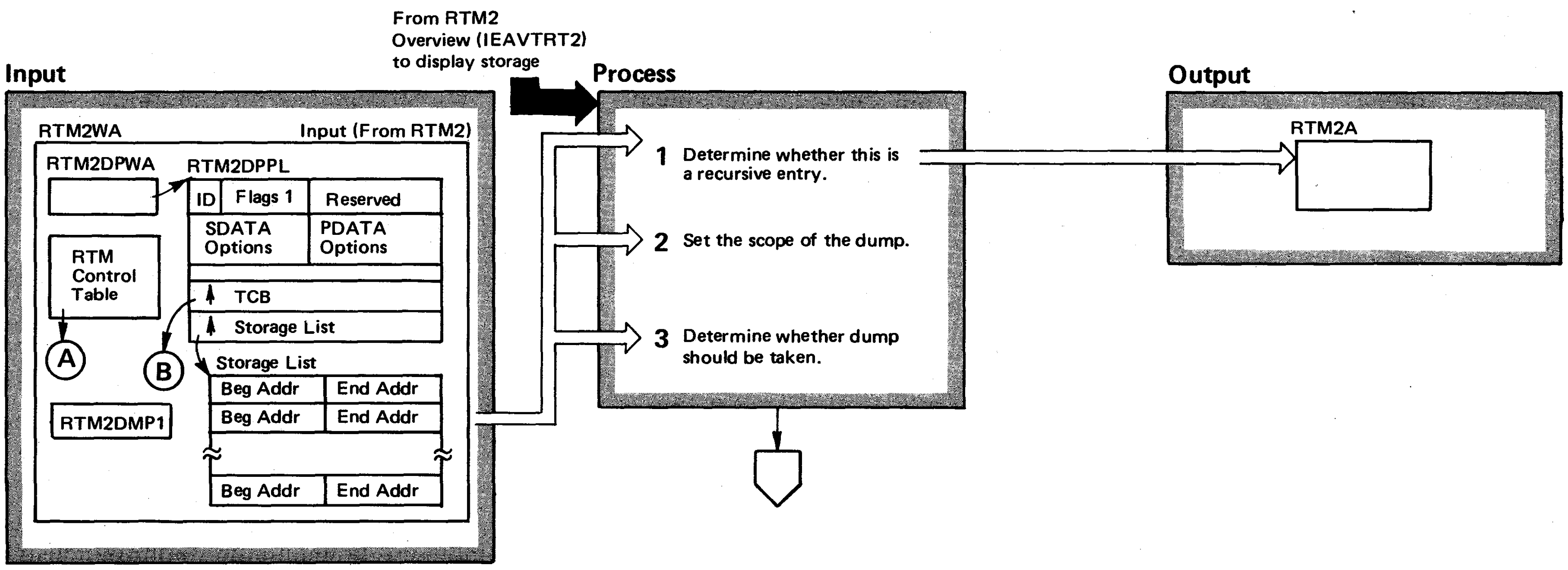


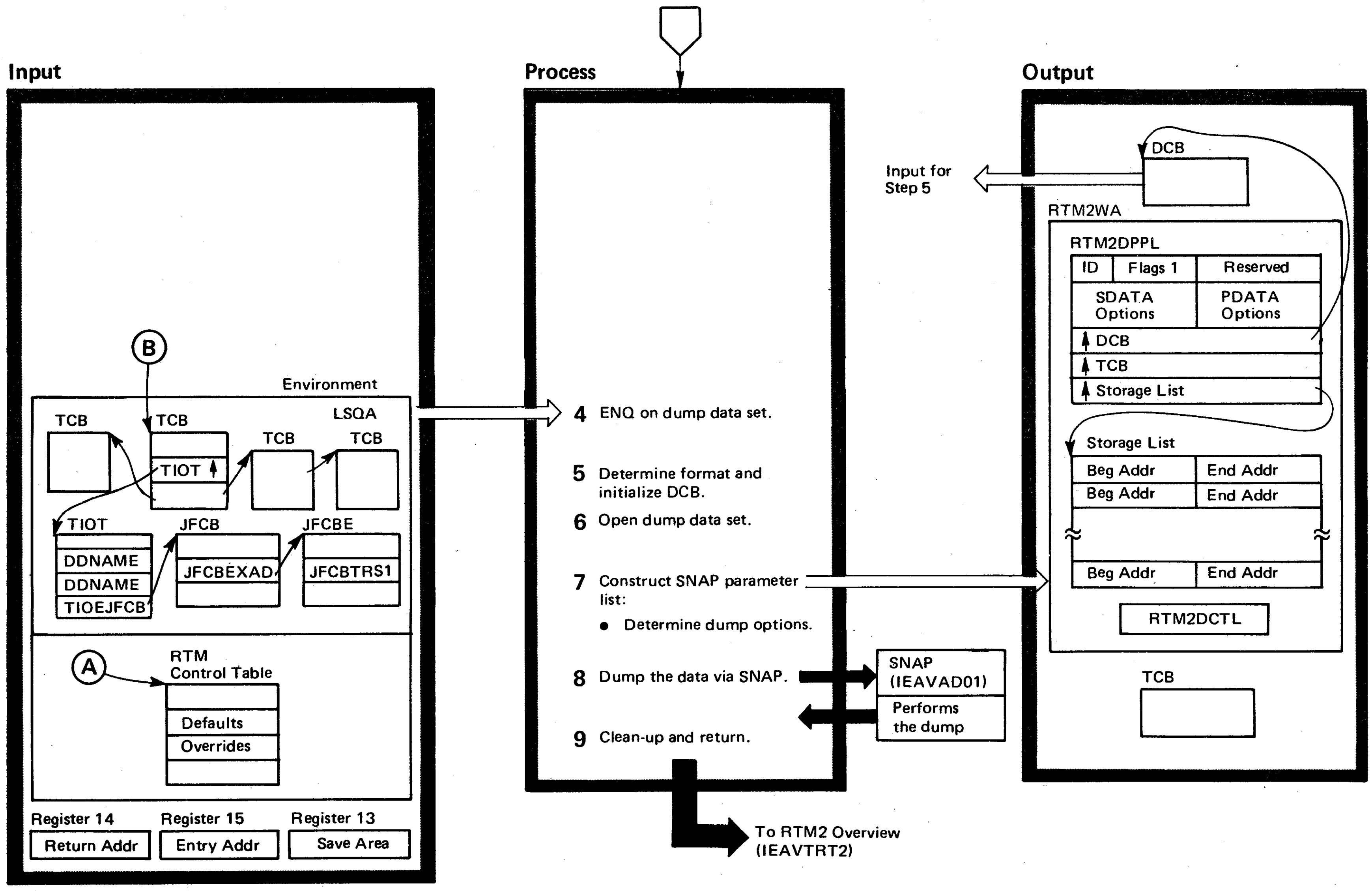
Diagram 22-17. ABDUMP Processing (IEAVTABD) (Part 2 of 4)

Extended Description	Module	Segment
Terminating tasks can request a storage display. RTM2 provides the dump via ABDUMP processing. The RTM2WA contains the dump options for the terminating task; ABDUMP processing checks these options and prepares the dump data set and constructs a SNAP parameter list (for the actual dump), and gives control to SNAP processing (see the M.O. diagram SNAP Dump Processing (IEAVAD01) for the description of SNAP's operation).		
1 ABDUMP protects itself from recursions by setting indicators to denote external functions in control, and to denote the completion of external functions. These indicators follow:	IEAVTABD	ADRECOV

External Function	In Control	Completed
Enqueue for dump resource	RTM2EENQ	RTM2DENQ
GETMAIN for DCB	RTM2EGET	RTM2DGET
OPEN dump data set	RTM2EOPN	RTM2DOPN
SNAP dump	RTM2ESNP	RTM2DSNP
CLOSE dump data set	RTM2ECLS	RTM2DCLS
Free DCB storage	RTM2EFRM	RTM2DFRM
Dequeue for dump resource	RTM2EDEQ	RTM2DDEQ
QMNGRIO to determine dump format	RTM2EQMN	–
First TCB dumped	–	RTM2DFTK

Extended Description	Module	Segment
ABDUMP turns the in control' indicators off when control returns from the external functions.		
When a recursion occurs, ABDUMP checks the first set of indicators (the 'in control' set) to determine if an external function had control. If an external function had control, control goes to a clean up subroutine, ADRCLN, to perform the necessary clean up.		
If an external function did not have control, ABDUMP determines the last completed function, initializes the new RTM2WA with information from the previous work area, and passes control to the appropriate routine in ABDUMP to continue processing.		
This subroutine performs the necessary clean up, requests an SVC dump, and gives control back to the caller.		ADRCLN
2 The scope of a dump can either be a single task if RETRY with dump has been requested from a ESTA exit, or if the task has a subtask which is a jobstep task; a failing task tree, if no recovery from the ABEND was accomplished, or a jobstep tree if no recovery was accomplished and the ABEND is a "step" ABEND. If it is a RETRY with dump (RTM2DREQ=1 and RTM2RETR=1) situation or if the task has a subtask which is a jobstep, the RTM2DMP1 flag is set to 1 and the current TCB address is placed in the TCB field of the dump parameter list in the RTM2WA (SNPTCBA). If it is not a "step" ABEND from a subtask of a step (RTM2STPT=0), the current TCB address is again placed in the SNPTCBA, otherwise the address of the jobstep TCB is placed in the field.	IEAVTRTC	RTCADINT
3 The RTCT (recovery termination control table), bits RTCTISAB and RTCTISYU, is checked to determine whether a SYSABEND or SYSUDUMP dump should be taken. If so, the TIOT is scanned for a SYSABEND or SYSUDUMP ddname. If neither is found, control returns to RTM2 with X'00' in RTM2SNCC. If no dump is to be taken, control also returns to RTM2 with '00' in RTM2SNCC.	IEAVTABD	ADDSCAN

Diagram 22-17. ABDUMP Processing (IEAVTABD) (Part 3 of 4)



4 ENQ on dump data set.

5 Determine format and initialize DCB.

6 Open dump data set.

7 Construct SNAP parameter list:

- Determine dump options.

8 Dump the data via SNAP.

9 Clean-up and return.

Diagram 22-17. ABDUMP Processing (IEAVTABD) (Part 4 of 4)

Extended Description	Module	Segment
4 The dump data set is enqueued upon with the option RET=HAVE, a major name of SYSIEA01 and a minor name of IEA. On a non-zero return code from ENQ, SVC dump is issued and control returns to RTM2.		ADENQ
5 Storage for the DCB and parameter list is obtained from subpool 230. The DCB is initialized with DSORG=PS, MACRF=W, RECFM=VBA, and DDNAME as defined in the TIOT. The format of the dump is determined from information in the JFCB and JFCBE. If condensed dump is requested, the DCB is initialized with LRECL=209; if a standard dump is requested, LRECL=125.		ADITCB
6 The DCB is opened in TCB key (via MODESET). If open is unsuccessful (DCBOFOPN=0), message IEA030I 'OPEN FAILED FOR DUMP DATA SET FOR JS)' is routed to the programmer. Control returns to RTM2 with a return code of 4.		ADOPEN ADOPFAIL
7 The dump options for SNAP are determined from the options passed by RTM2, the installation default options (specified in PARMLIB members IEAABD00-SYSABEND and IEADMP00-SYSUDUMP) or the installation override options (specified via the CHNGDUMP operator command). When more than one of these groups is available, the order of selection is as follows: 1. The options specified via the CHNGDUMP operator command completely override the options specified via the PARMLIB members or passed by RTM2. 2. Lacking CHNGDUMP options, those options passed from RTM2 and merged with the options specified in the PARMLIB members (if available) will control the content of the dump. 3. If no options were passed by RTM2, the PARMLIB options will define the dump contents. However, if PARMLIB options are not present, CHNGDUMP options are not present and no options were passed by RTM2, no dump will be provided. If an ABEND is in progress, different options are selected for TCB's other than the input TCB to prevent the display of redundant data.		ADETOPT
8 After the input TCB has been dumped, a check is made to determine whether an ABEND is in progress (RTM2DMP1=0). If so, STATUS is issued to prevent the subtasks from terminating during dump processing. The subtasks of the abending task are dumped followed by the mother task. TCBFS is set to 1 to indicate the task has been dumped on an ABEND. On a non-zero return code from SNAP, message IEA9121 'RECOVERY/TERMINATION DUMP FAILED' is issued and control returns to RTM2 with return code from SNAP stored in RTM2SNCC. Return codes from SNAP are: 0 – successful completion. 4 – DCB not opened, undefined page reference on DCB. 8 – TCB not valid, undefined page reference on TCB, insufficient storage, invalid parameter list, a subtask is a jobstep TCB, read for JFCB or JFCBE failed and the dump was canceled. 12 – DCB type incorrect, DCB incompatabilities with options specified on dump related DD statement.		ADSNAP ADSPFAIL ADTSLO
9 Close dump data set, free DCB storage, turn off the dumped flag indicator (TCBFS=0) if an ABEND was in progress, dequeue from dump data set, and set subtasks dispatchable if an ABEND was in progress.		ADCLEAN

Diagram 22-18. Synchronize Failing Tasks (IEAVTRTC) (Part 1 of 2)

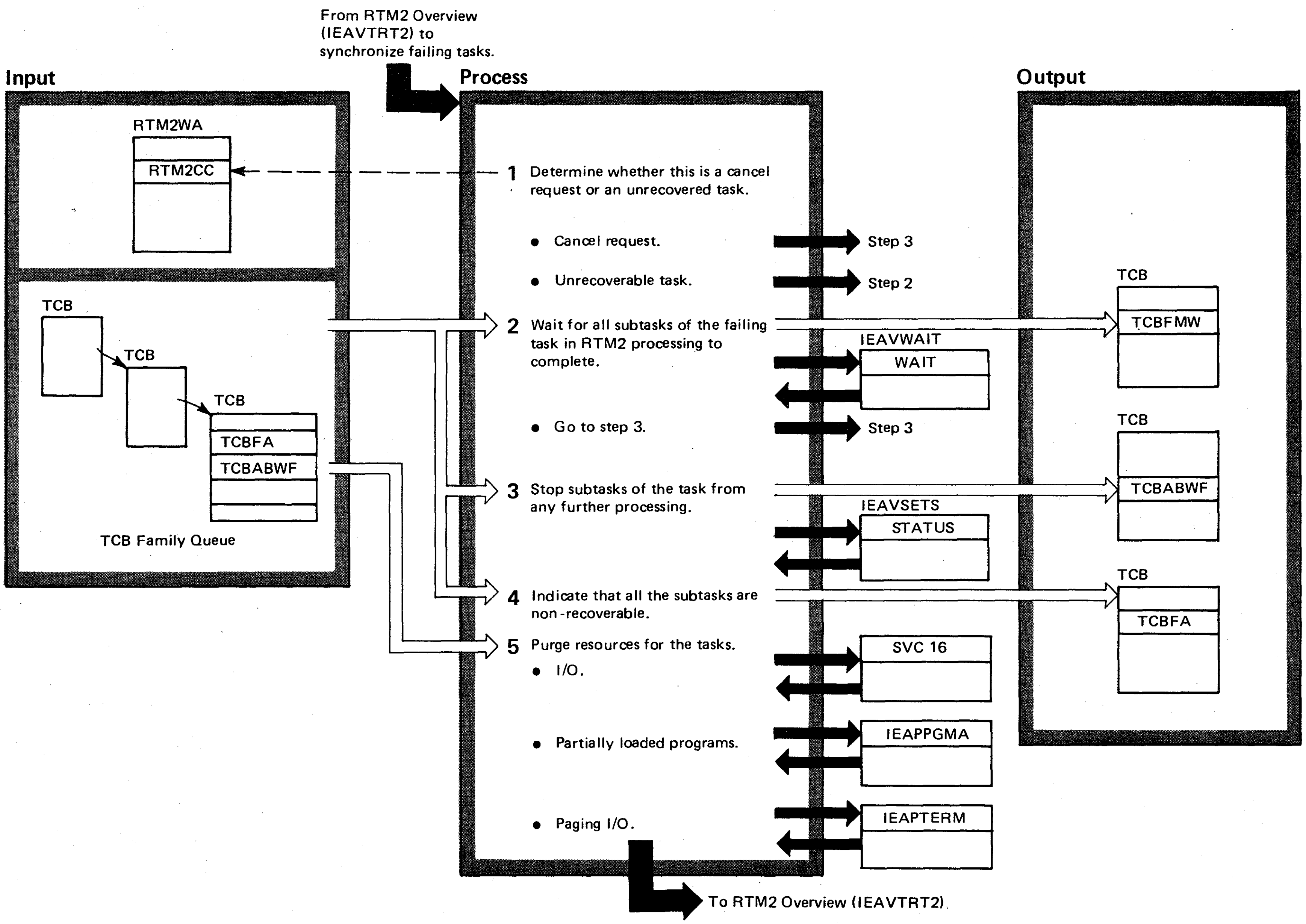

From RTM2 Overview (IEAVTRT2) to synchronize failing tasks.
Input
Process
Output
RTM2WA
RTM2CC
TCB
TCB
TCB
TCBFA
TCBABWF
TCB Family Queue
1 Determine whether this is a cancel request or an unrecovered task.
● Cancel request.
Step 3
● Unrecoverable task.
Step 2
2 Wait for all subtasks of the failing task in RTM2 processing to complete.
IEAVWAIT
WAIT
● Go to step 3.
Step 3
3 Stop subtasks of the task from any further processing.
IEAVSETS
STATUS
4 Indicate that all the subtasks are non-recoverable.
5 Purge resources for the tasks.
● I/O.
SVC 16
● Partially loaded programs.
IEAPPGMA
● Paging I/O.
IEAPTERM
TCB
TCBFMW
TCB
TCBABWF
TCB
TCBFA
To RTM2 Overview (IEAVTRT2)

Diagram 22-18. Synchronize Failing Tasks (IEAVTRTC) (Part 2 of 2)

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

Diagram 22-19. Task Purge Processing (IEAVTSKT) (Part 1 of 4)

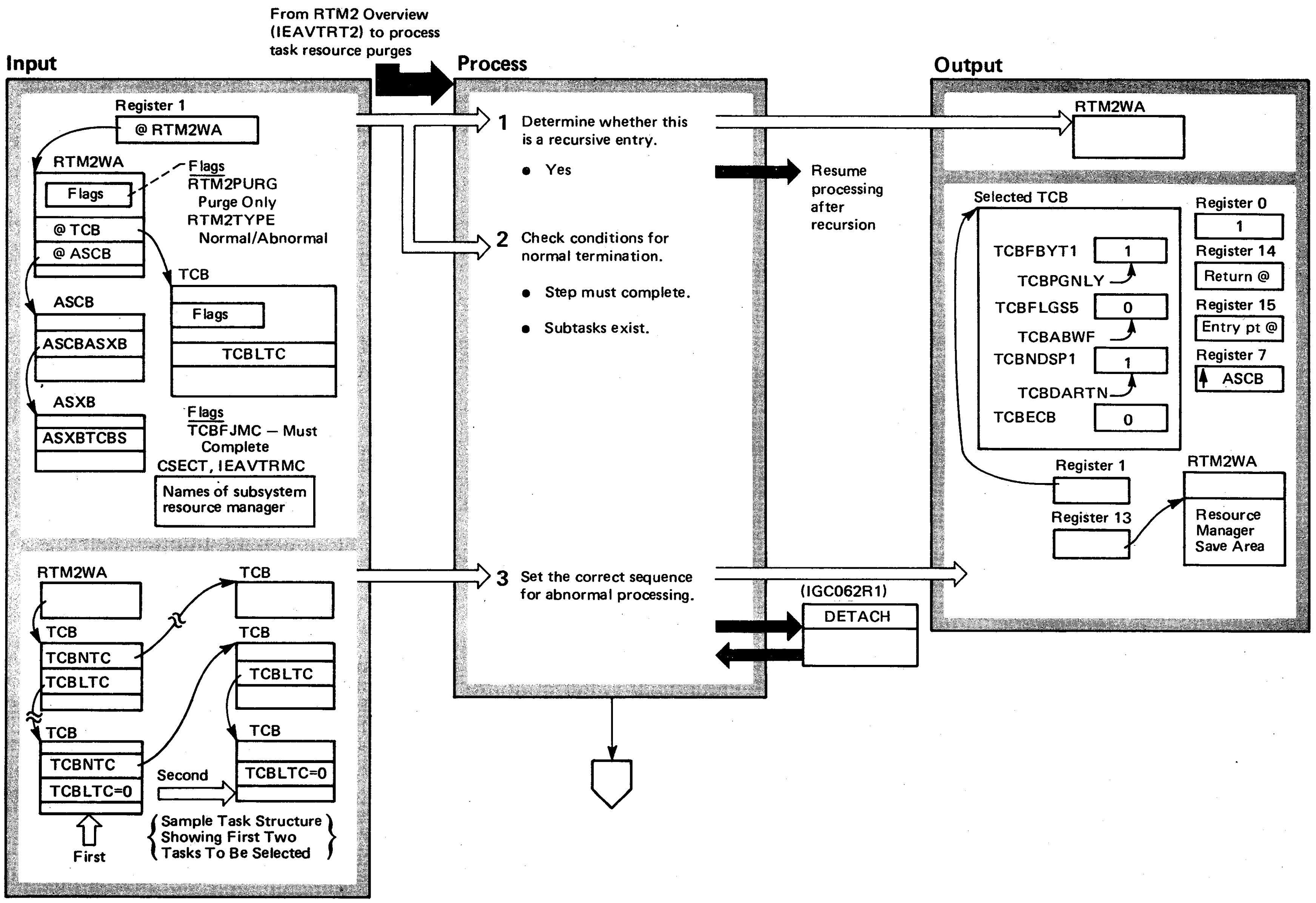


Diagram 22-19. Task Purge Processing (IEAVTSKT) (Part 2 of 4)

Extended Description	Module	Segment
Task purge processing removes the resources used by a task. RTM2 uses the task purge processing function to route control sequentially to installation-defined and IBM-defined resource manager routines to remove their task related resources.		
Task purge processing will remove the resources of the lowest task in the TCB family queue first, and then ascend the queue to the current task, removing their resources.		
Task purge processing receives control from the mainline RTM2 routine, IEAVTRTE, shown as M.O. diagram RTM2 Overview (IEAVTRT2). Input for task purge processing comes from M.O. diagram RTM2 Initialization (IEAVTRT2) which shows the creation and initialization of the RTM2WA.		
1 Task purge processing performs recursion processing, as described in M.O. diagram Recursion Processor 1 (IEAVTRT2).	IEAVTSKT	
The RTM2TRRA field contains the addresses of routines that handle recursions for processes in steps 3, 4, and 5.		
• If a CANCEL recursion occurs for step 3, restart step 3 by selecting the lowest task in the family and detaching it. For any other type of recursion, terminate the address space.		
• If a subsystem resource manager fails, skip the failing subsystem resource manager on a recursive entry. If more than 2 failures occur, skip all the subsystem resource managers, and go to step 5.		
• If an IBM-defined resource manager fails, skip it on any recursive entries and continue processing the others.		

Extended Description	Module	Segment
2 For a normally terminating task, task purge processing checks the terminating task for "step must complete" status, for open data sets, and for existing subtasks.		
• For tasks having "step must complete" status, terminate with an E03 ABEND code.		
• If subtasks exist, task purge processing terminates the task being terminated with an X'A03' ABEND code. RTM2 will then regain control as a result of the SVC 13 instruction issued to terminate the task.		
3 The terminating task may have active subtasks. In this case, task purge processing follows down the TCBLTC chain until it finds the lowest TCB (as indicated by a 0 in TCBLTC). Task purge processing then issues a DETACH (see the Task Management section for a description of DETACH processing) for that TCB, with an indicator to perform termination purging. DETACH will terminate the task if it is still active. Task purge processing detaches all the subtasks, and then purges the resources for the current task.		

Diagram 22-19. Task Purge Processing (IEAVTSKT) (Part 3 of 4)

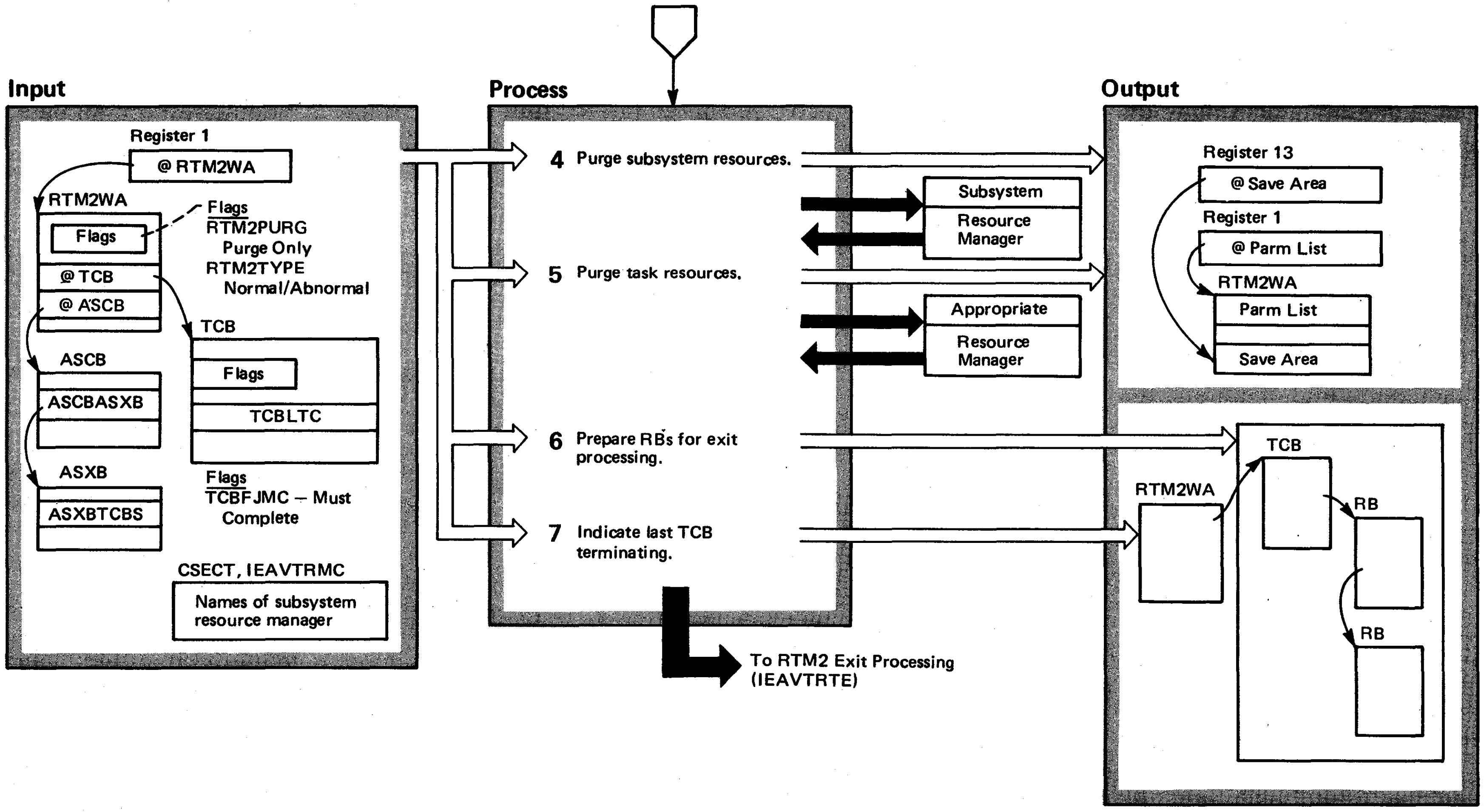


Diagram 22-19. Task Purge Processing (IEAVTSKT) (Part 4 of 4)

Extended Description	Module	Segment

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

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

Task purge processing gives control sequentially to IBM-defined resource manager routines to free task related resources. These routines are called in the following sequence:

1)	Data Management	IFG0TC0A
2)	Timer	IEAVRTI1
3)	Type 1 Message	IEAVTPMT
4)	SPIE	IEAVSPIE
5)	ENQ/DEQ	IEAVENQ2
6)	WTOR	IEECVPRG
7)	Region Control Task	IEAVAR07
8)	VTAM	ISTRAMA1
9)	TCAM	IEDQOT01
10)	Subsystem Interface	IEFJRECM
11)	TIOC	IEDAY8
12)	POST	IEARPOST
13)	Real Storage Management	IEAVTERM
14)	IQE	IEAVEEEP
15)	3850 Mass Storage System	SSCRMCR
16)	ENQ RM	IEAVENQ2
17)	Type 1 Message	IEAVTPMT
18)	SRB Purge	IEAVPD0

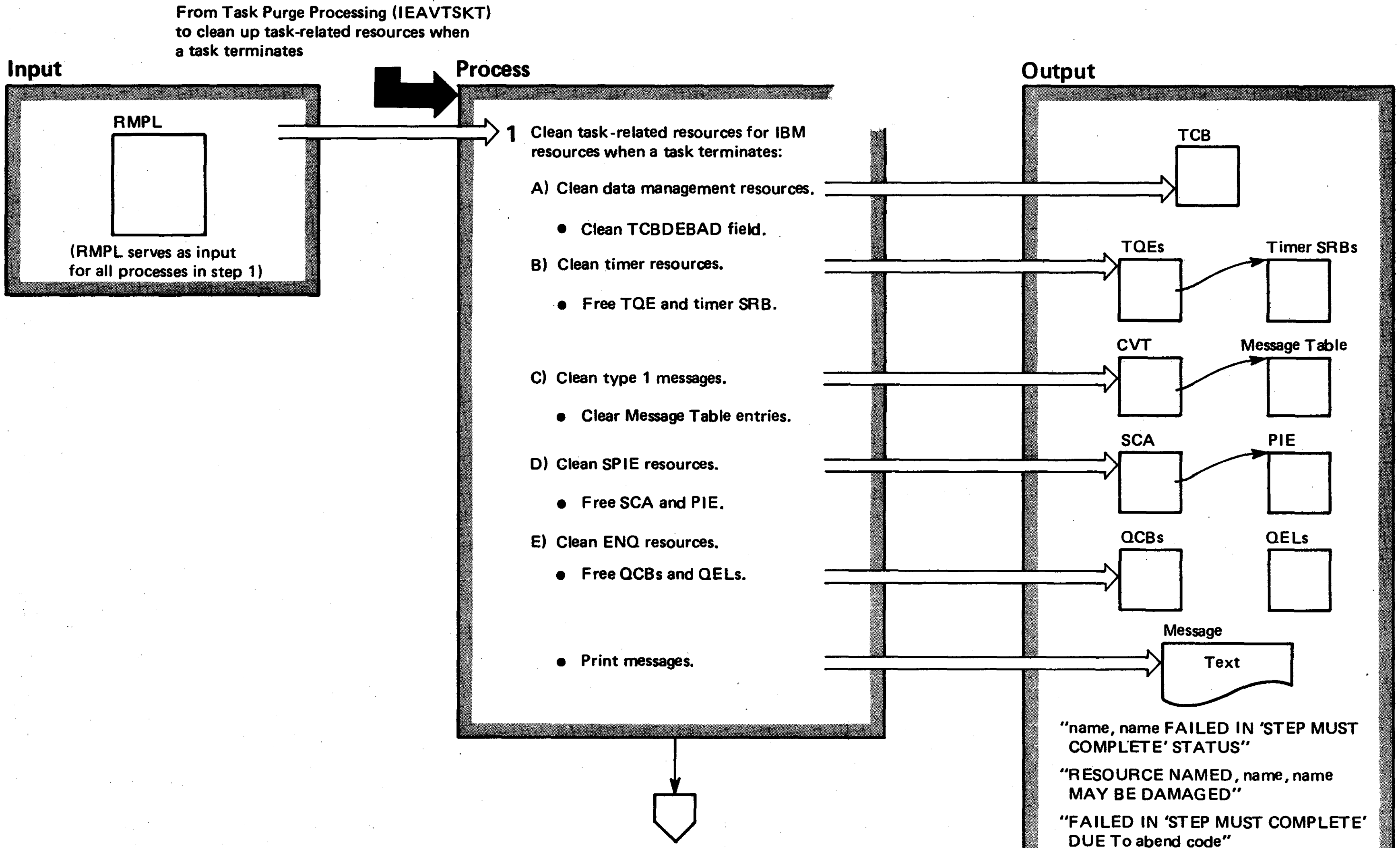
These routines free any control blocks related to the task. Control returns from these routines to the task purge processing function.

Extended Description	Module	Segment

6 Task purge processing prepares the RBs (request blocks) of the failing tasks to exit by placing the address of the EXIT routine in their RBOPSW field. When these RBs receive control, they will go to EXIT.

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

Diagram 22-20. Task Purge Resource Managers (IEAVTSKT) (Part 1 of 6)


From Task Purge Processing (IEAVTSKT) to clean up task-related resources when a task terminates
Input
RMPL
(RMPL serves as input for all processes in step 1)
Process
Output
TCB
TQEs
Timer SRBs
CVT
Message Table
SCA
PIE
QCBs
QELs
Message
Text

1 Clean task-related resources for IBM resources when a task terminates:

A) Clean data management resources.

- Clean TCBDEBAD field.

B) Clean timer resources.

- Free TQE and timer SRB.

C) Clean type 1 messages.

- Clear Message Table entries.

D) Clean SPIE resources.

- Free SCA and PIE.

E) Clean ENQ resources.

- Free QCBs and QELs.

- Print messages.

"name, name FAILED IN 'STEP MUST COMPLETE' STATUS"

"RESOURCE NAMED, name, name MAY BE DAMAGED"

"FAILED IN 'STEP MUST COMPLETE' DUE To abend code"

Diagram 22-20. Task Purge Resource Managers (IEAVTSKT) (Part 2 of 6)

Extended Description	Module	Segment
The IBM-defined task clean up resource managers free resources held during task processing. The task purge processing routine, module IEAVTSKT, routes control to these resource managers after establishing an interface via the RMPL (recovery management parameter list) in the RTM2WA. Control goes to each resource manager sequentially until all resource managers have performed their clean up processing.		
1 The task purge routine routes control to each of the IBM-defined resource managers. After one resource manager completes its processing, control comes back to the task purge routine, which routes control to the next resource manager. This continues until all the resource managers have performed clean up.	IEAVTSKT	TPURG1
A. The data management resource manager cleans the TCBDEBAD field of the TCB. (See the "Open/Close/EOV Logic" manual, SY26-3827, for more information about the data management resource manager.)	IFG0TC0A	
B. The timer resource manager frees the TQEs and timer SRBs associated with the task terminated. (See section 19, Timer Supervision, for a description of the timer purge routine.)	IEAVRTI1	
C. The type 1 message resource manager cleans the message table pointed to from the CVTQMSG field of the CVT.	IEAVTPMT	
D. The SPIE resource manager frees SPIE resources used by the terminating task by freeing the associated SCA (SPIE control area) and the PIE (program interruption element). (Section 21, Task Management (IEAVTB00), describes SPIE processing.)	IEAVSPIE	
E. The ENQ resource manager frees associated ENQ resources used by the terminating task by freeing QCBs (queue control block) and QELs (queue element). The ENQ resource manager also prints messages explaining which task failed while it controlled the resource. (See section 21, Task Management (IEAVENQ1), for a description of ENQ processing.)	IEAVENQ2	

Diagram 22-20. Task Purge Resource Managers (IEAVTSKT) (Part 3 of 6)

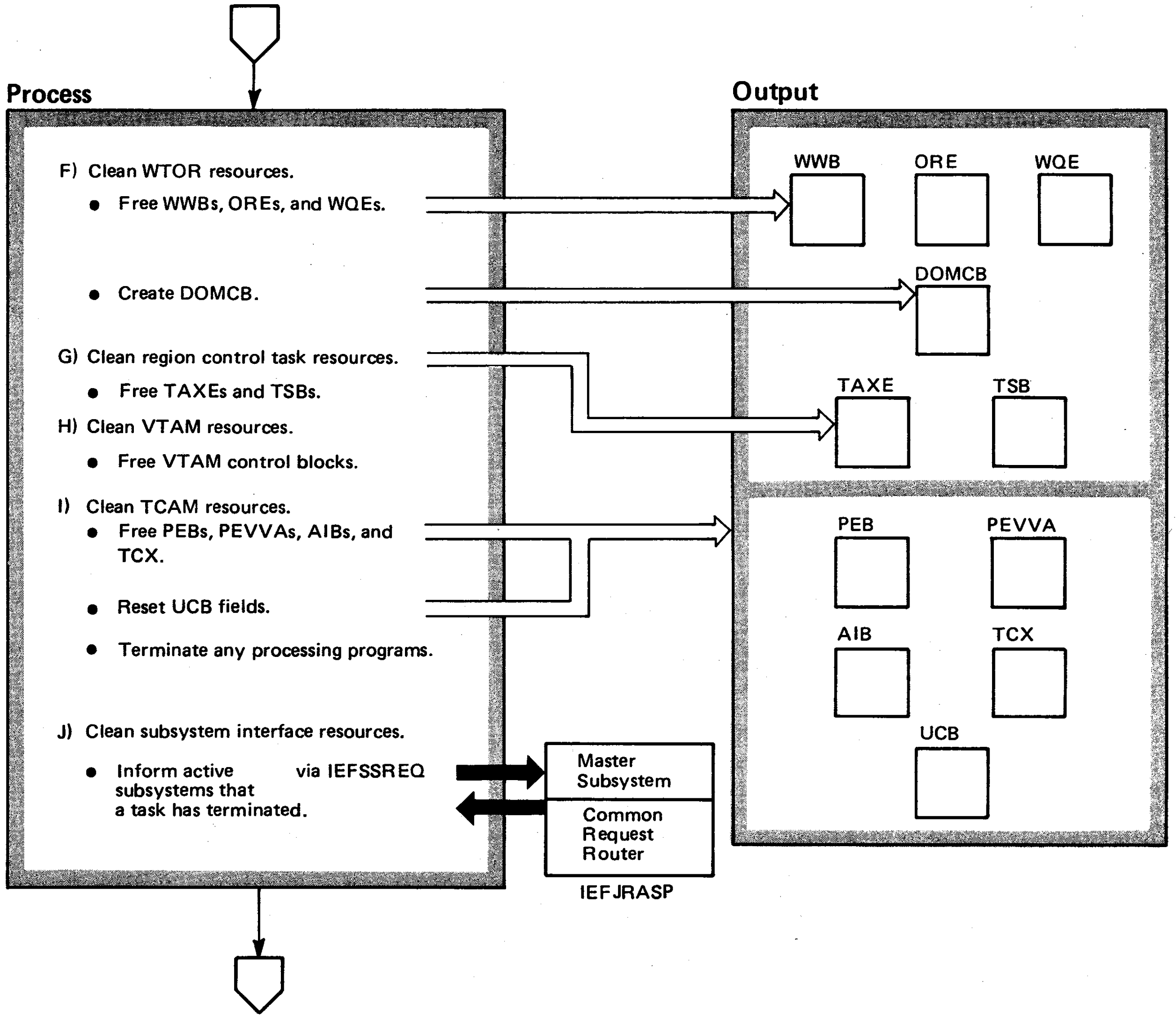
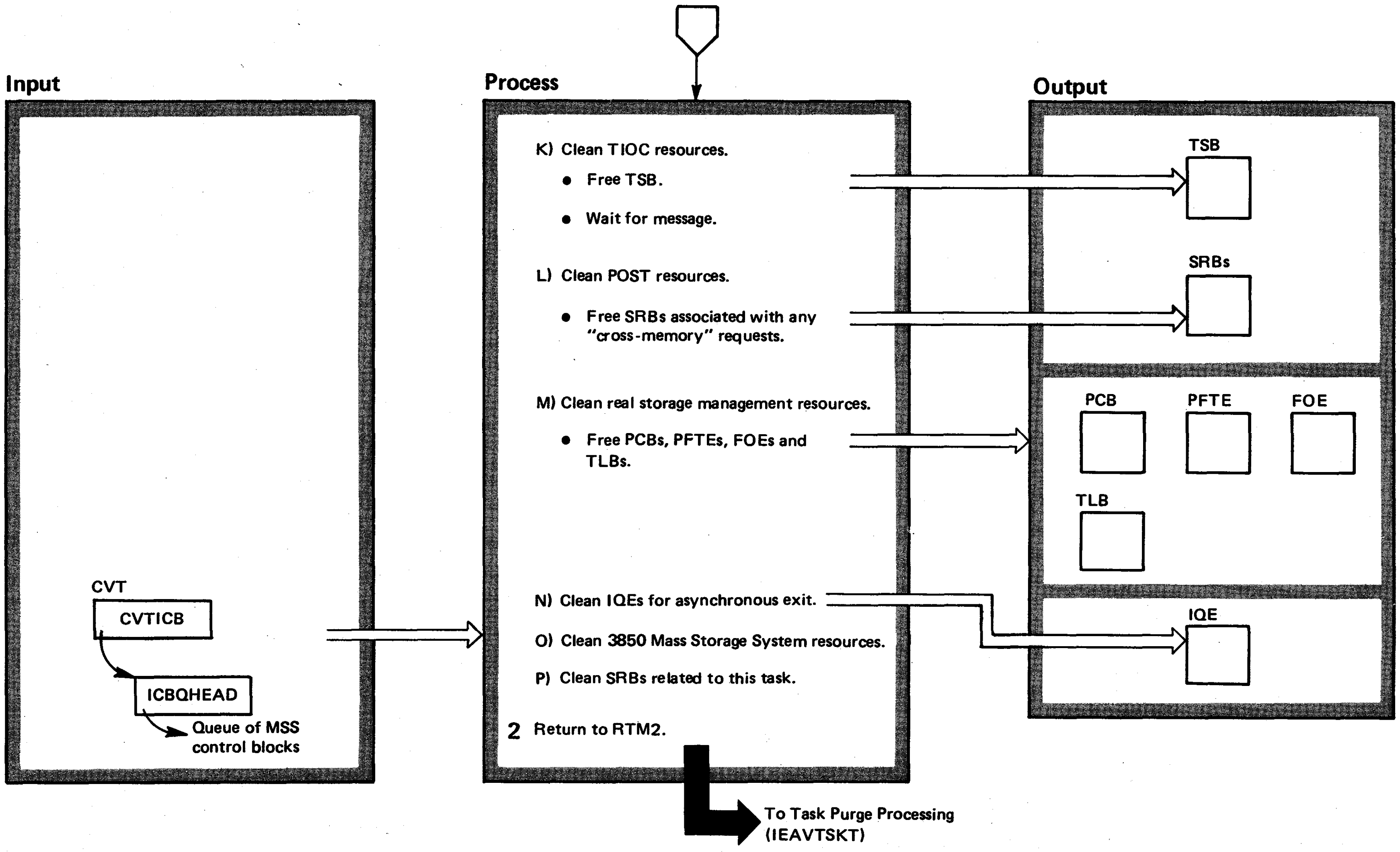

Process
F) Clean WTOR resources.
● Free WWBs, OREs, and WQEs.
● Create DOMCB.
G) Clean region control task resources.
● Free TAXEs and TSBs.
H) Clean VTAM resources.
● Free VTAM control blocks.
I) Clean TCAM resources.
● Free PEBs, PEVVAs, AIBs, and TCX.
● Reset UCB fields.
● Terminate any processing programs.
J) Clean subsystem interface resources.
● Inform active subsystems that a task has terminated. via IEFSSREQ
Master Subsystem
Common Request Router
IEFJRASP
Output
WWB
ORE
WQE
DOMCB
TAXE
TSB
PEB
PEVVA
AIB
TCX
UCB

Diagram 22-20. Task Purge Resource Managers (IEAVTSKT) (Part 4 of 6)

Extended Description	Module	Segment
F. The communications task resource manager cleans WTOR (write to operator with reply) resources associated with the task being terminated, by freeing the WWBs (write wait block), OREs (operator reply element), WQEs (write queue element), and DOMCs (delete operator message control blocks).	IEAVMED2	
G. The region control task resource manager cleans the resources associated with the task being terminated by freeing the TAXEs (terminal attention exit element) and TSBs (terminal status block). (See section 3, Region Control Task, for a complete description of the region control task resource manager.)	IEAVAR07	
H. The VTAM resource manager cleans up resources associated with the VTAM user task. These resources include storage, VTAM locks, and the following control blocks associated with the VTAM devices and applications active for the terminating task: ● Active CRAs (component recovery area) ● DEBs (data extent block) ● FMCBs (function management control block) ● NCBs (node control block) ● ICEs (inactive connection element) ● ACEs (active connection element) ● DCEs (DEB chain element) ● PST (process scheduling table) ● Application RDTEs (resource definition table) ● Destination RDTEs ● DVTs (destination vector table) ● EPTs (entry point table) (See the publication *OS/VS2 VTAM Logic,* SY28-0621, for a description of VTAM processing.)	ISTRAMA1	

Extended Description	Module	Segment
I. The TCAM (telecommunications access method) resource manager frees the resources associated with the terminating task. This resource manager frees the PEBs, PEWAs (process entry work area), AIBs, and TCXs associated with the failing task, and it resets UCB (unit control block) fields. (See the publication *OS/VS2 TCAM Logic,* SY30-2059, for a description of the TCAM resource manager.)	IEDQOT01	
J. The subsystem interface resource manager cleans the resources associated with the failing task by notifying the active subsystems, via the IEFSSREQ macro, of the task that just terminated.	IEFJRECM	

Diagram 22-20. Task Purge Resource Managers (IEAVTSKT) (Part 5 of 6)


Input
CVT
CVTICB
ICBQHEAD
Queue of MSS control blocks
Process
Output
TSB
SRBs
PCB
PFTE
FOE
TLB
IQE
To Task Purge Processing (IEAVTSKT)

K) Clean TIOC resources.

- Free TSB.
- Wait for message.

L) Clean POST resources.

- Free SRBs associated with any "cross-memory" requests.

M) Clean real storage management resources.

- Free PCBs, PFTEs, FOEs and TLBs.

N) Clean IQEs for asynchronous exit.

O) Clean 3850 Mass Storage System resources.

P) Clean SRBs related to this task.

2 Return to RTM2.

Diagram 22-20. Task Purge Resource Managers (IEAVTSKT) (Part 6 of 6)

Extended Description	Module	Segment
K. The TIOC (terminal input/output coordinator) resource manager cleans the TSB for the task being terminated.	IEDAY8	
L. The POST resource manager cleans the resources associated with the task being terminated by freeing the SRB associated with any cross-memory POST requests. (Section 21, Task Management, describes POST processing (IEAVSY50).)	IEAVSY50	
M. The real storage management resource manager cleans the resources associated with the task being terminated by freeing the PCBs (page control block), PFTE (page frame table entry), FOE (fix ownership entry), and TLB (translation lookaside buffer).	IEAVTERM	
N. The asynchronous exit resource manager cleans the resources for the task being terminated by freeing the IQE (interruption queue element).	IEAVEEEP	
O. The 3850 Mass Storage System resource manager marks invalid all delayed response queue elements relating to the terminating task.	ICB2AIR	
P. The task purge routine uses the PURGEDQ function to clean any SRBs related to the terminating task.	IEAVEPDO	
2 The task purge routine returns control to RTM2 after all the task resources have been freed.	IEAVTSKT	

Diagram 22-21. Address Space Purge Processing (IEAVTMMT) (Part 1 of 2)

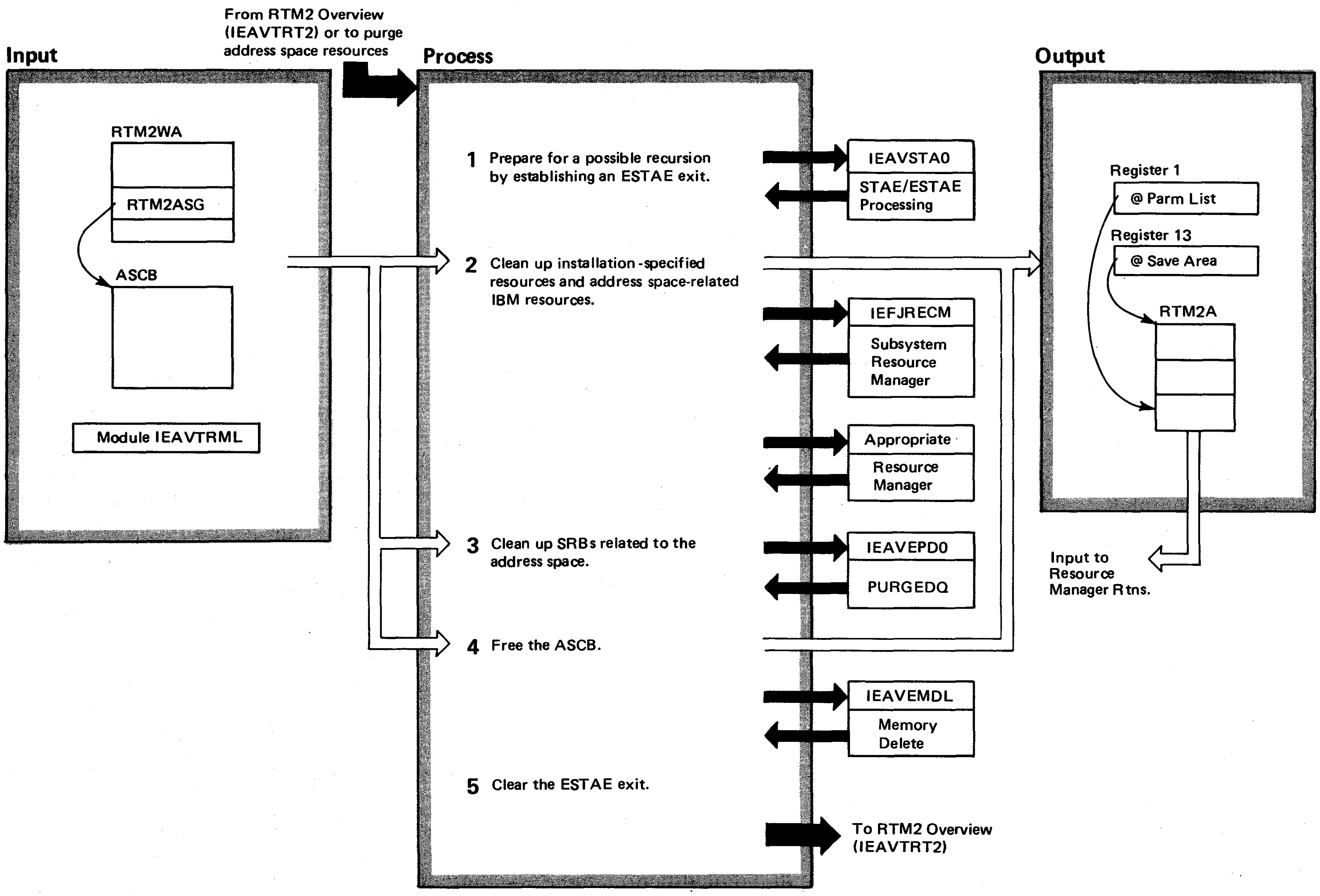
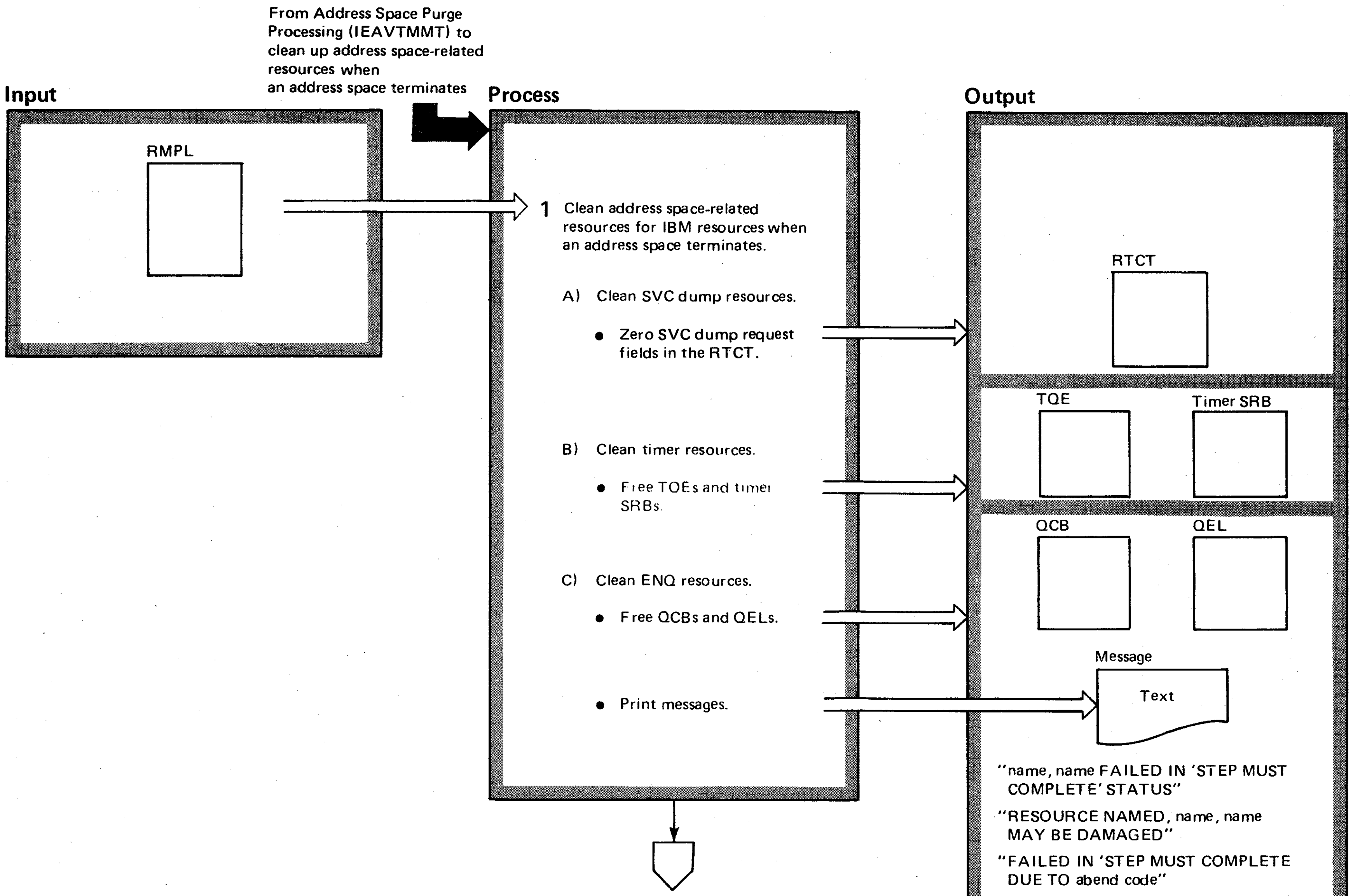

From RTM2 Overview (IEAVTRT2) or to purge address space resources
Input
RTM2WA
RTM2ASG
ASCB
Module IEAVTRML
Process
1 Prepare for a possible recursion by establishing an ESTAE exit.
2 Clean up installation-specified resources and address space-related IBM resources.
3 Clean up SRBs related to the address space.
4 Free the ASCB.
5 Clear the ESTAE exit.
IEAVSTA0
STAE/ESTAE Processing
IEFJRECM
Subsystem Resource Manager
Appropriate Resource Manager
IEAVEPD0
PURGEDQ
IEAVEMDL
Memory Delete
To RTM2 Overview (IEAVTRT2)
Output
Register 1
@ Parm List
Register 13
@ Save Area
RTM2A
Input to Resource Manager Rtns.

Diagram 22-21. Address Space Purge Processing (IEAVTMMT) (Part 2 of 2)

Extended Description	Module	Segment
The address space purge function cleans up the address space resources when it terminates. Control initially goes to the RTM1 mainline code (see M.O. diagram, RTM1 Overview (IEAVTRT2)) to service a CALLRTM= MEMTERM request. RTM1 then schedules the address space termination routines (see M.O. diagram, Address Space Termination Processing (IEAVTMMT)) to terminate the address space. The final process in address space termination occurs when RTM2 receives a request, from the address space termination routines, to purge the resources from the address space.		
Address space purge processing uses the RTM2WA initialized by initialization processing (see M.O. diagram RTM2 Initialization (IEAVTRT2)) for the basic input, along with the address of the ASCB being purged.		
The address space purge processing routine only honors requests from the master address space. Requesters from any other address space will be terminated.		
1 Address space purge processing establishes an ESTAE exit in case of failure.	IEAVTMMT	
2 Address space purge processing cleans up address space resources by first giving control to installation-defined subsystem clean-up routines (defined in module IEAVTRML) to clean any subsystem resources. These subsystem clean-up routines will receive control sequentially until they have all executed. Control next passes to the IBM-defined resource managers, which clean up system control program routines. The resource managers receive control sequentially:		

Extended Description	Module	Segment
1) SVC Dump	IEAVTSDR	
2) Timer	IEAVTRTI1	
3) ENQ/DEQ	IEAVENQ2	
4) Data Management	IFG0TC0A	
5) VTAM (virtual telecommunications access method)	ISTRAMA2	
6) TCAM (telecommunications access method)	IEDQOT01	
7) TIOC (terminal input/output coordinator)	IEDAY8	
8) WTOR (write-to-operator with reply)	IEAVMED2	
9) Schedule subsystem	IEFJRECM	
10) Initiator	IEFIRECM	
11) Scheduler allocation	IEFAB4E5	
12) POST	IEARPOST	
13) Virtual storage management	IEAVGFAS	
14) SETLOCK	IEAVELK0	
15) OLTEP (on-line test executive program)	OLT0A	
16) MSS	ICB2AIR	
17) RTM2	IEAVTMRM	
18) Type 1 message	IEAVTPMT	
19) ASCB Delete	IEAVEMDL	
SPIE and RCT M.O. diagram "Resource Managers" shows the modules that perform the clean-up, and the control blocks cleared.		
3 Control goes to the PURGEDQ routines (see M.O. diagram PURGEDQ Processing (IEAVEPD0) in the Supervisor Control section) to remove any SRBs left in the address space.		
4 Address space purge processing gives control to "memory delete" to free any non-permanent address spaces (ASID>1 in the ASCB). Address space purge processing does not free the address space if: ● ASID = 0 - system wait task ● ASID = 1 - master scheduler. Address space purge processing clears the ESTAE routine, and gives control to the caller (module IEAVTRTE).		

Diagram 22-22. Address Space Purge Resource Managers (IEAVTMMT) (Part 1 of 10)


From Address Space Purge Processing (IEAVTMMT) to clean up address space-related resources when an address space terminates
Input
RMPL
Process
Output
RTCT
TQE
Timer SRB
QCB
QEL
Message
Text

1 Clean address space-related resources for IBM resources when an address space terminates.

A) Clean SVC dump resources.

- Zero SVC dump request fields in the RTCT.

B) Clean timer resources.

- Free TQEs and timer SRBs.

C) Clean ENQ resources.

- Free QCBs and QELs.

- Print messages.

"name, name FAILED IN 'STEP MUST COMPLETE' STATUS"

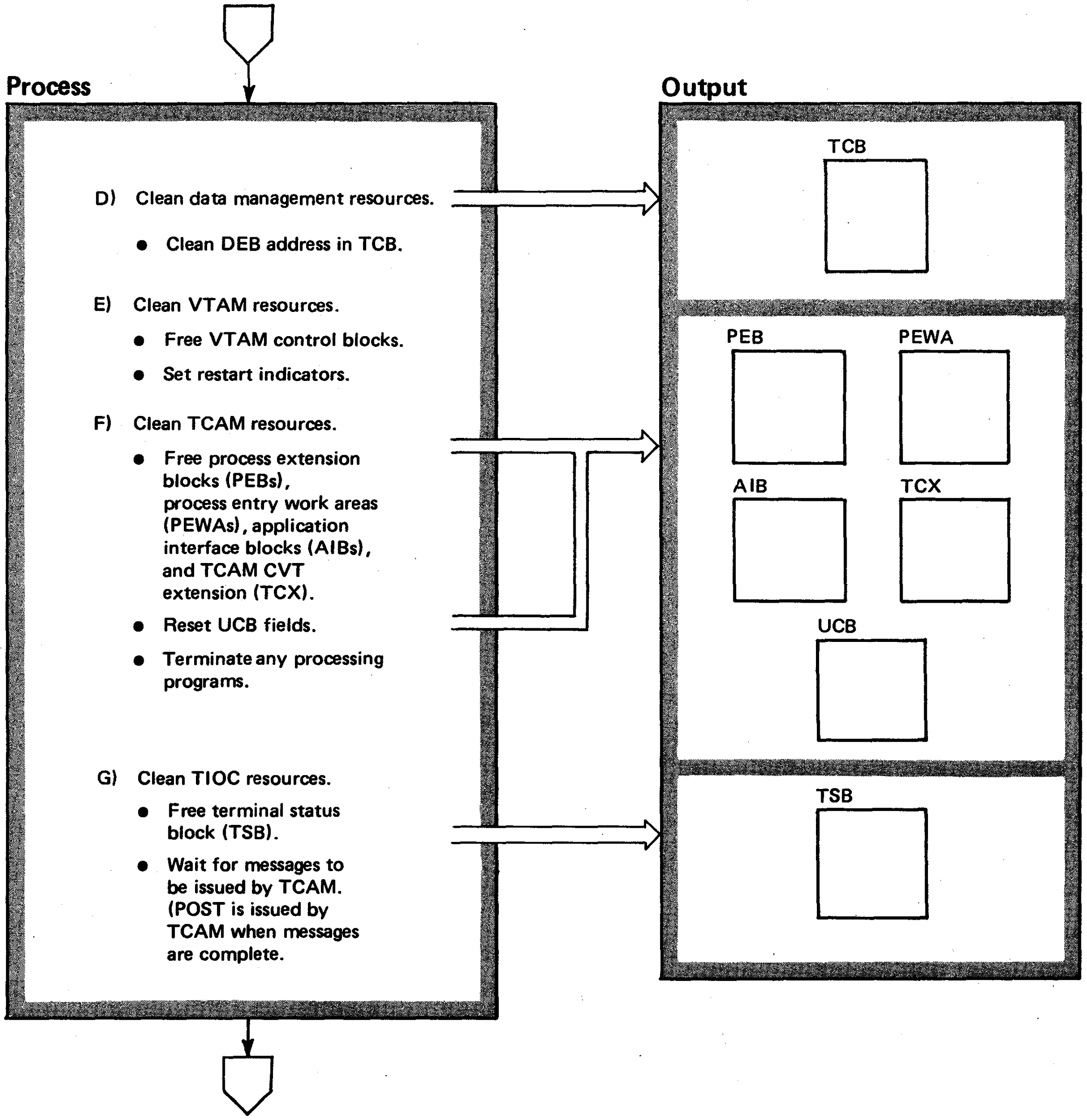
"RESOURCE NAMED, name, name MAY BE DAMAGED"

"FAILED IN 'STEP MUST COMPLETE DUE TO abend code"

Diagram 22-22. Address Space Purge Resource Managers (IEAVTMMT) (Part 2 of 10)

Extended Description	Module	Segment
The IBM-defined address space clean up resource managers free any resources held by an address space during processing. The address space purge processing routine, module IEAVTMMT, routes control to these resource managers after establishing an interface. Control goes to each address space resource manager sequentially until all of them have performed their clean up processing.		
1 The address space purge routine routes control to each of the IBM-defined resource managers. After one resource manager completes its processing, control returns to the address space purge routine, which routes control to the next resource manager. This continues until all the resource managers have performed clean up.	IEAVTMMT	
A. The SVC dump resource manager issues STATUS to set the system dispatchable if a dump was in progress in the failing address space. The address space purge routine sets supervisor trace active.	IEAVTSDR	
B. The timer resource manager frees the TQEs (timer queue elements) and timer SRBs associated with the address space being terminated. (See section 19, Timer Supervision (IEAVRTI1), for a description of the timer purge routine.)	IEAVTRTI1	
C. The ENQ resource manager frees associated ENQ resources used by the terminating address space by freeing QCBs (queue control blocks) and QELs (queue elements). The ENQ resource manager also writes messages explaining which address space failed while it controlled the resource. (See section 21, Task Management (IEAVENQ1) section for a detailed description of ENQ processing.)	IEAVENQ1	

Diagram 22-22. Address Space Purge Resource Managers (IEAVTMMT) (Part 3 of 10)



Process

D) Clean data management resources.

- Clean DEB address in TCB.

E) Clean VTAM resources.

- Free VTAM control blocks.
- Set restart indicators.

F) Clean TCAM resources.

- Free process extension blocks (PEBs), process entry work areas (PEWAs), application interface blocks (AIBs), and TCAM CVT extension (TCX).
- Reset UCB fields.
- Terminate any processing programs.

G) Clean TIOC resources.

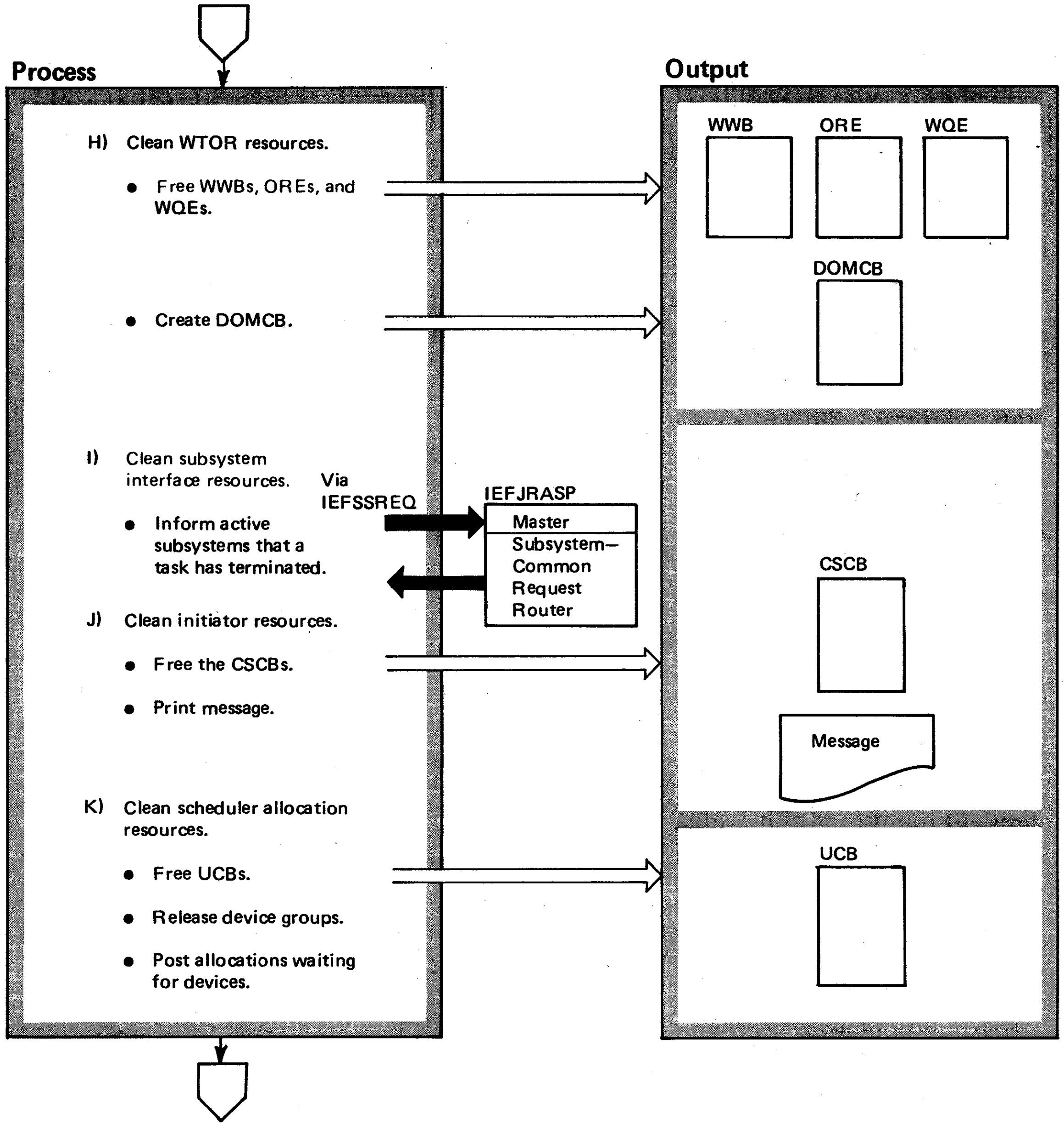
- Free terminal status block (TSB).
- Wait for messages to be issued by TCAM. (POST is issued by TCAM when messages are complete.

Diagram 22-22. Address Space Purge Resource Managers (IEAVTMMT) (Part 4 of 10)

Extended Description	Module	Segment
D. The data management resource manager cleans the TCBDEBAD field of the TCB. This field contains the DEB address from the DCB. (See the publication *OS/VS2 Open/Close/EOV Logic,* SY26-3827, for more detailed information about the data management resource manager.)	IFG0TC0A	
E. The VTAM resource manager cleans up resources associated with the VTAM user address space. These resources include storage, VTAM locks, and control blocks associated with the VTAM devices and applications which were active for this address space. The user's address space control blocks consist of: ● Active CRAs (component recovery area) ● DEBs (data extent block) ● FMCBs (function management control block) ● NCBs (node control block) ● ICEs (inactive connection element) ● ACEs (active connection element) ● DCEs (DEB chain element) ● PST (process scheduling table) ● Application RDTEs (resource definition table) ● Destination RDTEs ● DVTs (destination vector table) ● EPTs (entry point table) ● MPSTs (memory process scheduling table) VTAM's address space control blocks consist of: ● AVT (VTAM address vector table) ● ATCVT (VTAM communications vector table) ● ISTCONFT (configuration table) ● CVT If the terminated address space is VTAM's, appropriate indicators in the CVT are reset to zero to allow VTAM to be restarted. (These indicators are the CVTATCVT, the CVTRMPTT, and the CVTRMPMT.) (See the publication *OS/VS2 VTAM Logic,* SY28-0621 for a description of VTAM processing.)	ISTRAMA2	

Extended Description	Module	Segment
F. The TCAM (telecommunications access method) resource manager frees the resources associated with the terminating address space by freeing the PEBs, PEWAs, AIBs, and TCXs, and it resets UCB (unit control block) fields. (See the publication *OS/VS2 TCAM Logic,* SY30-2059, for a description of the TCAM resource manager.)	IEDQOT01	
G. The TIOC (terminal input/output coordinator) resource manager cleans the TSB (terminal status block) for the address space being terminated. (See the publication *OS/VS TCAM Logic,* SY30-2059, for more detailed information about the TCAM termination messages.)	IEDAY8	

Diagram 22-22. Address Space Purge Resource Managers (IEAVTMMT) (Part 5 of 10)



H) Clean WTOR resources.

- Free WWBs, OREs, and WQEs.
- Create DOMCB.

I) Clean subsystem interface resources.

- Inform active subsystems that a task has terminated.

J) Clean initiator resources.

- Free the CSCBs.
- Print message.

K) Clean scheduler allocation resources.

- Free UCBs.
- Release device groups.
- Post allocations waiting for devices.

Diagram 22-22. Address Space Purge Resource Managers (IEAVTMMT) (Part 6 of 10)

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

Diagram 22-22. Address Space Purge Resource Managers (IEAVTMMT) (Part 7 of 10)

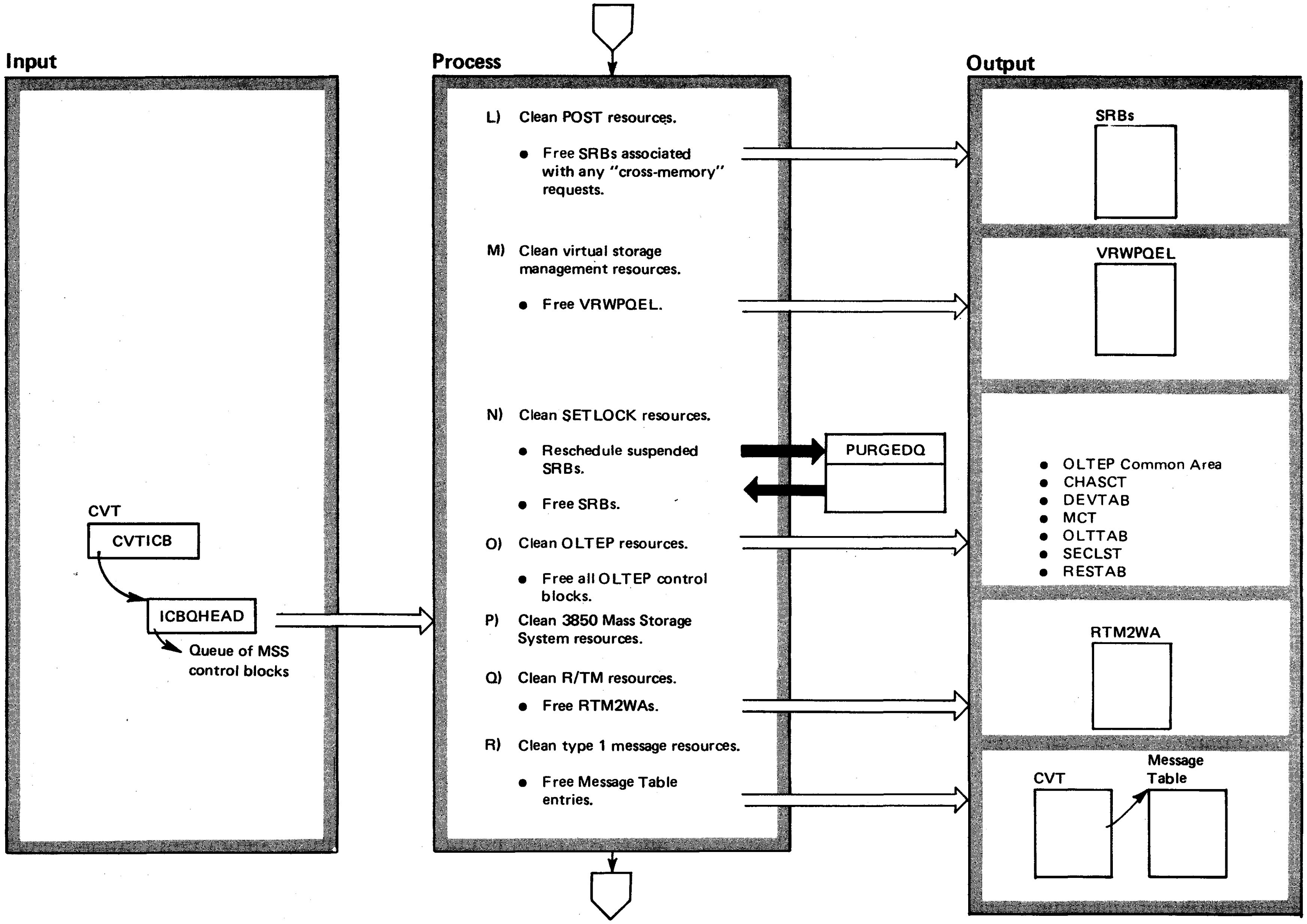
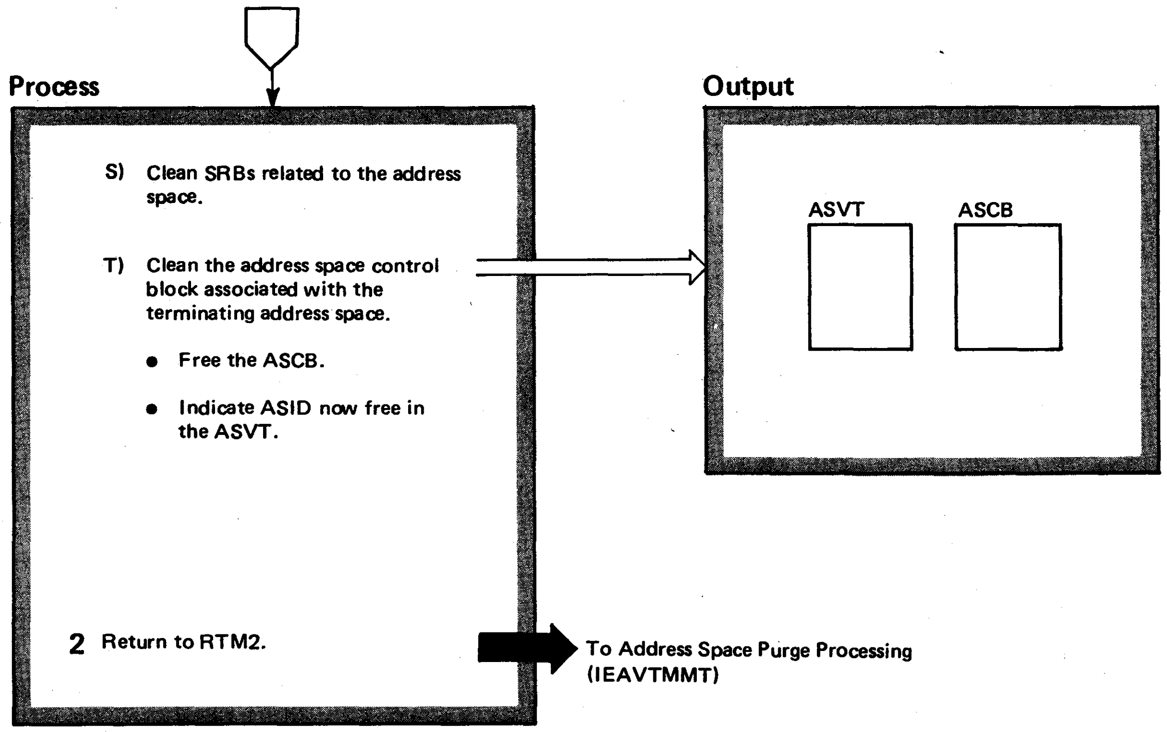

Input
CVT
CVTICB
ICBQHEAD
Queue of MSS control blocks
Process
L) Clean POST resources.
● Free SRBs associated with any "cross-memory" requests.
M) Clean virtual storage management resources.
● Free VRWPQEL.
N) Clean SETLOCK resources.
● Reschedule suspended SRBs.
● Free SRBs.
PURGEDQ
O) Clean OLTEP resources.
● Free all OLTEP control blocks.
P) Clean 3850 Mass Storage System resources.
Q) Clean R/TM resources.
● Free RTM2WAs.
R) Clean type 1 message resources.
● Free Message Table entries.
Output
SRBs
VRWPQEL
● OLTEP Common Area
● CHASCT
● DEVTAB
● MCT
● OLTTAB
● SECLST
● RESTAB
RTM2WA
CVT
Message Table

Diagram 22-22. Address Space Purge Resource Managers (IEAVTMMT) (Part 8 of 10)

Extended Description	Module	Segment
L. The POST resource manager cleans the resources associated with the address space being terminated by freeing the SRB associated with any cross-memory POST requests. (The "Task Management" section describes POST processing.)	IEAVSY50	
M. The virtual storage management resource manager cleans resources associated with the address space by freeing the VRWPQEL (virtual equals real wait or post queue element). (See the "Virtual Storage Management" section for a complete description of the resource manager.)	IEAVGCAS	
N. The SETLOCK resource manager cleans up resources associated with the address space being terminated by scheduling suspended SRBs. These SRBs will be freed after they complete their processing. (The "Supervisor Control" section describes SETLOCK processing.)	IEAVELK	
O. The OLTEP resource manager cleans the resources associated with the address space being terminated by freeing the OLTEP control blocks: • OLTEP common area (module IFDOLT23) • CHASCT (OLT program control table) • DEVTAB (device tables) • MCT (module control table) • OLTTAB (OLT program link table) • SECLST (test section list) • RESTAB (CDS equate resident table) (See the publication "OS/VS2 OLTEP Logic," SY28-0675, for a complete description of the OLTEP resource manager.)	IFDOLT0A	

Extended Description	Module	Segment
P. The 3850 Mass Storage System resource manager marks invalid all delayed response queue elements relating to the terminating address space.	ICB2AIR	
Q. The R/TM resource manager frees all RTM2WAs (recovery termination management 2 work area) obtained from SQA (system queue area) for tasks in the terminating address space.	IEAVTMRM	
R. The type 1 message resource manager cleans the resources by freeing any entries in the type 1 message table associated with the address space being terminated.	IEAVTPMT	

Diagram 22-22. Address Space Purge Resource Managers (IEAVTMMT) (Part 9 of 10)



S) Clean SRBs related to the address space.

T) Clean the address space control block associated with the terminating address space.

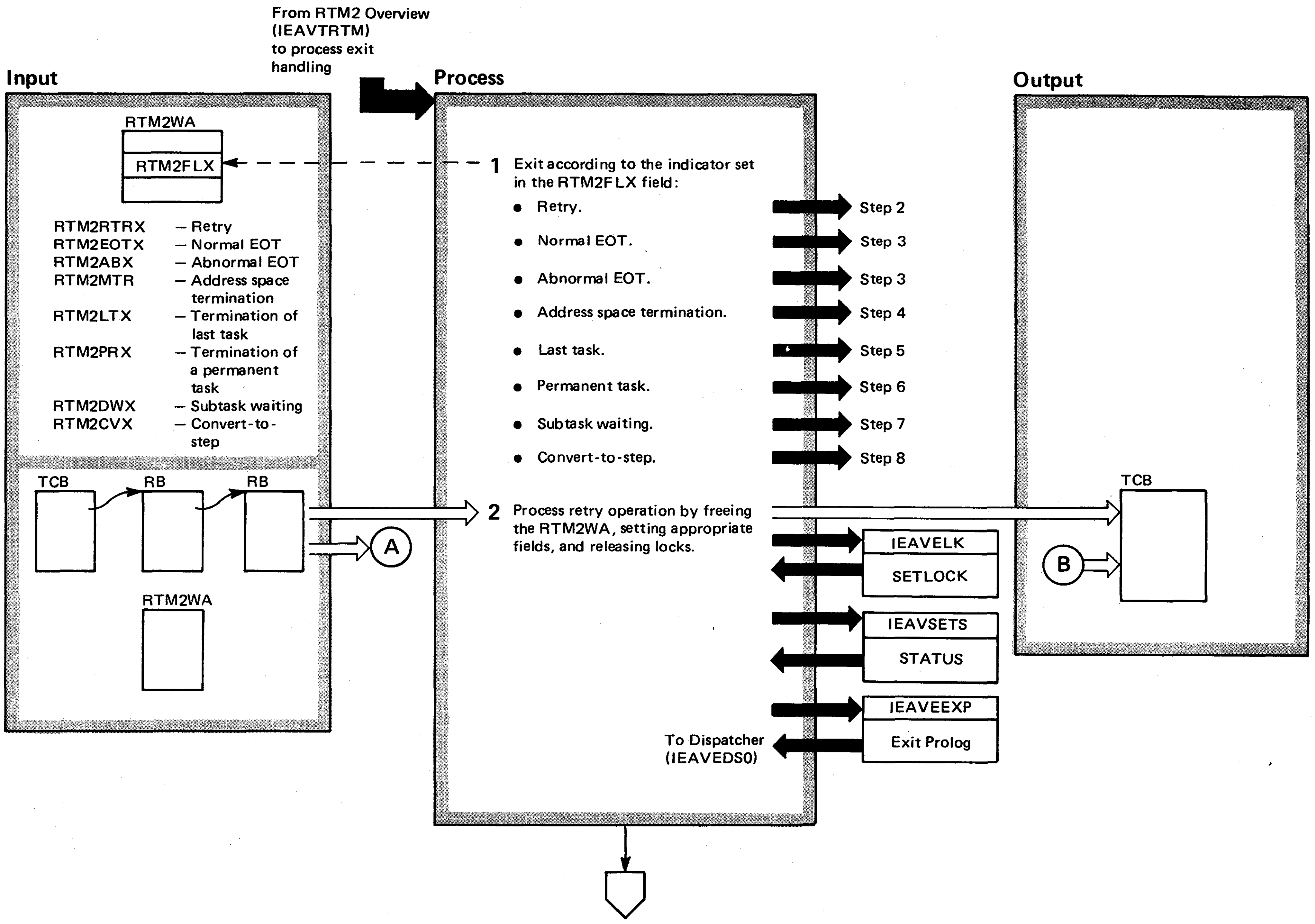
- Free the ASCB.
- Indicate ASID now free in the ASVT.

2 Return to RTM2.

Diagram 22-22. Address Space Purge Resource Managers (IEAVTMMT) (Part 10 of 10)

Extended Description	Module	Segment
S. The address space purge routine uses the PURGEDQ function to free SRBs associated with the terminating address space. (The "Supervisor Control" section fully describes PURGEDQ processing.)	IEAVTMMT	
T. The virtual address space terminating routine acts as a resource manager to clean up the resource held by the terminating address space by freeing the ASCB and indicating in the ASVT the ASID of the address space associated with the terminating address space.	IEAVGCAS	
2 The address space purge routine returns control to RTM2 after all the resources have been freed.	IEAVTMMT	

Diagram 22-23. RTM2 Exit Processing (IEAVTRTE) (Part 1 of 6)



1 Exit according to the indicator set in the RTM2FLX field:

- Retry.
- Normal EOT.
- Abnormal EOT.
- Address space termination.
- Last task.
- Permanent task.
- Subtask waiting.
- Convert-to-step.

2 Process retry operation by freeing the RTM2WA, setting appropriate fields, and releasing locks.

Diagram 22-23. RTM2 Exit Processing (IEAVTRTE) (Part 2 of 6)

Extended Description	Module	Segment
RTM2 exits to either exit prolog or STATUS (see the M.O. diagrams for Exit Prolog and STATUS for a description of their processing), depending on the settings of the RTM2FLX field of the RTM2WA, after task termination or address space termination.		
1 Exit processing determines the type of exit.	IEAVTRTE	
2 The current RTM2WA is freed; the TCB flags are cleared if no RTM2 SVRBs will remain on the RB queue after retry; and the registers that will not be altered by Exit (15, 0, 1) are reloaded from the SVRB. Then control is passed to the Exit prolog.		RTECMEX RTEFREWA

Diagram 22-23. RTM2 Exit Processing (IEAVTRTE) (Part 3 of 6)

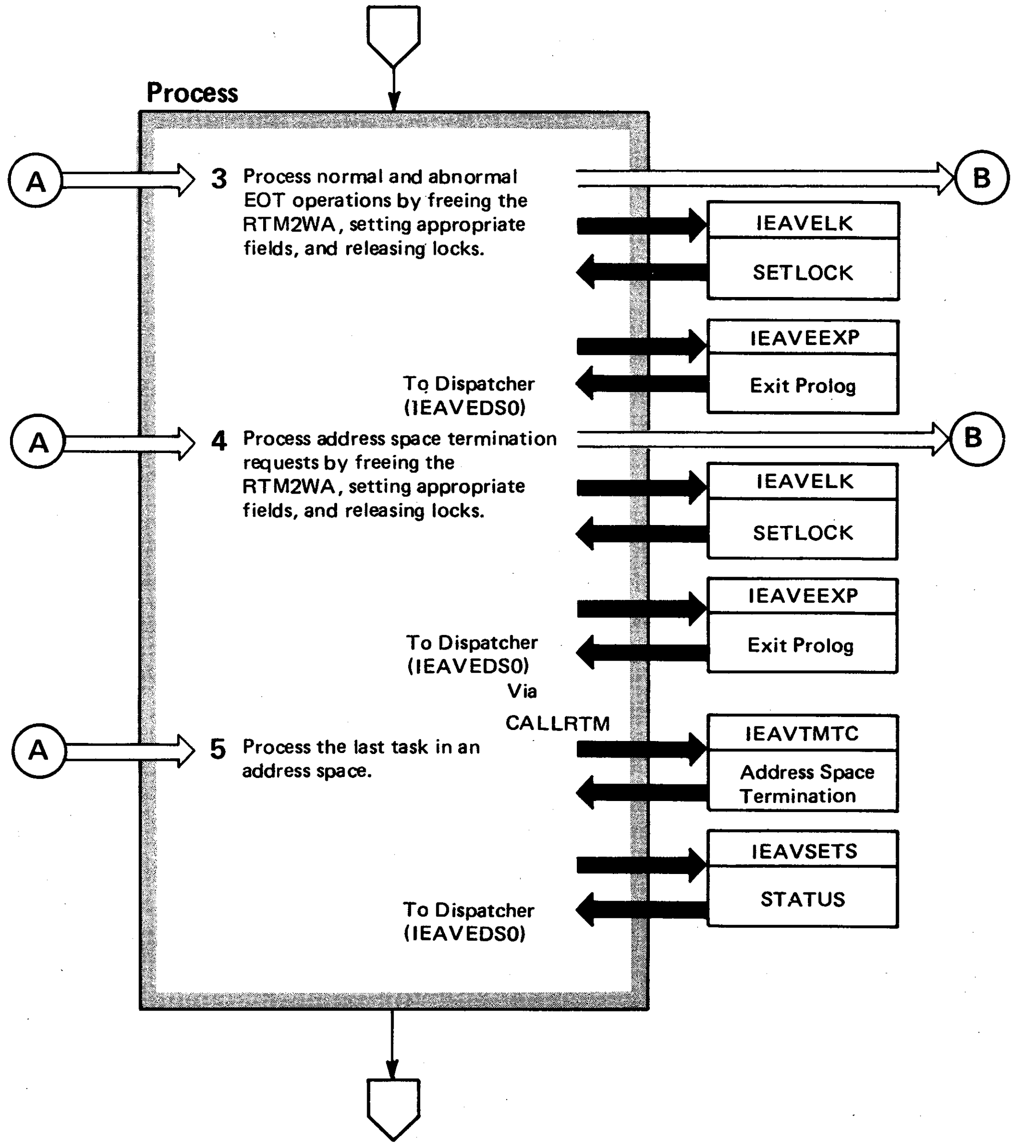

Process
A
3 Process normal and abnormal EOT operations by freeing the RTM2WA, setting appropriate fields, and releasing locks.
B
IEAVELK
SETLOCK
IEAVEEXP
Exit Prolog
To Dispatcher (IEAVEDSO)
A
4 Process address space termination requests by freeing the RTM2WA, setting appropriate fields, and releasing locks.
B
IEAVELK
SETLOCK
IEAVEEXP
Exit Prolog
To Dispatcher (IEAVEDSO)
Via CALLRTM
A
5 Process the last task in an address space.
IEAVTMTC
Address Space Termination
IEAVSETS
STATUS
To Dispatcher (IEAVEDSO)

Diagram 22-23. RTM2 Exit Processing (IEAVTRTE) (Part 4 of 6)

Extended Description	Module	Segment
3 The TCBEOT flag is set to indicate all RTM2 processing is complete for this task. All RTM2 work areas are freed, and control is passed to the Exit prolog.		
4 The RTM2WA is freed and control is passed to the Exit prolog.		
5 The memory is terminated using CALLRTM TYPE=MEMTERM. The current task is set non-dispatchable to await completion of memory termination.		RTELTEX

Diagram 22-23. RTM2 Exit Processing (IEAVTRTE) (Part 5 of 6)

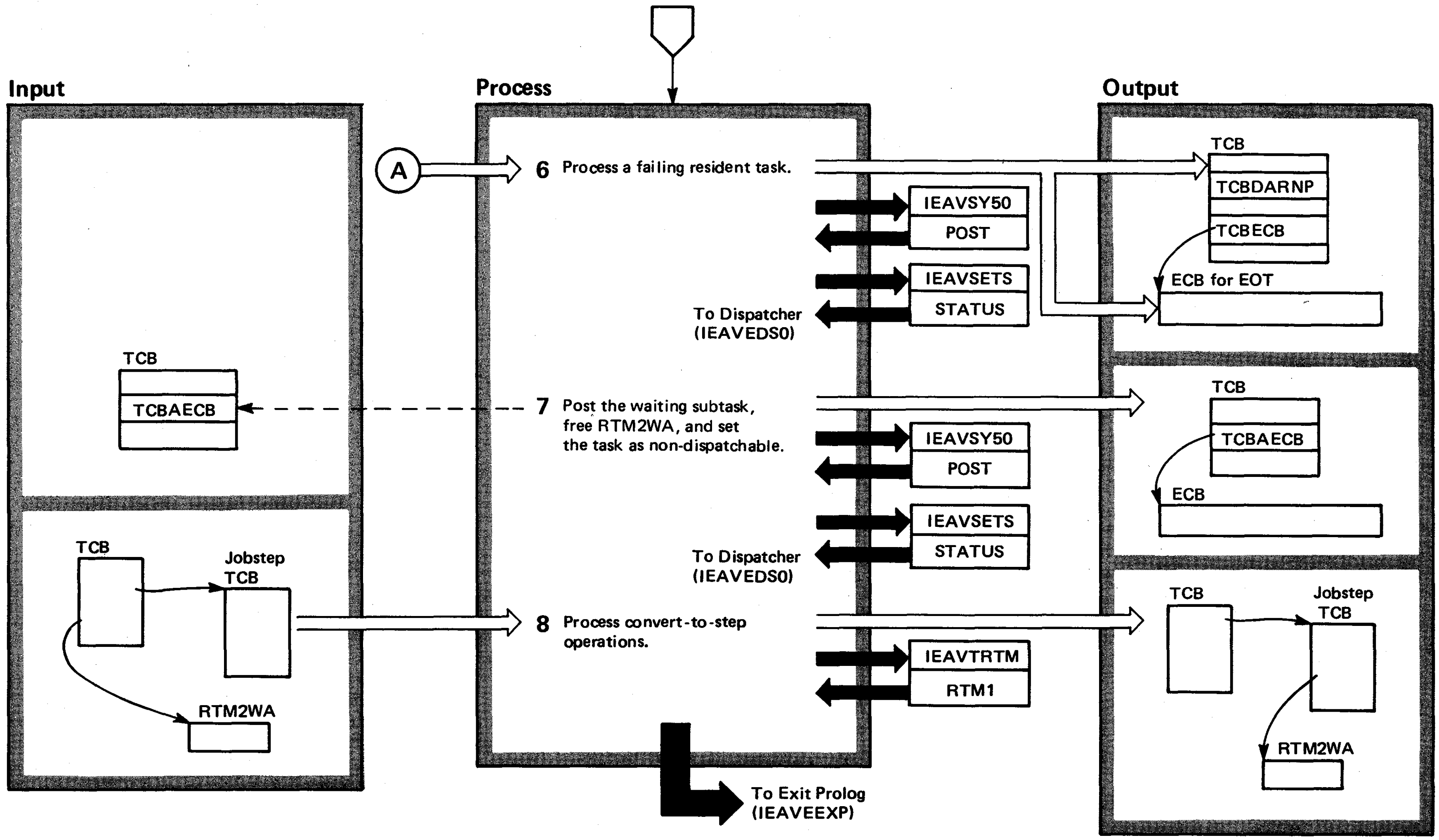


Diagram 22-23. RTM2 Exit Processing (IEAVTRTE) (Part 6 of 6)

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

Diagram 22-24. Address Space Termination Processing (IEAVTMTC) (Part 1 of 4)

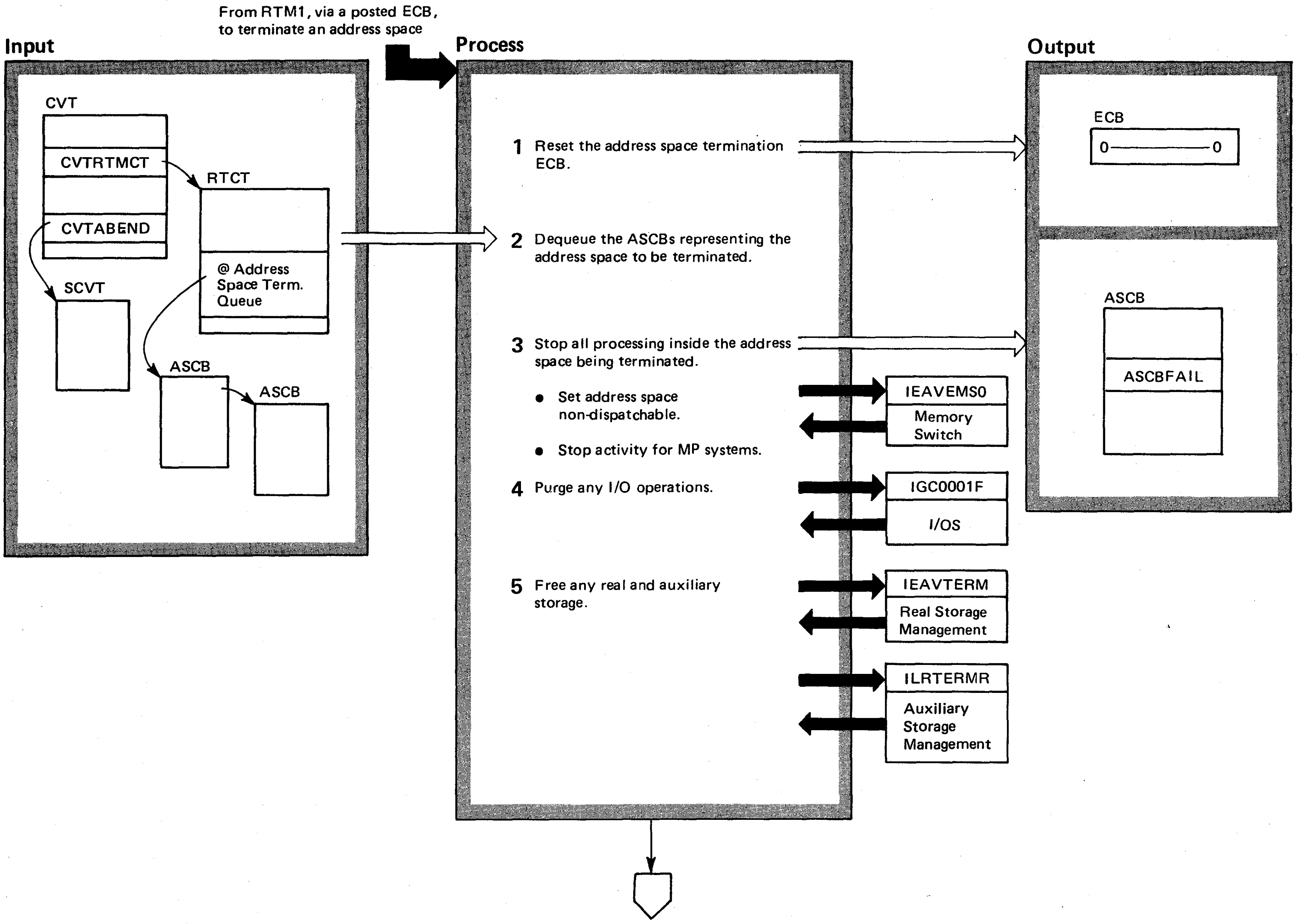

From RTM1, via a posted ECB, to terminate an address space
Input
CVT
CVTRTMCT
CVTABEND
RTCT
@ Address Space Term. Queue
SCVT
ASCB
ASCB
Process
1 Reset the address space termination ECB.
2 Dequeue the ASCBs representing the address space to be terminated.
3 Stop all processing inside the address space being terminated.
• Set address space non-dispatchable.
• Stop activity for MP systems.
4 Purge any I/O operations.
5 Free any real and auxiliary storage.
IEAVEMS0
Memory Switch
IGC0001F
I/OS
IEAVTERM
Real Storage Management
ILRTERMR
Auxiliary Storage Management
Output
ECB
0 — 0
ASCB
ASCBFAIL

Diagram 22-24. Address Space Termination Processing (IEAVTMTC) (Part 2 of 4)

Extended Description	Module	Segment
Address space termination routines receive control from RTM1 when a system routine issues a CALLRTM TYPE = MEMTERM request. Address space termination consists of two routines, (IEAVTMTC and IEAVTMTR) both resident in the master address space, that: ● Find and dequeue the ASCB (address space control block) representing the address space to be terminated. ● Stop the processing in the address space. ● Perform the actual termination. ● Repeat the operation for all the ASCBs on the termination queue. After this processing has completed for all the address spaces on the termination queue, module IEAVTMTC goes into a wait state, to wait for another address space termination request.		
1 Since this routine receives control after an SRB scheduled by RTM1 posts its ECB, the ECB must be zeroed to allow for later entries.	IEAVTMTC	
2 Address space termination proceeds to dequeue the last ASCB on the termination queue by following down the chain pointed to out of the RTCT (recovery termination control table). The CS (compare and swap) instruction is used to remove the ASCB.		
3 Address space termination sets the address space nondispatchable. If the system is an MP (multiprocessor) system with more than one CPU online, address space termination must stop all activity in the address space being terminated. It does this by giving control to the memory switch function (see the "Supervisor Control" section for a description of the memory switch function) and by waiting until any SRB or task activity stops.		
4 All I/O activity for the address space is stopped. Control goes to the I/O supervisor, via SVC 16, to perform this function.		
5 All real page frames and all auxiliary storage pages belonging to the address space are released. Control goes to the RSM (real storage management) and ASM (auxiliary storage management) routines to perform this function. The SCVTPTRM field of the SCVT contains the entry point address of the real storage management routine. The CVTASMRM field of the CVT contains the entry point of the auxiliary storage management routine.		

Diagram 22-24. Address Space Termination Processing (IEAVTMTC). (Part 3 of 4)

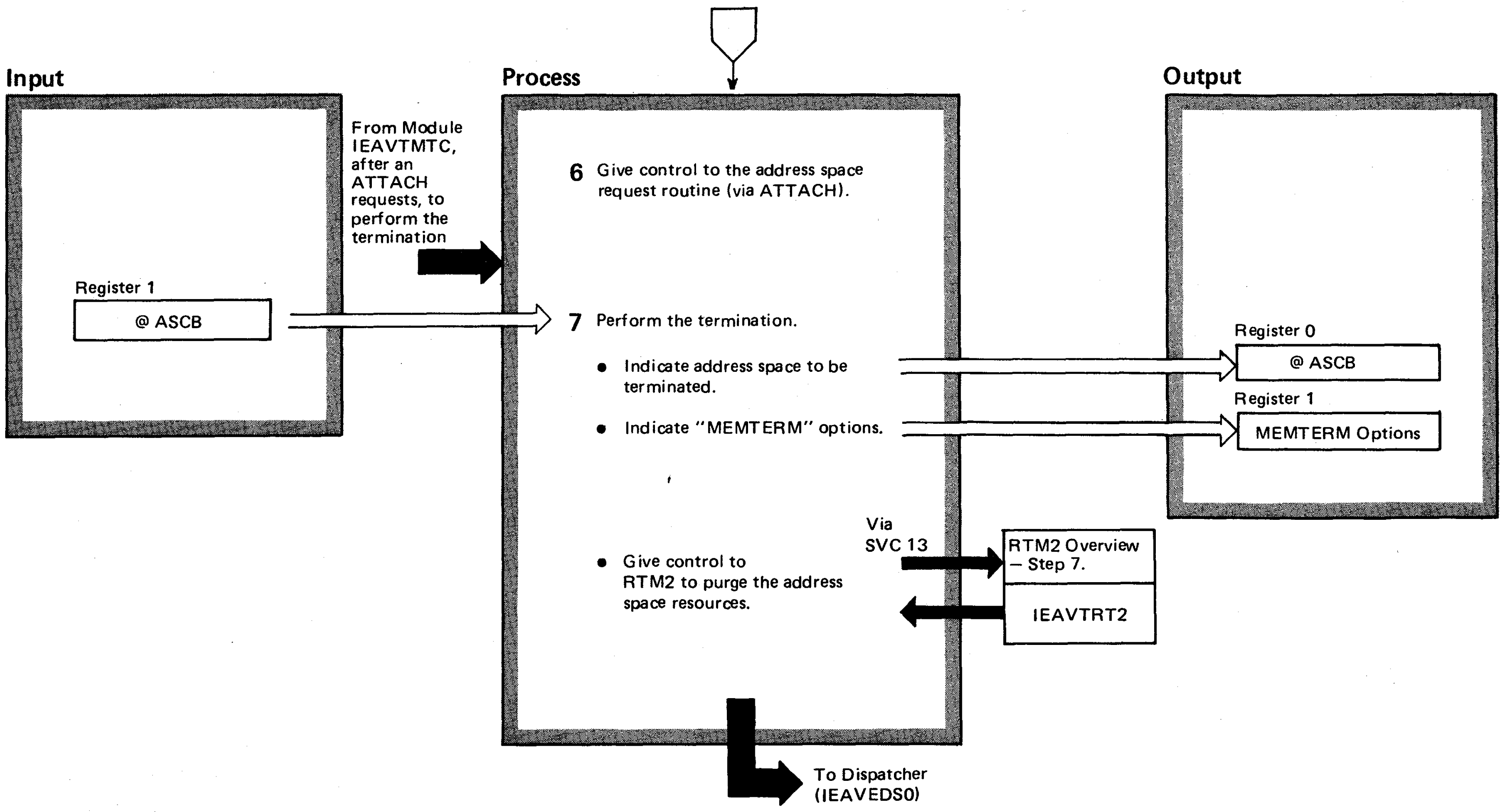

Input
Process
Output
From Module IEAVTMTC, after an ATTACH requests, to perform the termination
Register 1
@ ASCB
6 Give control to the address space request routine (via ATTACH).
7 Perform the termination.
• Indicate address space to be terminated.
• Indicate "MEMTERM" options.
• Give control to RTM2 to purge the address space resources.
Register 0
@ ASCB
Register 1
MEMTERM Options
Via SVC 13
RTM2 Overview – Step 7.
IEAVTRT2
To Dispatcher (IEAVEDSO)

Diagram 22-24. Address Space Termination Processing (IEAVTMTC) (Part 4 of 4)

Extended Description	Module	Segment
6 Address space termination continues after module IEAVTMTC, the controller routine, attaches the address space termination task, IEAVTMTR, to perform the actual termination. (IEAVTMTR runs in the master address space.)		
7 The address space termination task indicates the address space being terminated in register 0, and the "MEMTERM" options in register 1, and gives control to RTM2, via SVC 13, to purge the address space resources. (See M.O. diagram RTM2 Overview (IEAVTRT2) and M.O. diagram Address Space Purge Processing (IEAVTMMT) for the description of how R/TM purges address space resources.) After control comes back from RTM2, the address space termination task gives control to the dispatcher.	IEAVTMTR	

Diagram 22-25. STAE/ESTAE Processing (IEAVSTA0) (Part 1 of 6)

Branch Entry From Type 2, 3, or 4 SVCs, or From SVC IH to process a E/STAE macro instruction

Input

Branch Entry Only

Register 13

@ Reg Save Area

Register 15

Entry Point @

Branch and SVC

Register 1

@ Parm List

Register 0

Code

00, 10	– Create SCB
02	– Propagate SCB
04, 84, 94	– Cancel SCB
08, 18	– Overlay SCB

Register 14

Return @

SVC Entry Only

Register 3

@ CVT

Register 4

@ TCB

Register 5

@ SVC 60 SVRB

Register 6

Entry Point @

Register 7

@ ASCB

Process

1 Validate the request.

- Invalid.

ABEND

2 Obtain work area

- For SVC entry, use area in SVRB.
- For branch entry, use cell from free SCB queue.

3 Perform requested service

- Create SCB (use cell from FREESCBQ)
- Cancel SCB (add cell to FREESCBQ)
- Overlay SCB
- Propogate SCB (use cells from FREESCBQ)

Output

Completion Code

X'13C'

Register 15

Reason Code

For ESTAR/ESTAI when task terminated:

04	– Invalid ESTAR Request
08	– Invalid ESTAI Request
0C	– Invalid branch entry to SVC 60 service routine

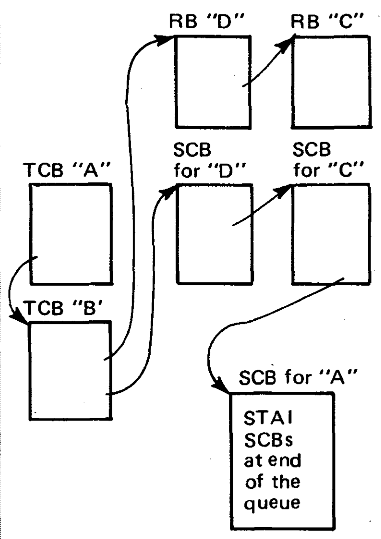
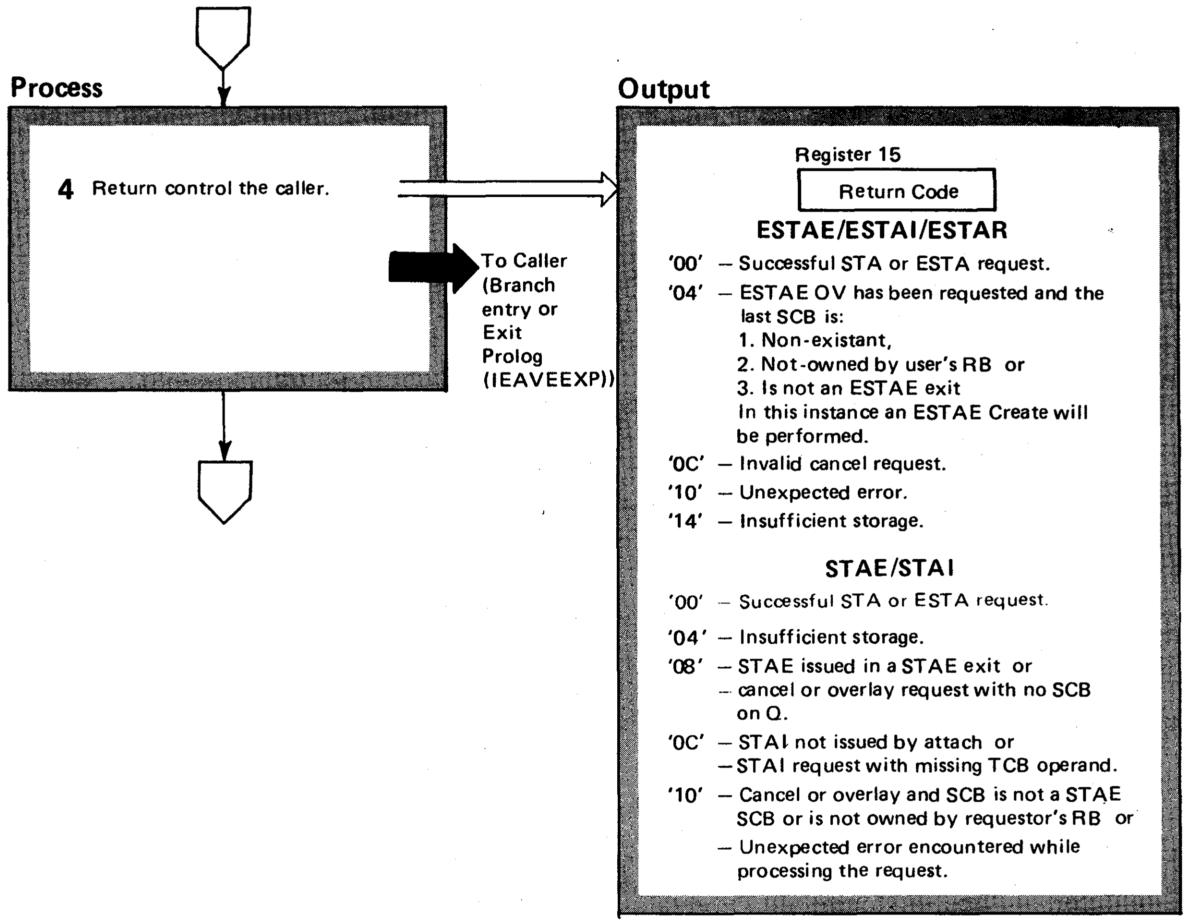


Diagram 22-25. STAE/ESTAE Processing (IEAVSTA0) (Part 2 of 6)

Extended Description	Module	Segment
The STAE/ESTAE routine creates and, initializes an SCB (STAE control block) to represent an abnormal interruption exit routine. The STAE/ESTAE routine can create, cancel, propagate, or overlay an SCB, according to the action codes passed as input. The STAE routine receives control from the SVC IH or via branch entry. Control returns to the caller.	IEAVTSIN	
1 The STAE/ESTAE routine validates both branch entered and SVC entered requests. ESTAE abnormally terminates invalid callers, passing a X'13C' ABEND code to the ABEND routine. The value in register 15 explicitly states the reason for the termination. STAE processing does not terminate callers requesting STAE, STAI, or SVC-entered ESTAE.	IEAVSTA0	
2 The STAE/ESTAE routine obtains a work area from the FREESCB queue for branch entries or uses an area in the STAE/ESTAE SVRB for SVC entries. If the FREESCB queue is full, a GETMAIN is issued for four cells from subpool 255 and added to the FREESCB queue.		

Extended Description	Module	Segment
3 The STAE/ESTAE routine performs requested service, as indicated in register 0.		
• For create requests, the STAE/ESTAE routine obtains a cell for an SCB. The newly created SCB is chained in the SCB queue, pointed to by the appropriate TCB. STAE/ESTAE indicates the caller owns the SCB by setting an indicator in the RBSCB field of the caller's RB. For STAI or ESTAI requests, STAE/ESTAE also propagates the STAI or ESTAI SCBs via propagate processing.		
• For cancel requests, the STAE/ESTAE routine dequeues the SCB from the specified TCB, returns the cell to the FREESCB queue, and zeroes the RBSCB indicator in the caller's RB if the caller does not own any more SCBs.		
• For overlay requests, the STAE/ESTAE routine initializes the existing SCB with the new values.		
• For propagate requests, the STAE/ESTAE routine obtains cells, copies the SCB information from the appropriate SCB (addressed by the TCB pointed to in register 4), and chains the SCB to the TCB being attached.		

Diagram 22-25. STAE/ESTAE Processing (IEAVSTA0) (Part 3 of 6)



Process

4 Return control the caller.

To Caller (Branch entry or Exit Prolog (IEAVEEXP))

Output

Register 15

Return Code

ESTAE/ESTAI/ESTAR

'00' – Successful STA or ESTA request.

'04' – ESTAE OV has been requested and the last SCB is:
1. Non-existant,
2. Not-owned by user's RB or
3. Is not an ESTAE exit
In this instance an ESTAE Create will be performed.

'0C' – Invalid cancel request.

'10' – Unexpected error.

'14' – Insufficient storage.

STAE/STAI

'00' – Successful STA or ESTA request.

'04' – Insufficient storage.

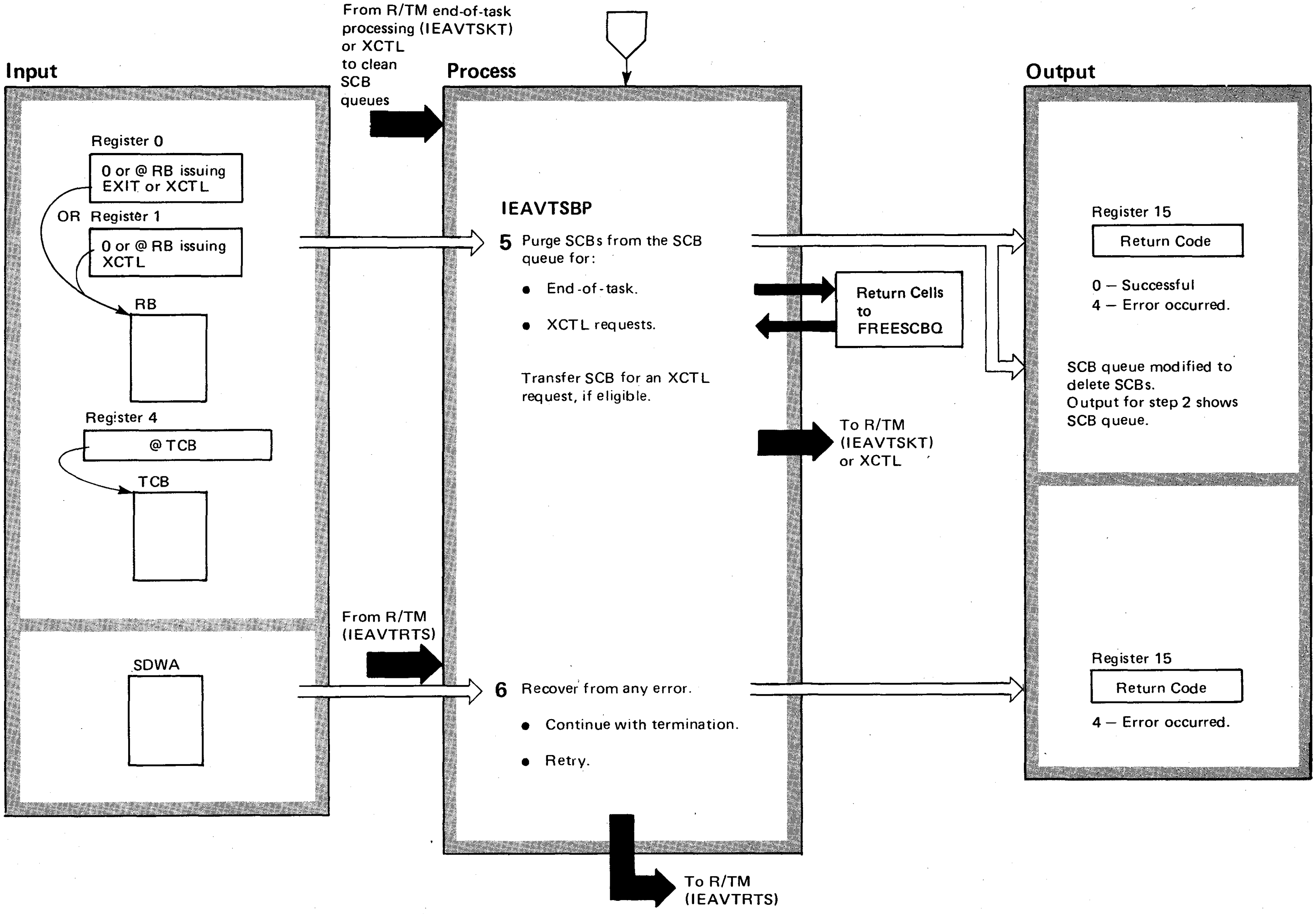
'08' – STAE issued in a STAE exit or
– cancel or overlay request with no SCB on Q.

'0C' – STAI not issued by attach or
– STAI request with missing TCB operand.

'10' – Cancel or overlay and SCB is not a STAE SCB or is not owned by requestor's RB or
– Unexpected error encountered while processing the request.

Extended Description	Module	Segment
4 STAE/ESTAE returns control to the caller, with return codes indicating the results of the request in register 15.		

Diagram 22-25. STAE/ESTAE Processing (IEAVSTA0) (Part 5 of 6)



IEAVTSBP

5 Purge SCBs from the SCB queue for:

- End-of-task.
- XCTL requests.

Transfer SCB for an XCTL request, if eligible.

6 Recover from any error.

- Continue with termination.
- Retry.

0 – Successful
4 – Error occurred.

SCB queue modified to delete SCBs. Output for step 2 shows SCB queue.

4 – Error occurred.

Diagram 22-25. STAE/ESTAE Processing (IEAVSTA0) (Part 6 of 6)

Extended Description	Module	Segment
5 The SCB task recovery resource manager (TRRM) removes or transfers an SCB to another RB as follows:	IEAVTSBP	TRMPROCS
• RBs issuing EXIT have their SCBs purged.		TRMFREE
• RBs issuing XCTL have their SCBs transferred or purged. The SCBs will be transferred if the caller issued the XCTL with the YES option.		
• For end-of-task requests, or if an ATTACH request fails, the TRRM purges the entire SCB QUEUE. The TRRM purges SCBs by returning the SCBs created by ESTAE/STAE processing to the FREESCBQ. The TRRM purges SCBs created by BRANCH entries by zeroing the SCB field in the SVRB. TRRM dequeues the SCBs from the SCB queue. (See the output from step 2, which shows the SCB queue.) Finally, the TRRM sets the RB indicator to indicate that no SCB is owned.		
6 The FRR attempts to recover from any errors that occur in the TRRM. It performs recovery as follows:	IEAVTSBP	TRRMFRR
• Continue with termination for memory switch conditions.		
• Zero the SCB queue pointer in the TCB if the caller requested a purge of all SCBs of the task.		
• For storage key failures and storage data checks, the FRR scans the queue for an SCB within the range indicated in the SDWA. If an FRR is found within the range, the FRR zeroes the queue pointer in the TCB.		
• Dequeues all SCBs owned by RBs for RB EXIT and XCTL requests when no SCBs fall within the range indicated in the SDWA.		
For retry requests, the FRR returns to the caller, with a return code of 4 in register 15.		

Diagram 22-26. Alternate CPU Recovery (ACR) Overview (IEAVTACR) (Part 1 of 4)

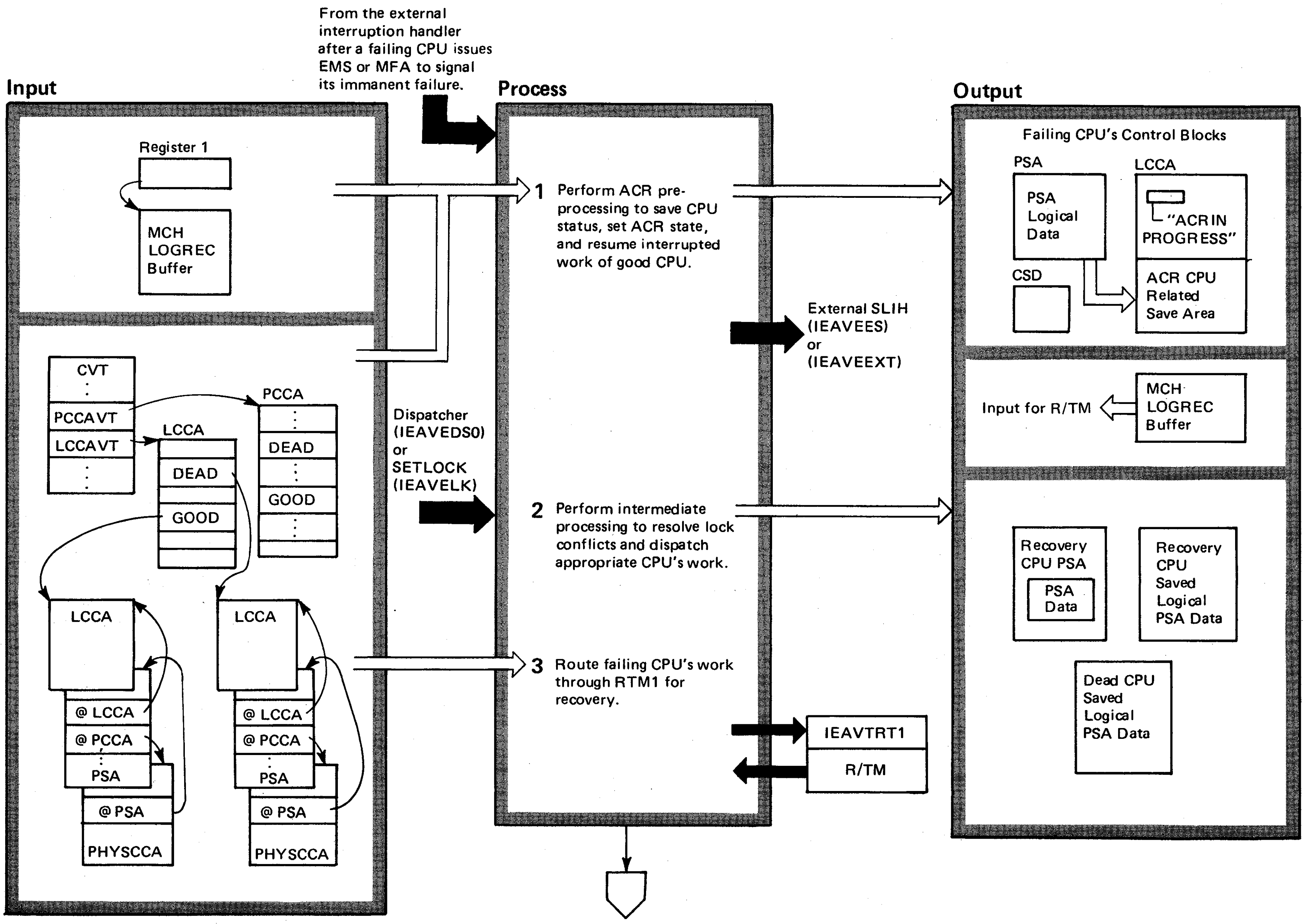

From the external interruption handler after a failing CPU issues EMS or MFA to signal its immanent failure.
Input
Process
Output
Register 1
MCH LOGREC Buffer
CVT
PCCAVT
LCCAVT
PCCA
LCCA
DEAD
GOOD
LCCA
@ LCCA
@ PCCA
PSA
@ PSA
PHYSCCA
LCCA
@ LCCA
@ PCCA
PSA
@ PSA
PHYSCCA
1 Perform ACR pre-processing to save CPU status, set ACR state, and resume interrupted work of good CPU.
External SLIH (IEAVEES) or (IEAVEEXT)
Dispatcher (IEAVEDS0) or SETLOCK (IEAVELK)
2 Perform intermediate processing to resolve lock conflicts and dispatch appropriate CPU's work.
3 Route failing CPU's work through RTM1 for recovery.
IEAVTRT1
R/TM
Failing CPU's Control Blocks
PSA
PSA Logical Data
LCCA
"ACR IN PROGRESS"
CSD
ACR CPU Related Save Area
Input for R/TM
MCH LOGREC Buffer
Recovery CPU PSA
PSA Data
Recovery CPU Saved Logical PSA Data
Dead CPU Saved Logical PSA Data

Diagram 22-26. Alternate CPU Recovery (ACR) Overview (IEAVTACR) (Part 2 of 4)

Extended Description	Module	Segment
Alternate CPU recovery (ACR) recovers the system on the remaining CPU when one CPU in a multiprocessing environment fails. ACR quiesces the operation of the failing CPU and attempts to recover as much processing as possible — ACR keeps the system operational.		
ACR processing begins when a CPU receives a signal, via an EMS (emergency signal) or an MFA (malfunction alert), of another CPU's imminent failure just before it stops operation. (See the M.O. diagram Signal Service Routines (IEAVERI) in the Supervisor Control section for a description of how CPUs signal one another.) ACR initially receives control from the external interruption handler and proceeds to recover the failing CPU's work by giving control to R/TM as if a machine check occurred. R/TM routes control to any FRRs defined by the abnormally terminated process. These FRRs free resources associated with the terminating functions. This provides as much recovery function as possible. As ACR processing continues, it cleans up resources associated with the failing CPU and frees them, where possible, for use by the system. The failing CPU is logically disconnected along with any devices affiliated with that CPU. ACR gives control to the dispatcher to again begin normal system operation.		

Extended Description	Module	Segment
1 ACR uses the LCCA and saves the PSA data of the failing CPU in the ACR save area. ACR extracts all logical fields from the failing CPU's PSA and saves them in the failing CPU's ACR save area. ACR then sets the "ACR in progress" indicator in the LCCAs of both the failing and recovery CPUs. The CSD also contains an "ACR in progress" indicator. Then, ACR marks the failing CPU offline by setting indicators in the CSD (common system data). The CPU remaining in the system continues processing its own work by returning control to the external interruption handler so the system continues processing. Work for the remaining CPU will be dispatched. When the recovery CPU enters the dispatcher, or when a lock conflict arises, ACR will resume processing.	IEAVTACR	ACRPREP
2 An entry from SETLOCK or the dispatcher causes a suspension of the currently executing work and a dispatch of the previously suspended work. Dispatching of the appropriate CPU's work will be accomplished by saving the logical data of the current PSA in the corresponding ACR save area and restoring the logical data of the suspended CPU's PSA back to the current PSA. Processing of work can then resume.		ACRLKSPI
3 The first entry from SETLOCK or the dispatcher causes the failing CPU's work to become the work to be dispatched. ACR treats this work as a machine check condition, by routing control to RTM1 with the machine check indication. FRRs defined for the terminated process will receive control to provide some level of recovery (including the releasing of locks held or the retry of a process, if appropriate).		

Diagram 22-26. Alternate CPU Recovery (ACR) Overview (IEAVTACR) (Part 3 of 4)

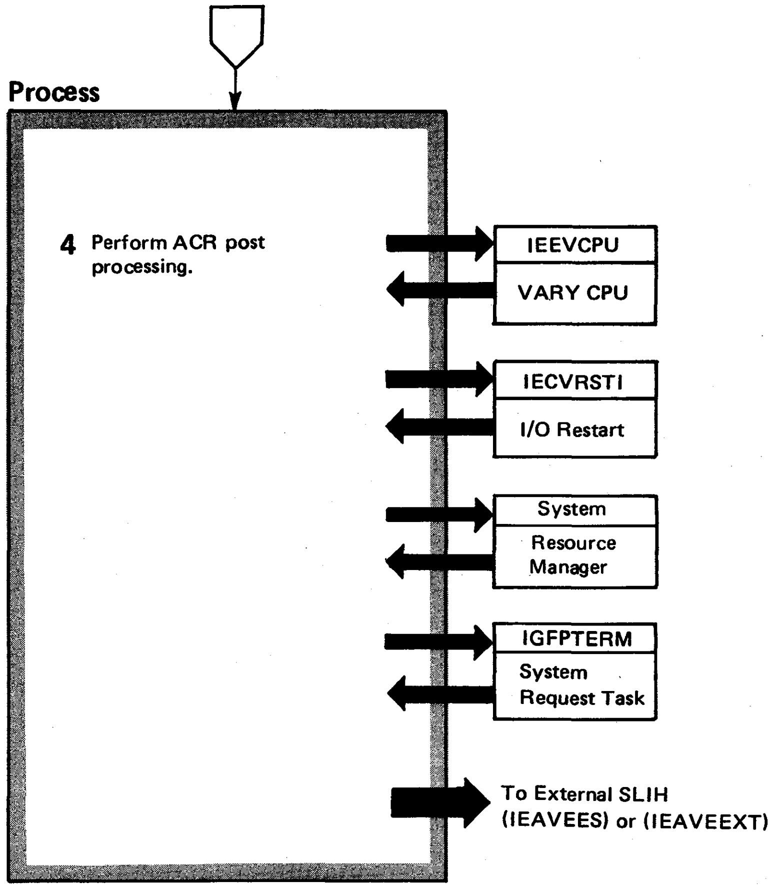


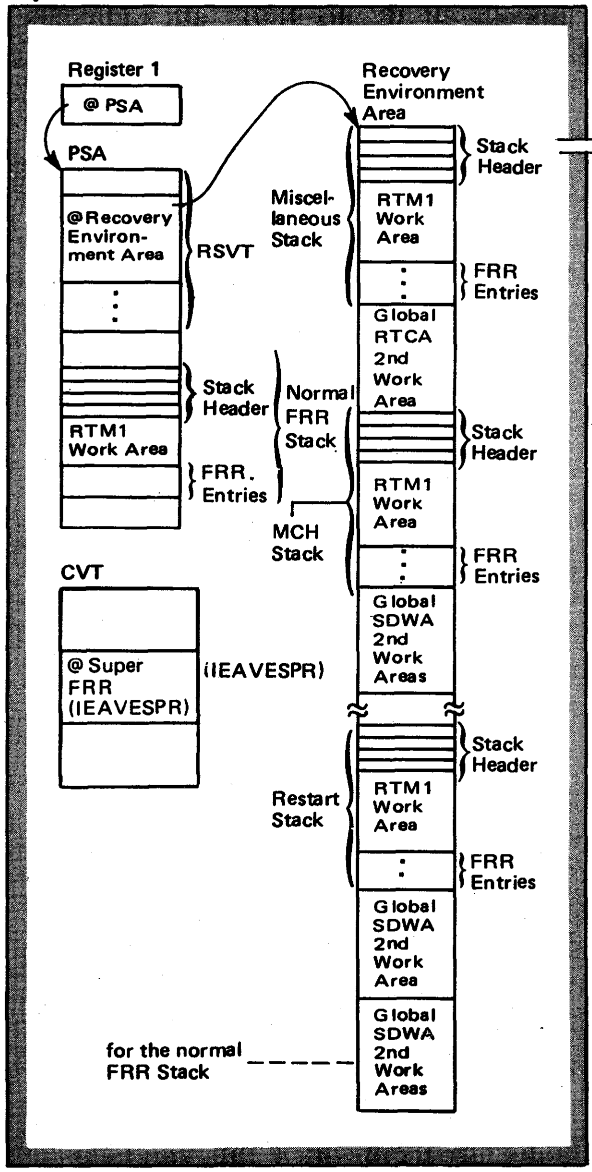
Diagram 22-26. Alternate CPU Recovery (ACR) Overview (IEAVTACR) (Part 4 of 4)

Extended Description	Module	Segment
4 When ACR finds that both CPU's processes are physically enabled, it cleans up I/O device requests, switches consoles if necessary, notifies the system operator, and notifies the System Resource Manager that one CPU in the system has failed. The system can now process normally, even though one CPU now performs the work done by two.		ACRPOSTP
The dispatcher will continue to dispatch tasks as usual, with no consideration that ACR processing has occurred.		

Diagram 22-27. FRR Stack Initialization (IEAVTSIN) (Part 1 of 2)

From NIP (IEAVNIPO)
or Vary CPU (IEEVCPU)

Input



Process

1 Locate the Recovery Environment Area.

2 Locate the FRR Stack.

3 Initialize the RSVT.

4 Initialize the FRR Stack.

5 Zero the remaining portion of the stack and the SDWA.

- Go to Step 2 until all the FRR stacks have been initialized.

Step 2

6 Exit to caller when complete.

NIP (IEAVNIPO) or Vary CPU (IEEVCPU)

Output

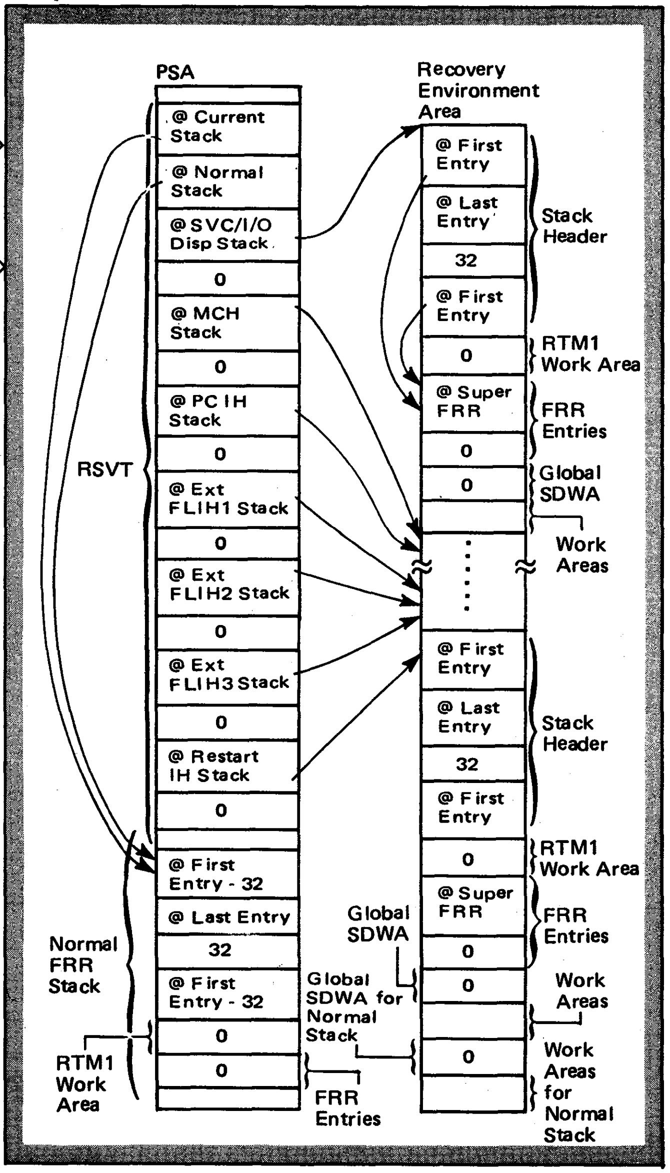
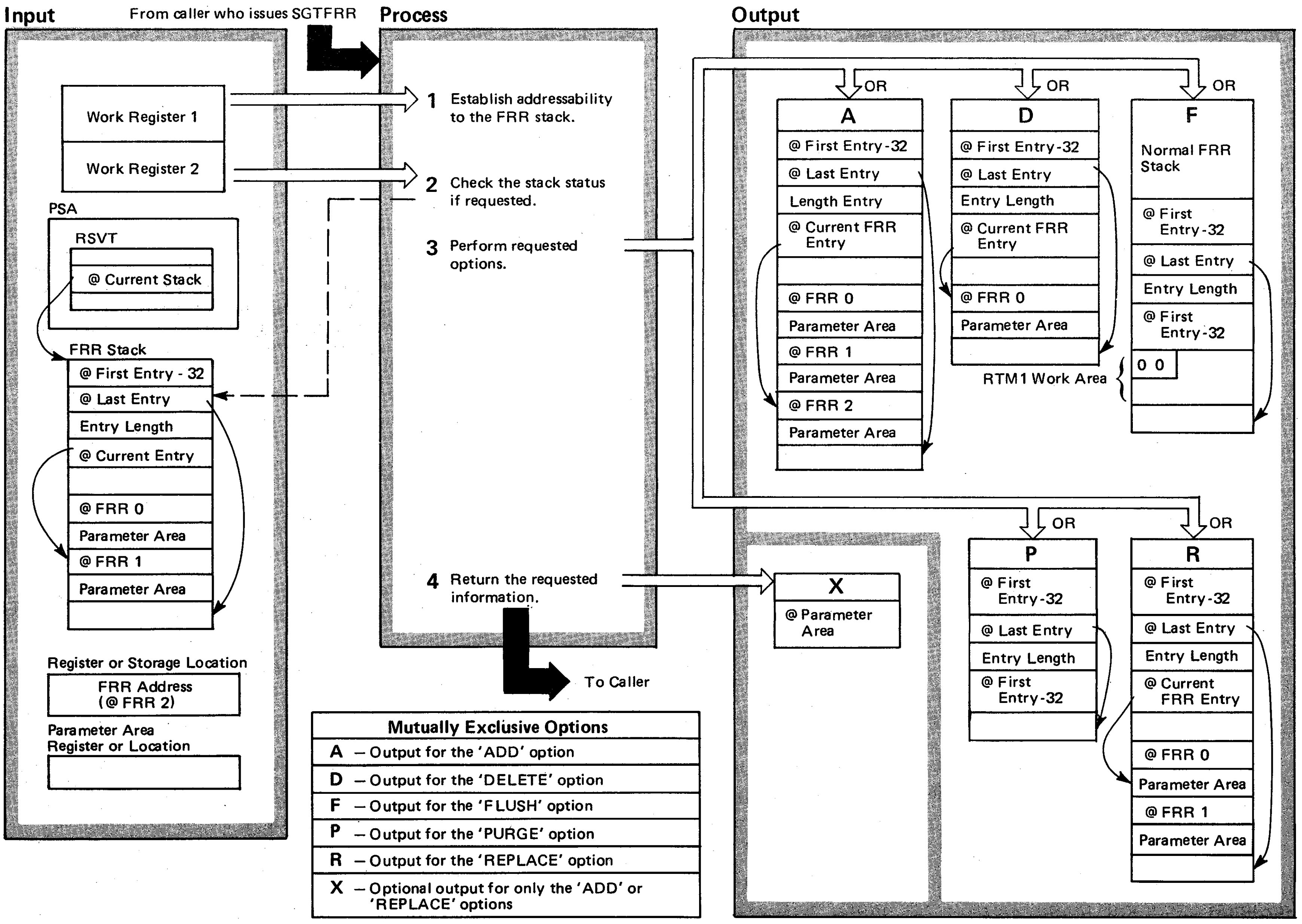


Diagram 22-27. FRR Stack Initialization (IEAVTSIN) (Part 2 of 2)

Extended Description	Module	Segment
The IEAVTSIN macro instruction expands inline to initialize all FRR recovery stacks in the system and identify each such stack by initializing the recovery stack vector table (RSVT) in the PSA (prefix storage area). VARY CPU and system initialization use IEAVTSIN.		
1 The first word of the RSVT points to the recovery environment area.	IEAVTSIN	
2 The initialization routine locates a recovery stack in the recovery environment area for initialization.		
3 The initialization routine places the address of the recovery stack into its appropriate slot in the RSVT.		
4 The recovery stack is initialized as follows: ● The four words in the stack header contain 1) The address of the first entry – 32 (if the normal stack) The address of the first entry (if not the normal stack) 2) The address of the last FRR entry 3) The FRR entry length 4) The address of the current entry. ● Initialization zeroes the FRR address field in the first entry of the stack for the normal stack; otherwise, it initializes this FRR address field with the address of the super FRR (obtained from the CVT). The remaining fields in this entry are zeroed.		
5 The initialization routine zeroes the remaining portions of the stack as follows: a) The RTM1 work area portion of the stack is zeroed. b) All FRR entries from the second entry to the last entry are zeroed. The global SDWA associated with this stack is zeroed. The work areas associated with the global SDWA consist of two types: a) A 72-byte save area is zeroed. b) A 200-byte FRR work area is not zeroed.		
6 Return to caller occurs if all recovery stacks have been initialized; otherwise, control returns to step 2 to initialize the next recovery stack.		

Diagram 22-28. SETFRR (SETFRR) (Part 1 of 2)



1 Establish addressability to the FRR stack.

2 Check the stack status if requested.

3 Perform requested options.

4 Return the requested information.

Mutually Exclusive Options	
A	– Output for the 'ADD' option
D	– Output for the 'DELETE' option
F	– Output for the 'FLUSH' option
P	– Output for the 'PURGE' option
R	– Output for the 'REPLACE' option
X	– Optional output for only the 'ADD' or 'REPLACE' options

Diagram 22-28. SETFRR (SETFRR) (Part 2 of 2)

Extended Description	Module*	Segment
The SETFRR macro instruction expands inline and alters the contents of an appropriate FRR stack based on given options.		
1 One of the two input work registers contains the information needed to establish addressability to the FRR stack.	SETFRR	
2 The other work register contains the information necessary to examine the "stack header" – the first four words of the FRR stack. SETFRR determines the stack status as follows (and only for the ADD, REPLACE, or DELETE options):		
A – If the first and fourth words of the stack header are equal *the FRR stack is empty.*		
B – If the second and fourth words of the stack header are equal *the FRR stack is full.*		
3 Five mutually exclusive options can be performed by SETFRR, as follows:		
• **ADD** – The FRR address supplied as input is added to the stack and the current FRR entry pointer is updated to point to this new FRR address. If the stack is full, a X'07D' ABEND will occur if the caller requests another FRR to be added.		
• **REPLACE** – Performs a replacement of the FRR address pointed to by the fourth word of the stack header by the input FRR address. If the FRR stack is *empty*, an addition equivalent to A is performed.		

	Module*	Segment
• **DELETE** – Removes an FRR address from the stack by adjusting the fourth word of the stack header to point to the preceding FRR entry. If the stack is empty this delete function is a NOP.		
• **PURGE** – Adjusts the stack header to reflect an empty stack (i.e., setting the fourth word equal to the first word of the stack header).		
• **FLUSH** – A special option to be used only by the Dispatcher, purges the normal FRR stack (making it empty) and zeroes R/TM recursion indicators in the RTM1 work area portion of the normal FRR stack.		
4 An optional parameter register or storage location, when specified as input, becomes the receiver of the address of the parameter area associated with the FRR address for which the "ADD" or "REPLACE" option is to be executed.		

Notes:

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

2) SETFRR operates on a supervisor control FRR stack identical to that described for a normal FRR stack.

* SETFRR expands inline; it has no service routine module.

Diagram 22-29. SVC 51 Overview (IEAVAD00) (Part 1 of 2)

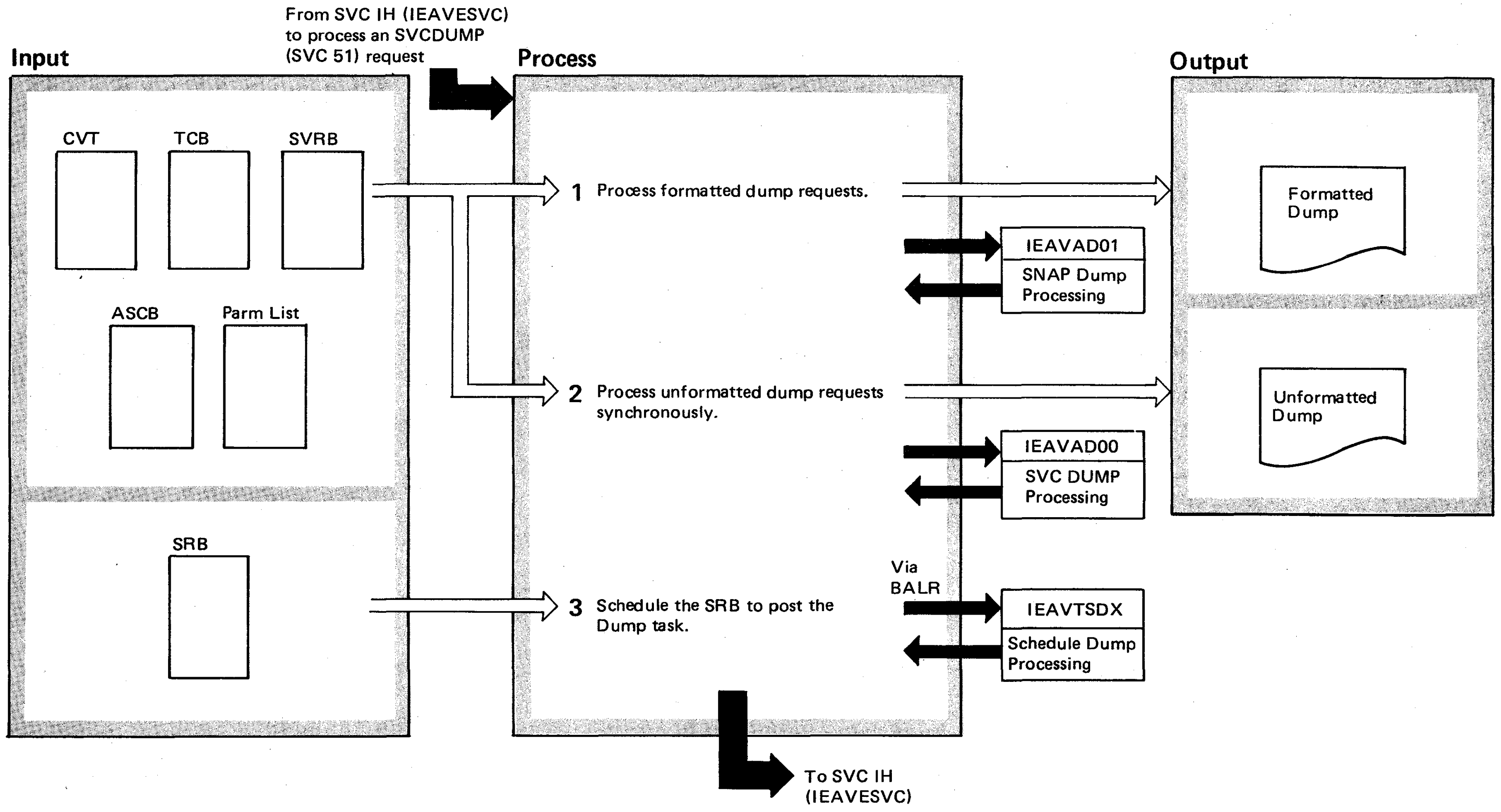


Diagram 22-29. SVC 51 Overview (IEAVAD00) (Part 2 of 2)

Extended Description	Module	Segment
An SVC 51 instruction provides linkage to both the SNAP function and to the SVC DUMP function. Both functions require Register 1 to point to a parameter list.		
The difference between a SNAP and SVC DUMP parameter list is in byte 1 of the first word (B) and byte 0 of the third word (C).		

B=X'00' ,C=X'00'	OS/VS2 Release 1 SNAP Parameter List
B=B'01 . . . . . . '	OS/VS2 Release 2 SNAP Parameter List
B=X'80'	OS/VS2 Release 2 SVC DUMP Parameter List
B=X'00' ,C=X'80'	INVALID – OS/VS2 Release 1 SVC DUMP Parameter List

Extended Description	Module	Segment
1 The "SNAP DUMP Processing" M.O. diagram describes the processing of a formatted dump.	IEAVAD00	SDTOP
2 Callers of SVC DUMP must be authorized by APF or have control program key. If the caller is not authorized he will be abnormally terminated with completion code 133.		SDTOP

SVC DUMP provides two services, a Synchronous Dump and a Schedule Dump. The distinction between the two dumps is in the 6th word of the parameter list. D is the first halfword, and E is the second halfword of the 6th word.

D=X'0000' ,E=X'0000' – SYNCHRONOUS DUMP
D=X'0000' ,E=ASID – SCHEDULE DUMP
D=CURRENT ASID,E=ASID – Target of SCHEDULE DUMP, process as if a SYNCHRONOUS DUMP request.

A Synchronous Dump will be taken now off the current TCB.

A Schedule Dump results in a branch to the Schedule Dump routine.

If an invalid parameter list is passed, the caller is abended with a 233 completion code.

3 SVC DUMP processing (IEAVAD00) describes the processing for a dump scheduled to the dump task in each address space.

Diagram 22-30. SNAP Dump Processing (IEAVAD01) (Part 1 of 6)

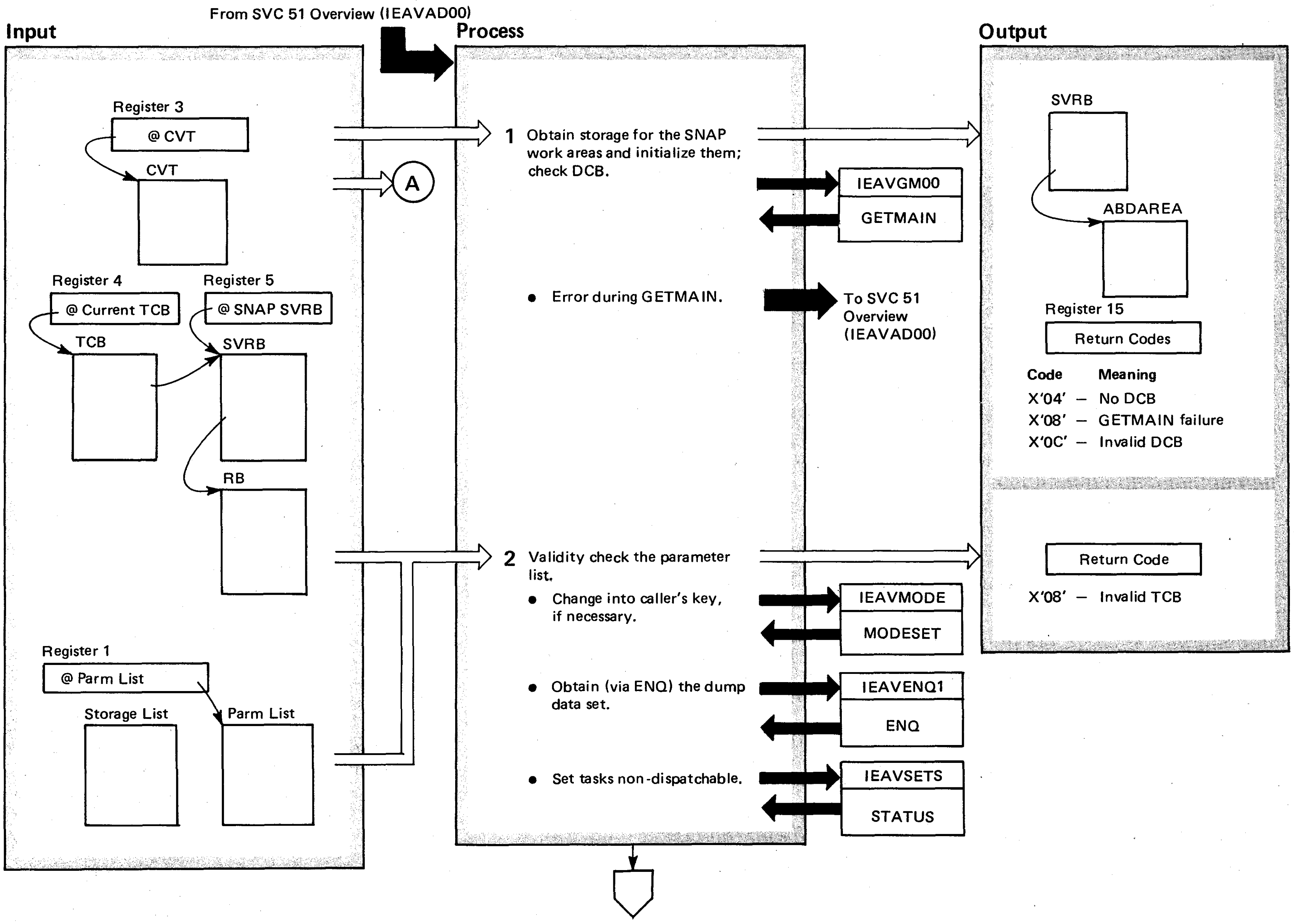


Diagram 22-30. SNAP Dump Processing (IEAVAD01) (Part 2 of 6)

Extended Description	Module	Segment
The SNAP dump routines produce a formatted dump of various areas of storage, depending on the parameters. As shown in M.O. diagram SVC 51 Overview (IEAVAD00), SNAP receives control via an SVC 51 macro instruction.		
The main SNAP module, IEAVAD01, does initialization for and then routes to various formatting routines. These routines format the dump.		
1 The SNAP routine obtains storage, via GETMAIN, for an ABDAREA. The ABDAREA contains the information used by the formatting routines. Control goes to the caller if an error occurs during GETMAIN processing.	IEAVAD01	
2 SNAP processing does not validity check the parameter list for calls from ABEND, nor does SNAP enqueue upon the dump data set for calls from ABEND. The enqueue process has already been performed by ABDUMP.		
STATUS is issued if the task being dumped is not the current task.		

Diagram 22-30. SNAP Dump Processing (IEAVAD01) (Part 3 of 6)

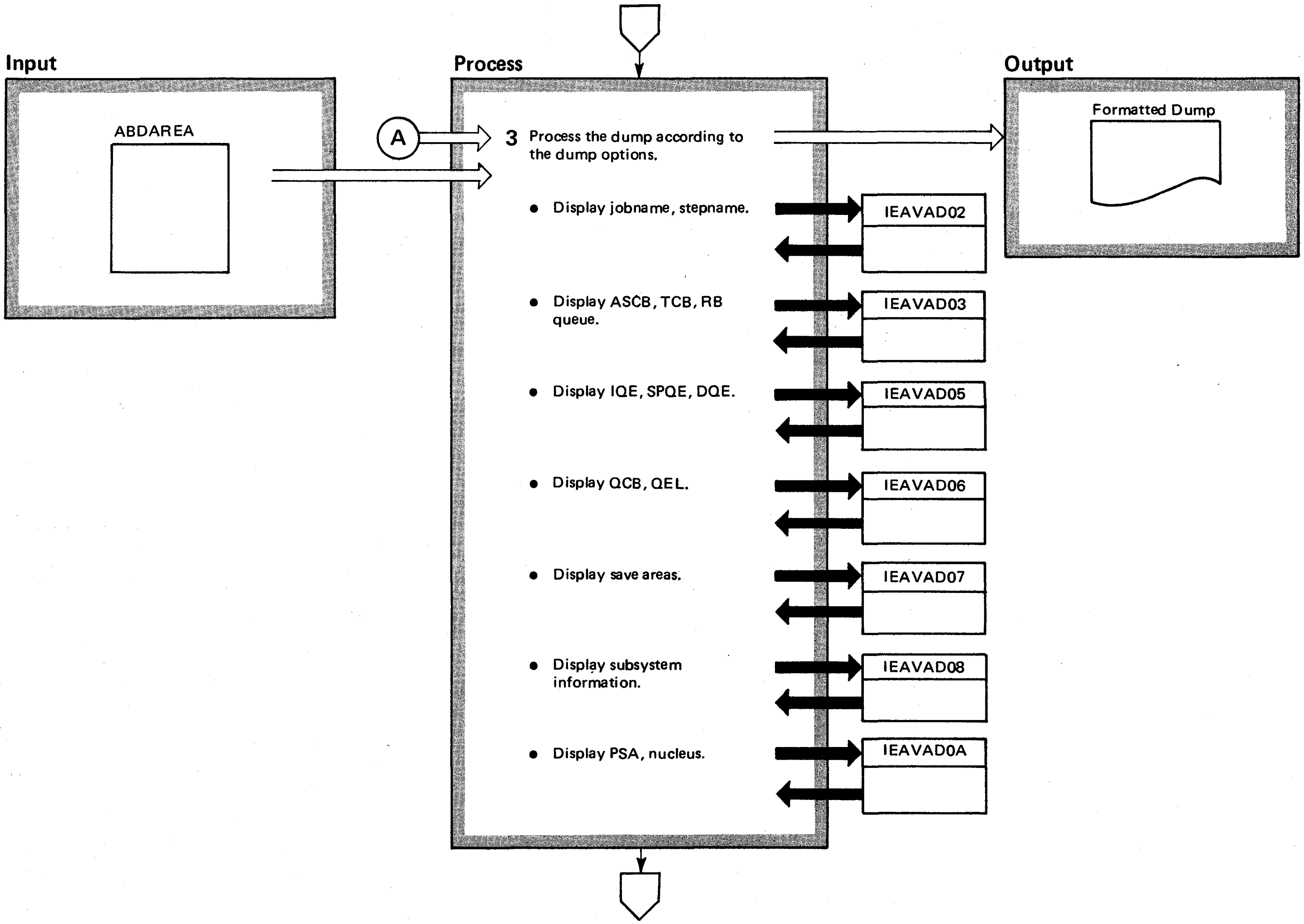
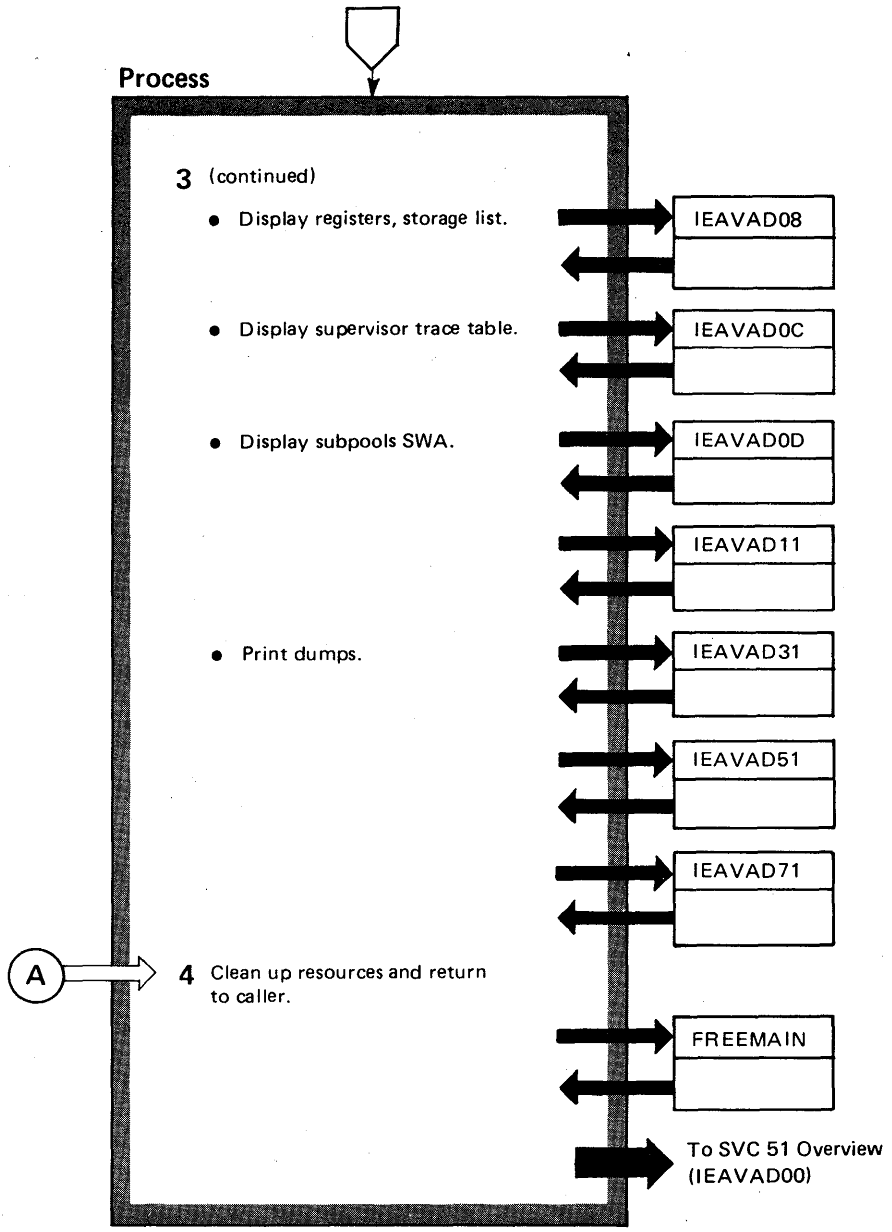

Input
ABDAREA
Process
A
3 Process the dump according to the dump options.
● Display jobname, stepname.
IEAVAD02
● Display ASCB, TCB, RB queue.
IEAVAD03
● Display IQE, SPQE, DQE.
IEAVAD05
● Display QCB, QEL.
IEAVAD06
● Display save areas.
IEAVAD07
● Display subsystem information.
IEAVAD08
● Display PSA, nucleus.
IEAVAD0A
Output
Formatted Dump

Diagram 22-30. SNAP Dump Processing (IEAVAD01) (Part 4 of 6)

Extended Description	Module	Segment
3 SNAP routes control to the formatting routines, based on the information in the ABDAREA. This list shows the formatting module IEAVAD01 combination:		
• IEAVAD02. This formatting module displays the jobname, stepname, time, date, ID, completion code, PSW, ILC (instruction length count), and interruption code.	IEAVAD02	
• IEAVAD03. This formatting module displays the ASCB, TCB, RB queue, LLE queue, CDE, XTLIST, DEB, and TIOT.	IEAVAD03	
• IEAVAD05. This formatting module displays the IQE, SPQE, DQE, FQE, PQE, and FBQE.	IEAVAD05	
• IEAVAD06. This formatting module displays the QCB and QEL.	IEAVAD06	
• IEAVAD07. This formatting module displays the save areas.	IEAVAD07	
• IEAVAD08. This module acts as the interface between SNAP and the formatting routines for TCAM, GTF, VTAM, VSAM, and an installation defined formatting routine.	IEAVAD08	
• IEAVADOA. This formatting module displays the PSA, nucleus, SQA, and LSQA.	IEAVADOA	

Diagram 22-30. SNAP Dump Processing (IEAVAD01) (Part 5 of 6)



Process

3 (continued)

- Display registers, storage list.
- Display supervisor trace table.
- Display subpools SWA.
- Print dumps.

4 Clean up resources and return to caller.

Diagram 22-30. SNAP Dump Processing (IEAVAD01) (Part 6 of 6)

Extended Description	Module	Segment
• IEAVAD0B. This formatting module displays registers, storage lists, JPA modules, active SVCs, and LPA modules.	IEAVAD0B	
• IEAVAD0C. This formatting module displays the supervisor trace table.	IEAVAD0C	
• IEAVAD0D. This formatting module displays the subpools 0-127 and SWA. It also displays subpools 229 and 230 when LSQA is requested.	IEAVAD0D	
• IEAVAD11. This formatting module prints the lines of the dump on an output device.	IEAVAD11	
• IEAVAD31. This formatting module unpacks and translates data in the print line, providing indentation.	IEAVAD31	
• IEAVAD51. This module translates data in the print line.	IEAVAD51	
• IEAVAD71. This module prints blocks of storage	IEAVAD71	
4 Prior to returning control to the caller, SNAP cleans the resources it used.		

Diagram 22-31. SVC Dump Processing (IEAVAD00) (Part 1 of 6)

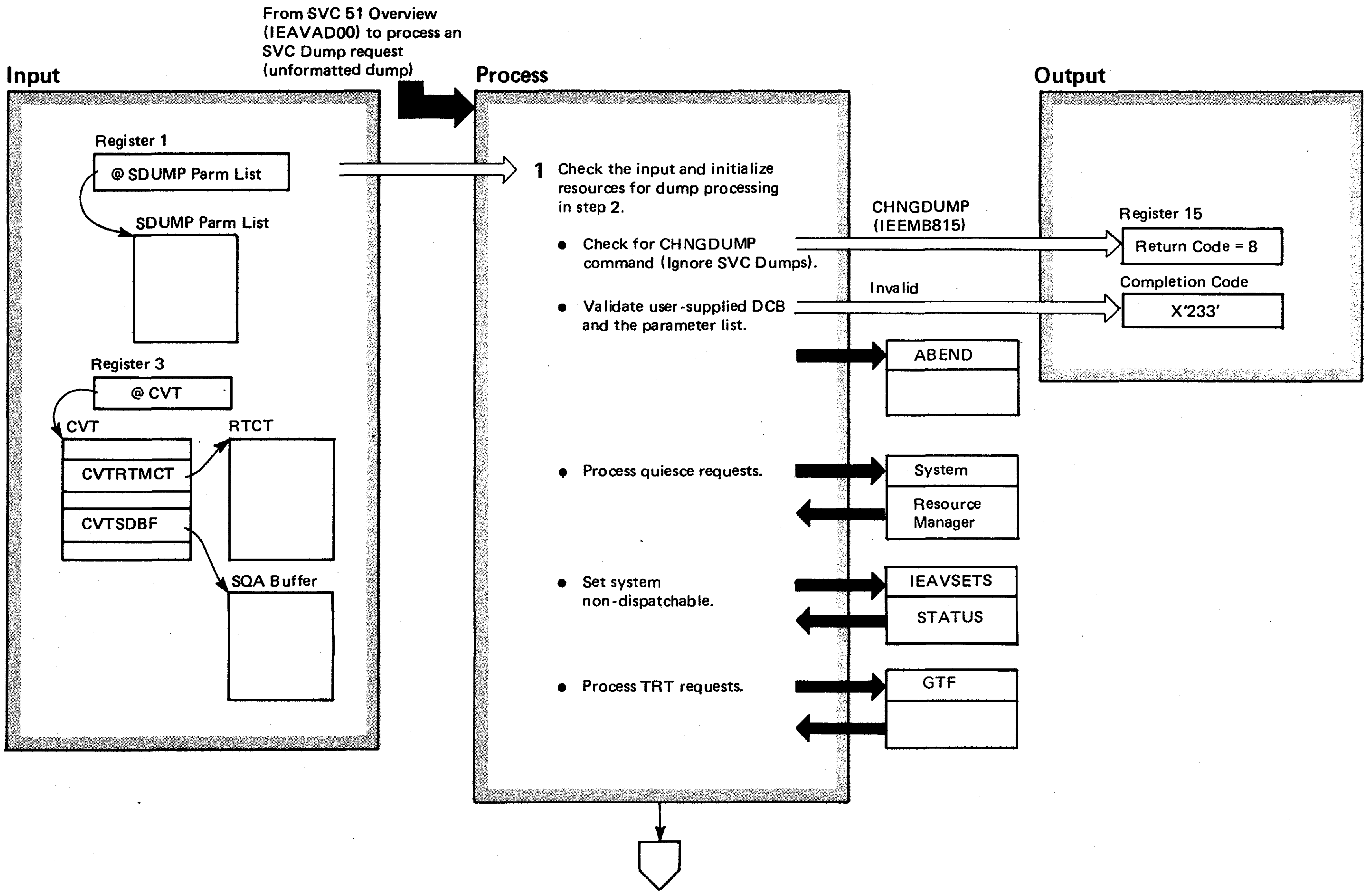

From SVC 51 Overview (IEAVAD00) to process an SVC Dump request (unformatted dump)
Input
Process
Output
Register 1
@ SDUMP Parm List
SDUMP Parm List
Register 3
@ CVT
CVT
CVTRTMCT
CVTSDBF
RTCT
SQA Buffer
1 Check the input and initialize resources for dump processing in step 2.
• Check for CHNGDUMP command (Ignore SVC Dumps).
• Validate user-supplied DCB and the parameter list.
• Process quiesce requests.
• Set system non-dispatchable.
• Process TRT requests.
CHNGDUMP (IEEMB815)
Invalid
ABEND
System Resource Manager
IEAVSETS
STATUS
GTF
Register 15
Return Code = 8
Completion Code
X'233'

Diagram 22-31. SVC Dump Processing (IEAVAD00) (Part 2 of 6)

Extended Description	Module	Segment
The SVC Dump routine will create a synchronous, unformatted dump, and write it on a data set. As in "SNAP Dump Processing," SVC dump receives control via an SVC 51 macro instruction. However, SVC dump receives control when the parameter list addressed in register 1 indicates an unformatted dump.		
1 SVC dump determines whether any CHNGDUMP (see the M.O. diagram CHNGDUMP Routine (IEEMB815)) operands override the parameters passed. If the CHNGDUMP command has been issued to override the SVC dump options, the SVC dump routine passes the caller a return code of 8 in register 15. Then, SVC dump checks for invalid user-supplied DCBs (data control blocks, that define the data set that will receive the dump), and terminates those callers. The system resource manager, STATUS, and GTF perform services for SVC dump, according to the original request.	IEAVAD00	POSSIBLE SDVALID SDENVIR

Diagram 22-31. SVC Dump Processing (IEAVAD00) (Part 3 of 6)

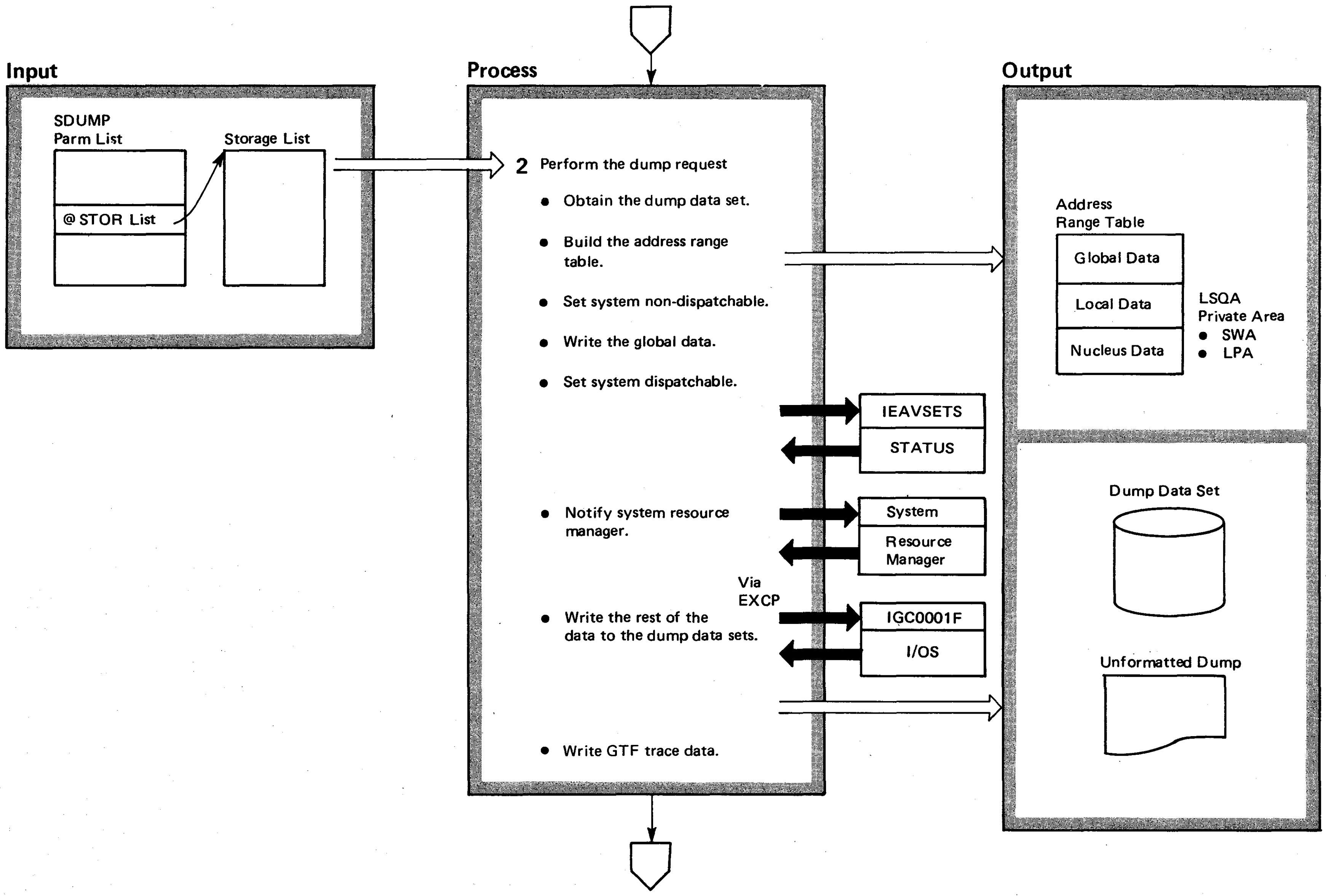
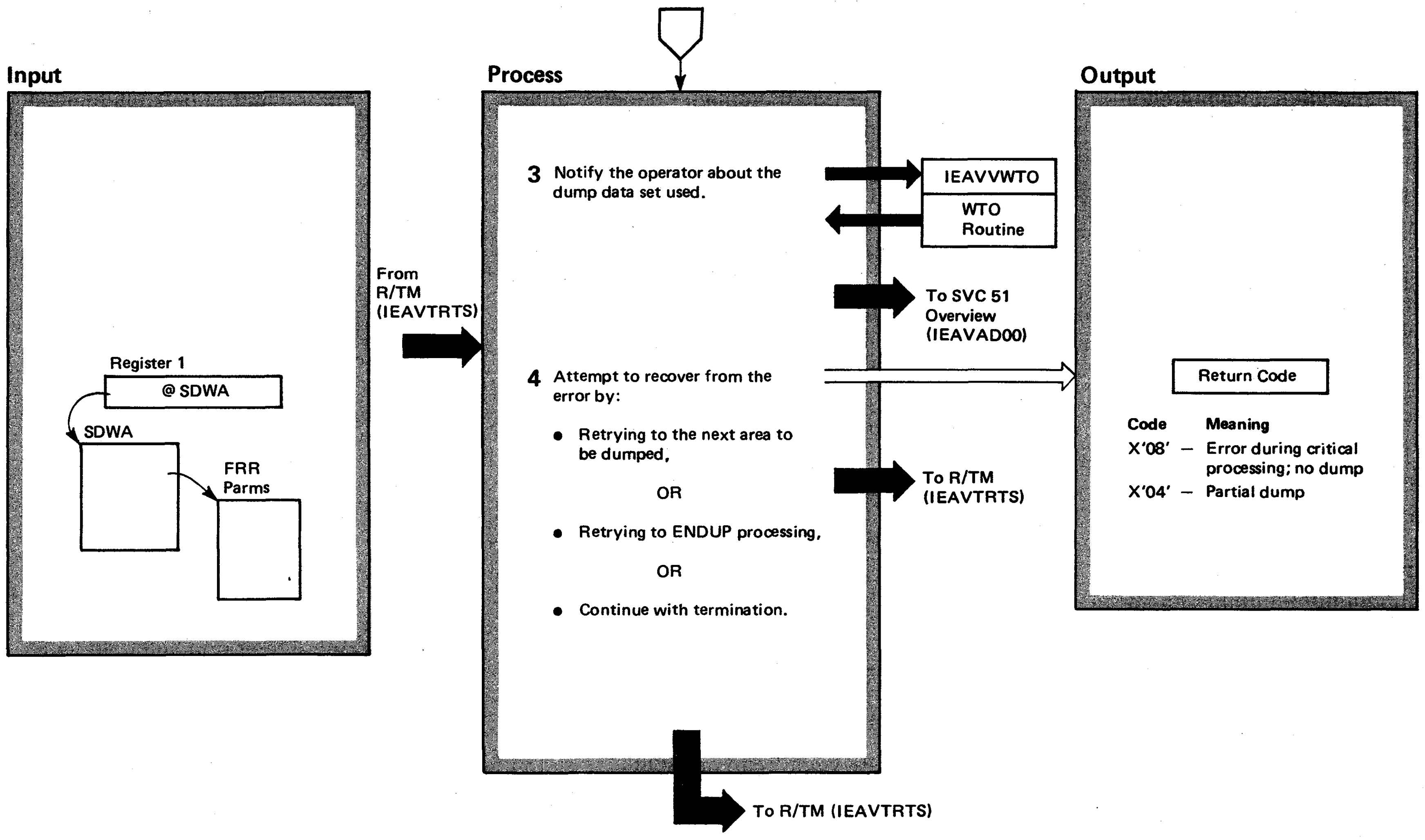


Diagram 22-31. SVC Dump Processing (IEAVAD00) (Part 4 of 6)

Extended Description	Module	Segment
2 SVC dump writes the dump, using EXCP (execute channel program) to the data set. An address range table, based on information in the SDUMP parameter list, delimits the address range of the dump.		SDIO

Diagram 22-31. SVC Dump Processing (IEAVAD00) (Part 5 of 6)



3 Notify the operator about the dump data set used.

4 Attempt to recover from the error by:

- Retrying to the next area to be dumped,

  OR

- Retrying to ENDUP processing,

  OR

- Continue with termination.

Code	Meaning
X'08'	– Error during critical processing; no dump
X'04'	– Partial dump

Diagram 22-31. SVC Dump Processing (IEAVAD00) (Part 6 of 6)

Extended Description		Module	Segment
3	The operator receives notification of the data set used for the dump.		WRITEMSG
4	An FRR (functional recovery routine) protects SVC dump processing.		SDFRR

Diagram 22-32. Schedule Dump Processing (IEAVTSDX) (Part 1 of 4)

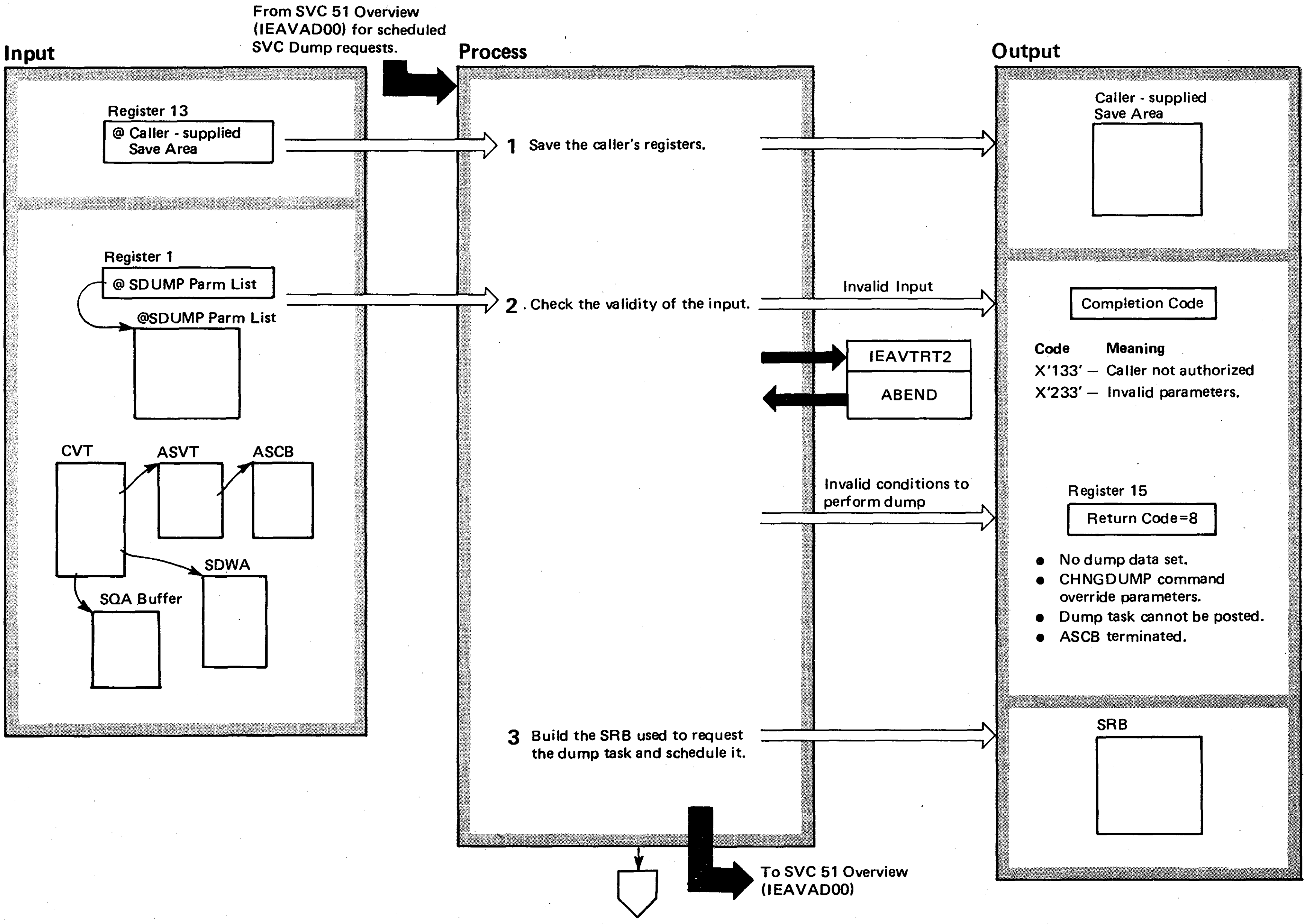

From SVC 51 Overview (IEAVAD00) for scheduled SVC Dump requests.
Input
Process
Output
Register 13
@ Caller - supplied Save Area
1 Save the caller's registers.
Caller - supplied Save Area
Register 1
@ SDUMP Parm List
@SDUMP Parm List
2 . Check the validity of the input.
Invalid Input
Completion Code
Code Meaning
X'133' – Caller not authorized
X'233' – Invalid parameters.
IEAVTRT2
ABEND
CVT
ASVT
ASCB
SDWA
SQA Buffer
Invalid conditions to perform dump
Register 15
Return Code=8
● No dump data set.
● CHNGDUMP command override parameters.
● Dump task cannot be posted.
● ASCB terminated.
3 Build the SRB used to request the dump task and schedule it.
SRB
To SVC 51 Overview (IEAVAD00)

Diagram 22-32. Schedule Dump Processing (IEAVTSDX) (Part 2 of 4)

Extended Description	Module	Segment
The dump task receives control from SVC 51 to dump contents of an address space.		
1 The first routine of schedule dump processing, module IEAVTSDX, saves the caller's registers.	IEAVTSDX	
2 Callers with invalid input are terminated with either a X'133' or X'233' completion code.		SCHVALID
3 The SCHEDULE macro is issued to schedule an SRB to give control to the address space-resident dump task, module IEAVTSDT. After SCHEDULE has scheduled the SRB, control returns to the caller.		SCHSRB

Diagram 22-32. Schedule Dump Processing (IEAVTSDX) (Part 3 of 4)

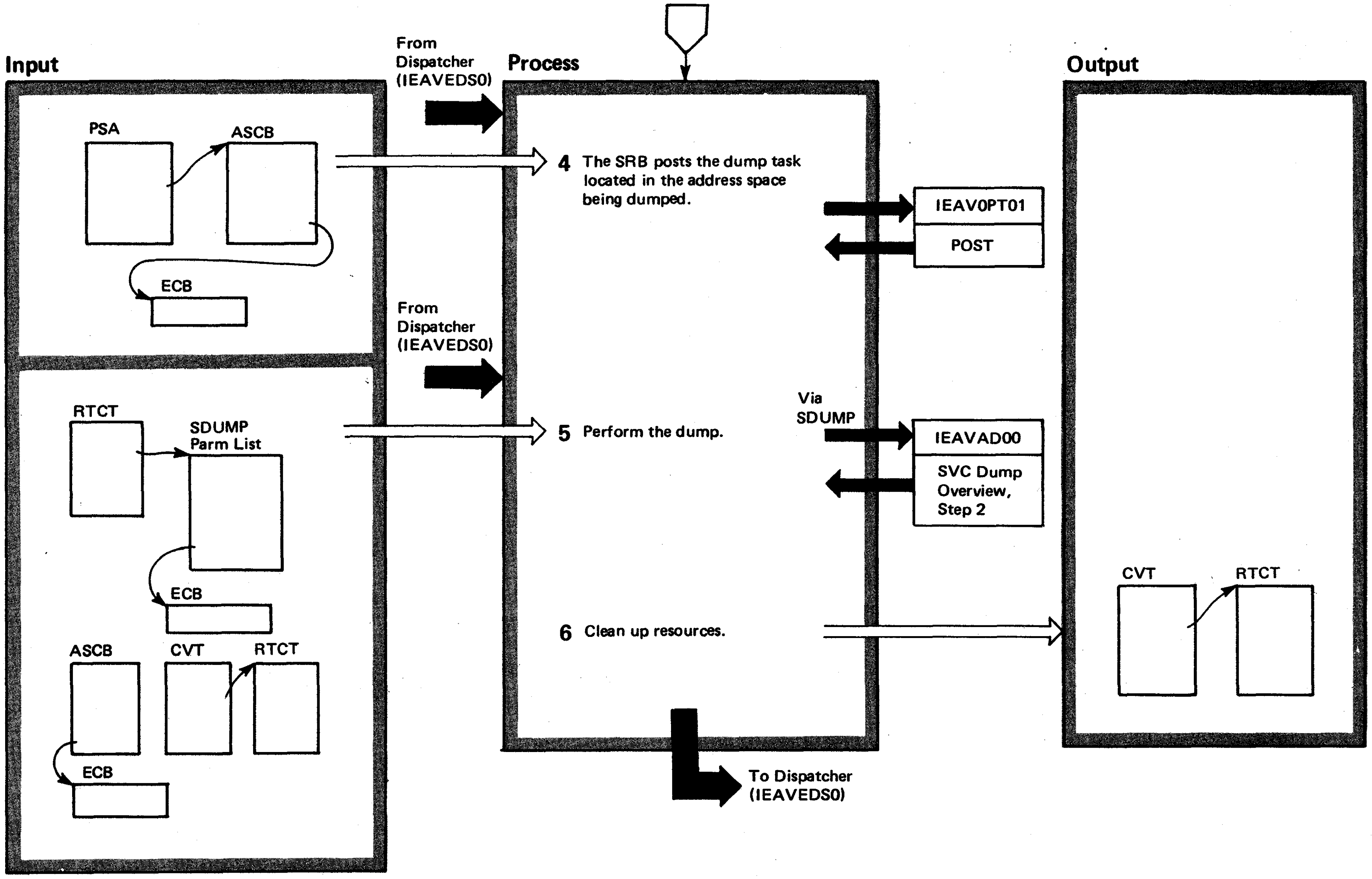


Diagram 22-32. Schedule Dump Processing (IEAVTSDX) (Part 4 of 4)

Extended Description		Module	Segment
4	The SRB, created in step 3, posts the ECB for the dump task located in the address space being dumped.		SCHSRB
5	The resident dump task receives control.	IEAVTSDT	
6	After cleanup, control returns to the caller.		

Diagram 22-33. CHNGDUMP Routine (IEEMB815) (Part 1 of 4)

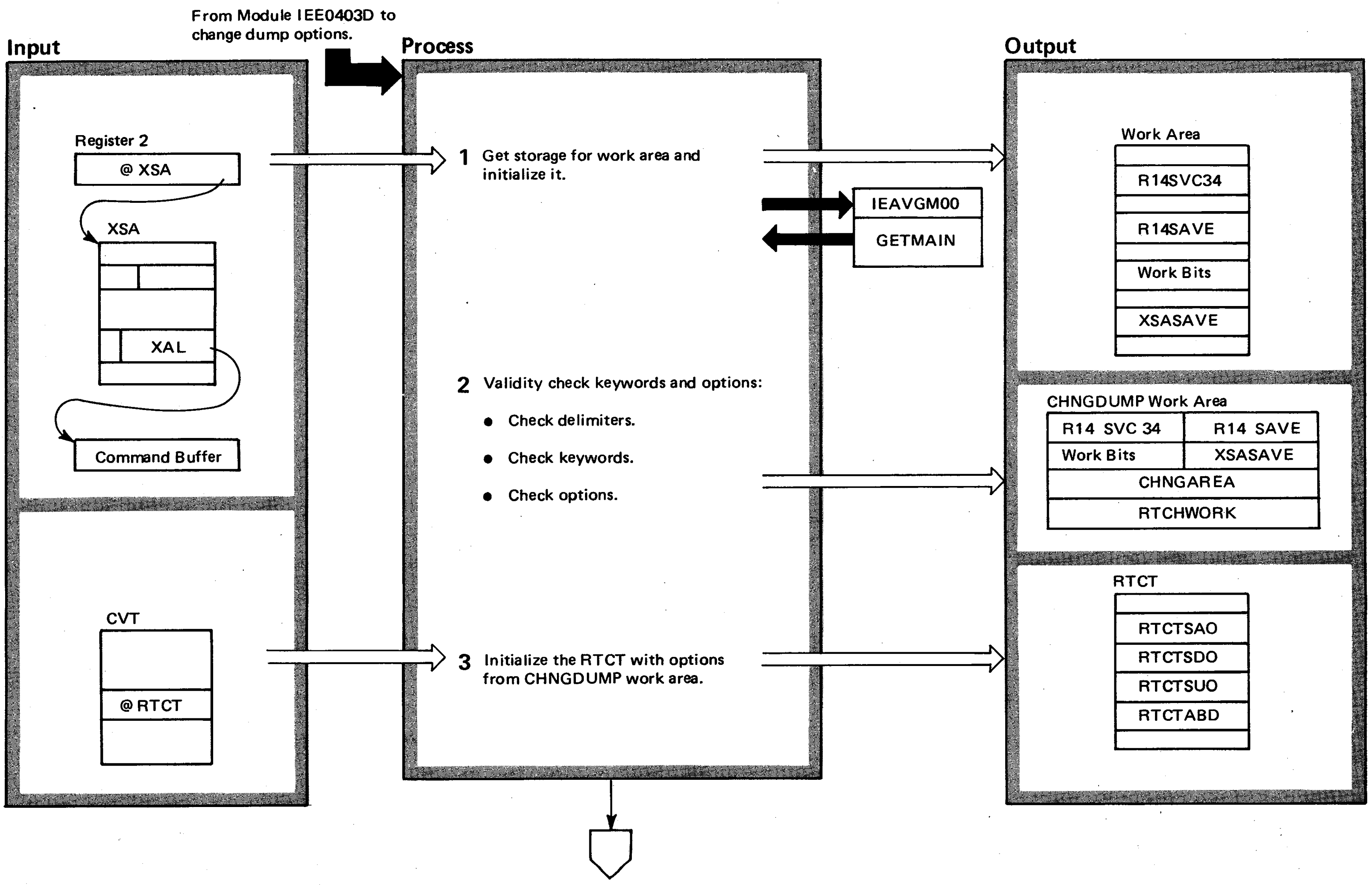
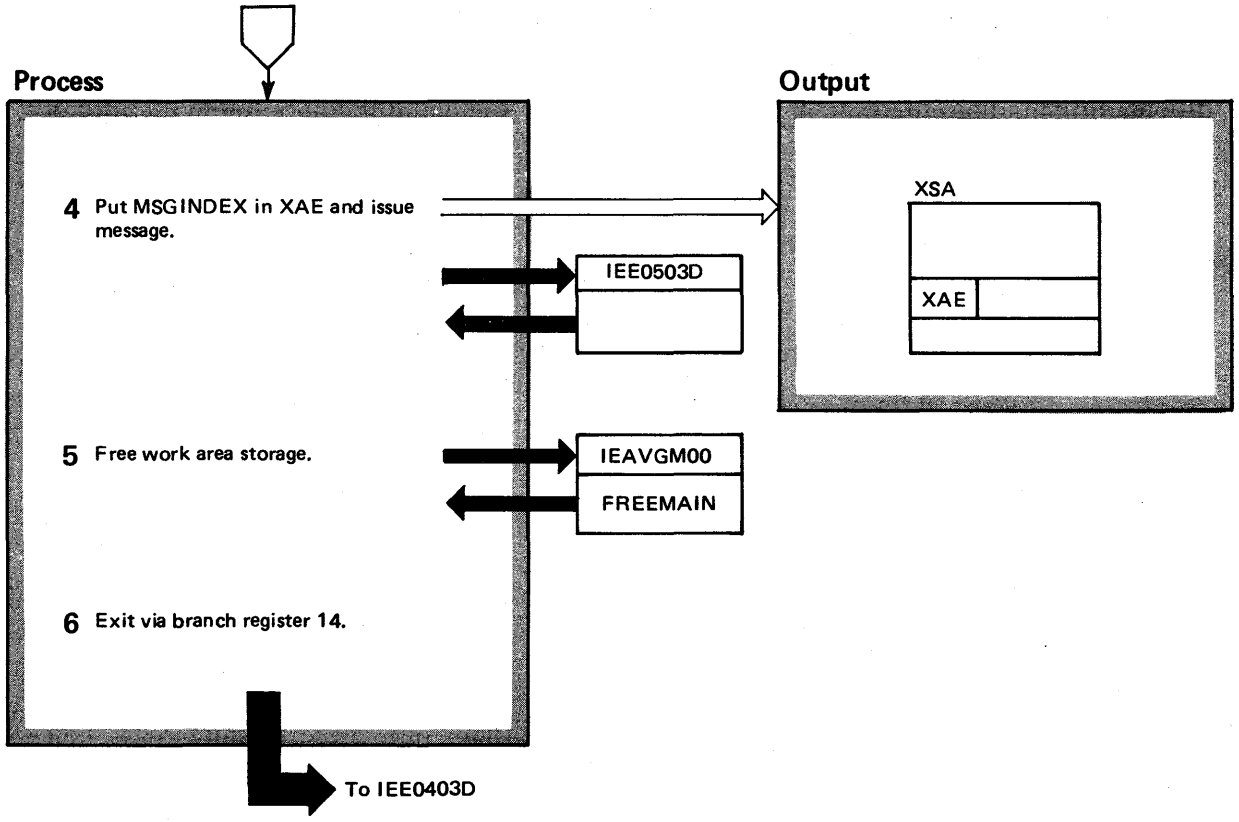


Diagram 22-33. CHNGDUMP Routine (IEEMB815) (Part 2 of 4)

Extended Description	Module	Segment
The CHNGDUMP routine processes the CHNGDUMP operator command which overrides any dump options that exist in the system. These options vary according to the type of dump originally requested. For SYSABEND and SYSUDUMP requests, the dump options which exist in the system are a result of merging all of the following:		
• IEAABD00 or IEADMP00 SYS1.PARMLIB members.		
• Options indicated on the ABEND macro instruction requests.		
• Options indicated on the CALLRTM macro instruction requests.		
• Options indicated on the SETRP macro instructions requested by recovery exits.		
For SVCDUMP requests, the dump options which exist in the system are those indicated on the SDUMP parameter list passed to the SVCDUMP routines.		
The XSA (extended save area) of the SVC 34 SVRB acts as the communications area between the SVC 34 router module (IEE0403D), and the various command processors, such as CHNGDUMP.		

Extended Description	Module	Segment
1 The CHNGDUMP routine obtains storage from subpool 229 for the work area.	IEEMB815	CHDINIT
2 CHNGDUMP performs a loop to check each option as set off by delimiters, as follows:		CHDCNTRL
• Scan the parameters for any delimiter, and then call the appropriate delimiter subroutine.		
• The delimiter subroutine determines whether the parameter is an option or a keyword. For keywords, the subroutine checks their validity; for options, control goes the option handler subroutine.		
• The option handler subroutine verifies the option and places it in the work area.		
3 If no errors occurred in the processing described in step 2, the CHNGDUMP routine sets the RTCT values as requested by the CHNGDUMP command.		CHDCDSS

Diagram 22-33. CHNGDUMP Routine (IEEMB815) (Part 3 of 4)



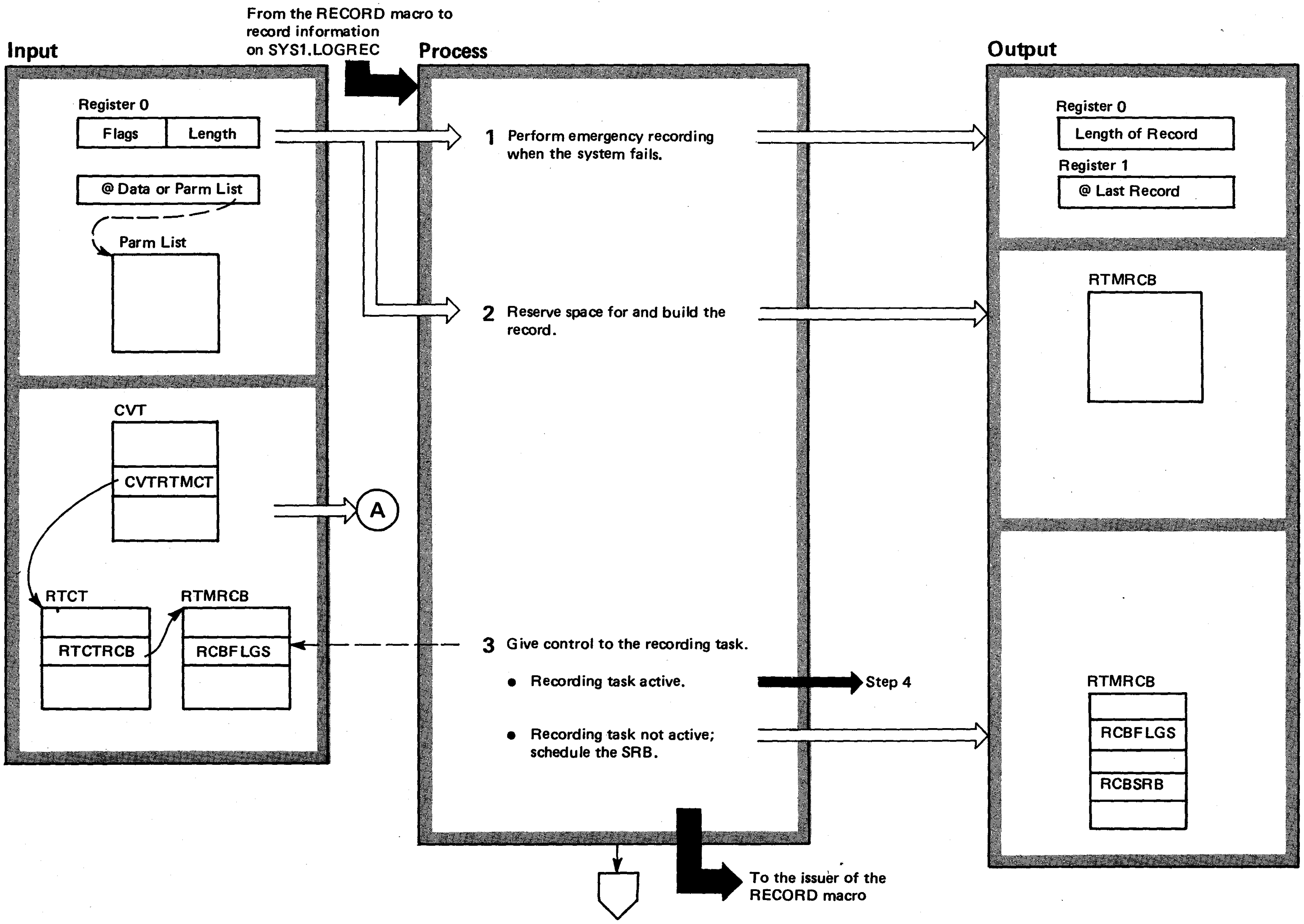
Process
Output
4 Put MSGINDEX in XAE and issue message.
5 Free work area storage.
6 Exit via branch register 14.
IEE0503D
IEAVGM00
FREEMAIN
XSA
XAE
To IEE0403D

Diagram 22-33. CHNGDUMP Routine (IEEMB815) (Part 4 of 4)

Extended Description	Module	Segment
4 The message index goes into the XAE field of the XSA (extended save area). Then, the CHNGDUMP routine uses the SVC 34 message module, module IEE0503D, to print the message. The message states that either the CHNGDUMP request was accepted or rejected.	IEE0503D	IEE0503D
5 The CHNGDUMP routine then frees the CHNGDUMP work area via the FREEMAIN service.	IEEMB815	IEEMB815
6 Control returns to the caller, module IEE040D, via a BR 14.		IEEMB815

Errors which occur during CHNGDUMP processing are handled by the SVC 34 ESTAE routine (module IEE5103D).

Diagram 22-34. Recording Processing (IEAVTRER) (Part 1 of 4)



From the RECORD macro to record information on SYS1.LOGREC
Input
Register 0
Flags
Length
@ Data or Parm List
Parm List
CVT
CVTRTMCT
A
RTCT
RTCTRCB
RTMRCB
RCBFLGS
Process
1 Perform emergency recording when the system fails.
2 Reserve space for and build the record.
3 Give control to the recording task.
• Recording task active.
Step 4
• Recording task not active; schedule the SRB.
To the issuer of the RECORD macro
Output
Register 0
Length of Record
Register 1
@ Last Record
RTMRCB
RTMRCB
RCBFLGS
RCBSRB

Diagram 22-34. Recording Processing (IEAVTRER) (Part 2 of 4)

Extended Description	Module	Segment
Recording processing writes the records that R/TM creates in the course of its processing. Recording processing builds the record in the RTMRCB (RTM record control block) to contain recording information. Then, the recording task writes the record, via SVC 76.		
Recording processing consists of two separate modules; the recording request routine which builds the record in the RTM RCB; and the recording task, which actually writes the record. The recording request routine receives control after a system routine issues the RECORD macro instruction. This routine uses the input information to build the record. After this routine finishes its processing, it gives control to the recording task. The recording task receives control when the recording request routine schedules an SRB (service request block), which posts the ECB the recording task is waiting on.		
R/TM creates records for hardware and IBM-software errors when requested by ESTAE routines or FRRs.		

Extended Description	Module	Segment
1 The recording request routine first determines whether the caller is the system termination routine. In this case, it returns the address of any records directed to SYS1.LOGREC to the caller. This means that WTO (write-to-operator) records are lost.	IEAVTRER	
The recording request routine places a return code of 4 in register 15 if no records remain to be written to SYS1.LOGREC.		
2 The recording request routine reserves the storage necessary to build the record in the RTMRCB. It then constructs a record header with the recording information from the parameter list.		FINDSPCE
3 The recording request routine can now give control to the recording task. The recording task receives control, to asynchronously write the records. The recording request routine schedules an SRB to post the waiting recording task.		SCHEDSRB
The recording request routine returns control to the caller who issued the RECORD macro instruction.		

Diagram 22-34. Recording Processing (IEAVTRER) (Part 3 of 4)

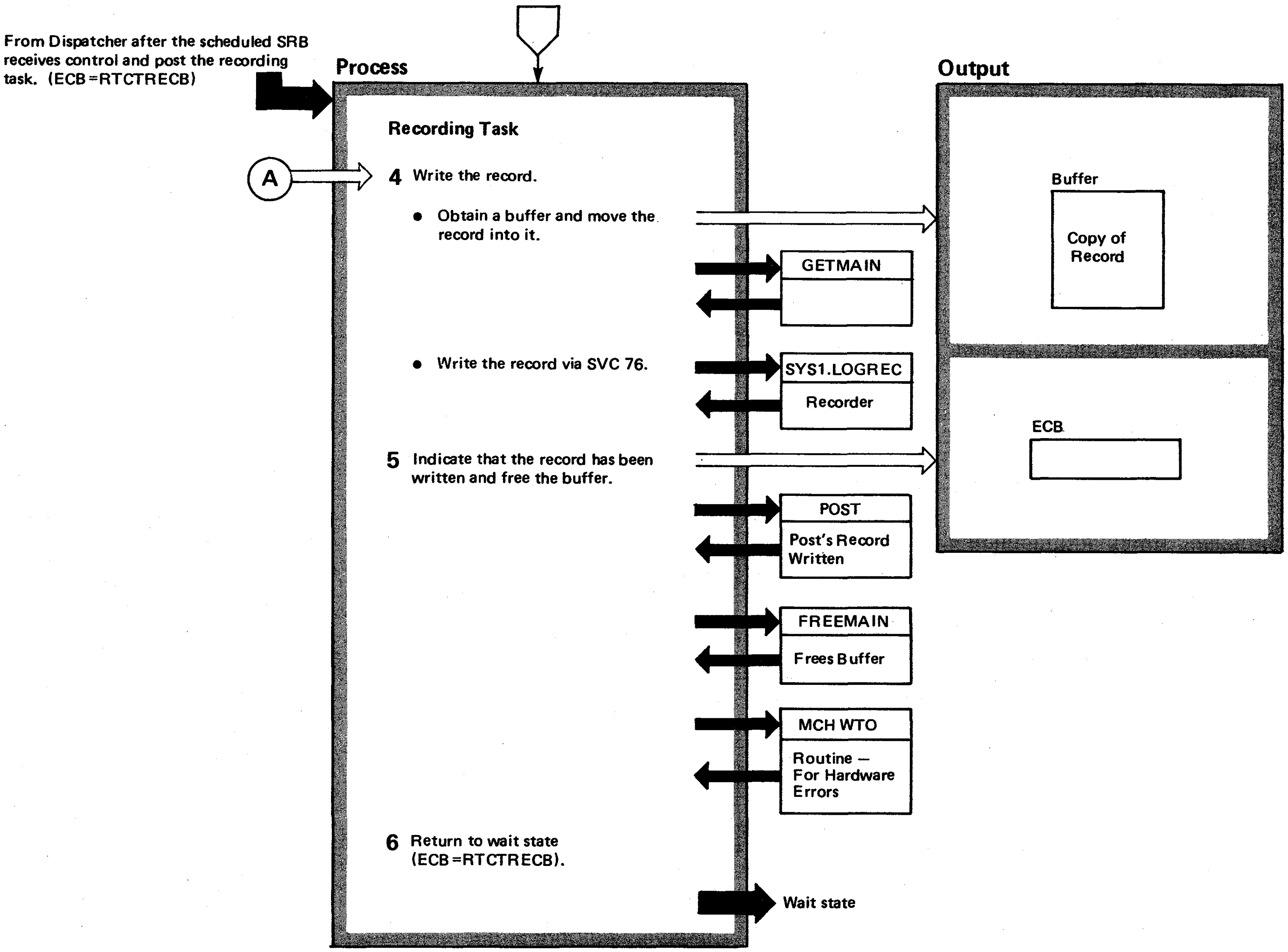

From Dispatcher after the scheduled SRB receives control and post the recording task. (ECB=RTCTRECB)
Process
Output
Recording Task
A
4 Write the record.
• Obtain a buffer and move the record into it.
• Write the record via SVC 76.
5 Indicate that the record has been written and free the buffer.
6 Return to wait state (ECB=RTCTRECB).
GETMAIN
SYS1.LOGREC
Recorder
POST
Post's Record Written
FREEMAIN
Frees Buffer
MCH WTO
Routine – For Hardware Errors
Wait state
Buffer
Copy of Record
ECB

Diagram 22-34. Recording Processing (IEAVTRER) (Part 4 of 4)

Extended Description	Module	Segment
4 The recording task first obtains a record buffer by issuing GETMAIN for storage equal to the length of the RTMRCB, and moves all records into this buffer.	IEAVTRET	REBUF
The recording task gives control to SVC 76 to actually write the records from the buffer to SYS1.LOGREC.		WRITERCD
5 The POST routine posts that the record has been written, if requested.		POSTER
The recording task then frees the buffer obtained in step 4.		
For all records written, the recording task gives control to the MCH (machine check handler) WTO routine.		
The MCH WTO routine then determines whether to write a message to the operator. In all cases, however, the MCH WTO routine notifies the operator for hardware errors.		
6 The recording task returns to the wait state to wait to be posted again.		

Index

INDEX

OS/VS2
System Logic Library
Volume 4
SY28-0764-0

READER'S COMMENT FORM

Your views about this publication may help improve its usefulness; this form will be sent to the author's department for appropriate action. Using this form to request system assistance or additional publications will delay response, however. *For more direct handling of such requests, please contact your IBM representative or the IBM Branch Office serving your locality.*

Possible topics for comment are:

Clarity Accuracy Completeness Organization Index Figures Examples Legibility

Cut or Fold Along Line

What is your occupation? ______________________________

Number of latest Technical Newsletter (if any) concerning this publication: ______________

Please indicate your name and address in the space below if you wish a reply.

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments.)

SY28-0764-0

Cut or Fold Along Line

OS/VS2 System Logic Library Volume 4 (S370-36) Printed in U.S.A. SY28-0764-0

Your comments, please . . .

This manual is part of a library that serves as a reference source for system analysts, programmers, and operators of IBM systems. Your comments on the other side of this form will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Fold

Fold

First Class
Permit 81
Poughkeepsie
New York

Business Reply Mail
No postage stamp necessary if mailed in the U.S.A.

Postage will be paid by:

International Business Machines Corporation
Department D58, Building 706-2
PO Box 390
Poughkeepsie, New York 12602

Fold

Fold

IBM®

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)